Dynamic Selection of Tasks and Methods as a Knowledge Level Reflective activity

Zoltán Istenes, Pierre Tchounikine, Francky Trichet
IRIN
2, Rue de la Houssinière
44072 Nantes Cedex 03
FRANCE
E-Mail : Tchou@irin.univ-nantes.fr

Abstract

This paper presents a model and an architecture for the Dynamic Selection of Tasks and Methods (DSTM) that enhances flexibility and genericity. Two aspects of DSTM have been modelled explicitly. First, the selection of tasks and methods. Second, how the selection process is achieved. These activities are modelled as knowledge level reflective activities. We present our modelisation of DSTM, the advantages of this approach and elements of comparison with similar work.

Keywords

Knowledge Representation, Knowledge Level Reflection, Conceptual Model, Dynamic Selection

1 Introduction

Many Knowledge Based Systems (KBS) model in some manner strategic knowledge that decides which task is the next more appropriate in a given situation (activity often referred to in the literature as "Reactive Planing"). Different publications have also pointed out that providing different possible methods to solve a task emphasises capacities such as flexibility or robustness [1,3,5,6,8]. Given a task, the system dynamically selects, at run-time, what method should be used. This selection can be achieved according to the current context of solving and/or to some meta-level decisions (time constraints, user preferences, etc.). Different languages have attempted to address such a selection, such as TroTelc [8], Lisa [3], Tips [5] or MML [1] (in this last reference a comparison of these different languages can be found).

This paper presents a model and an architecture for the dynamic selection of tasks and the dynamic selection of methods (which will be referred to as DSTM). The motivation of this work is to enhance flexibility and genericity.

Our general approach consists of an explicit representation of tasks and methods and control of solving modelled by reasoning on the representation of tasks and methods. It requires that a part of the system (the module that decides what task and what method must be selected) inspects another part of the system (the description of tasks and methods). This activity can thus be considered as what has been called in the Esprit "Reflect" project a knowledge level reflective activity [7].

The classical definition of a reflective system is: a system that can analyse itself. In the context of second generation KBS, reflection is mainly used as a means to enhance the capacities of the system. The initial definition is extended to architectures where a part of the system reasons on another part, or where a system (the reflective-system) reasons on another system (the object-system). In respect to these definitions, we call object-system the layer where we model tasks and methods and reflective-system the layer where we model the reasoning on task and method characteristics[1].

Our approach to DSTM has been prototyped with Zola [2]. Zola is an operationalisation high-level language. It allows the definition of domain specific types of knowledge such as "tasks" or "methods" and the definition of operations such as "Select a method" or "Test if a task verifies some criteria". Zola also allows the representation of conceptual information, i.e. information that is not present (or not in an accessible way) in the representation of the knowledge types and operations, but is necessary to obtain a knowledge level representation of the object-system. Conceptual information can be represented as conceptual handles, i.e. information attached to a knowledge type or an operation. As an example, "what an operation does" or "how it proceeds" are two elements of conceptual information that can not easily be retrieved from the description of the operation. This information can be added to the operation with conceptual handles, and used for knowledge level reflection.

The Reflect project suggests that instead of a direct access to the object-system structures (the implementation), one should construct reflective-systems that reason on a model of the object-system. In this separate layer, dedicated to the objectives of the reflective-system, can be represented aspects that are no longer present in its implementation and/or cannot be retrieved from it. In our framework, we have not used a separate layer to represent the conceptual model of the object-system: the object-system contains an abstract representation of itself.

2 Dynamic Selection of Tasks and Methods (DSTM)

2.1 A simple model for DSTM

An abstraction of DSTM can be roughly presented as follows. The problem to solve is defined as a task. Solving a task can be performed by a method. A method can split a task into (sub)tasks or directly solve the task (the two types of methods are often denoted as Decomposition methods and Action methods). Different methods can be used to solve a task. Methods can be given characteristics such as application criteria, in what context it will be particularly performant, etc. These characteristics can be manipulated using reflective activities such as "Identify possible methods", "Choose the most favourable method", etc. Control over these activities is defined by a scheduler.

In this model there is no control structure over the decomposition of tasks into subtasks. An ordering can be introduced as explicit constraints in the description of the sub-tasks. As a method can split a task into several sub-tasks and different methods can be tested to solve a task, DSTM can be viewed as a search in an and/or tree.

2.2 Representation and operationalisation

A task (Cf. Figure 1) is defined by its results and the context in which it can be achieved. If known, methods that can achieve the task can be associated. In this case, one dissociates if the list of methods is exhaustive or not (if no or no more methods are explicitly defined other methods can be searched).

A method (Cf. Figure 2) is defined by its results, the context in which it can be activated and the knowledge required for its achievement. If known, the description of favourable contexts can be added.

[1]There is no general consensus about this definition of "reflection", that is called "meta-reasoning" in other work. We use the Reflect project terminology, that is the major recent contribution to the study of reflection in KBS.

Slots	Definitions
Name	*Name of the Task*
Expected Results	*Description of the expected results*
Input Context	*Description of the context in which the Task can be achieved*
Associated Methods	
Possible Methods	*If known, the exhaustive list of Methods that can realise the Task*
Known Methods	*If known, a non-exhaustive list of Methods that can realise the Task*
Characteristics of Resolution	*Preference on how to solve the Task (prefer the use of an Action Method or a Decomposition Method, etc.)*

Figure 1: Description of a task

Slots	Definitions
Name	*Name of the Method*
Results	*Description of what will be produced by the Method*
Input Context	*Description of the context in which the Method can be activated*
Requirements	*Description of the knowledge required for the execution of the Method*
Favourable Context	*Description of when the Method is particularly relevant*
Decomposition	*Explicit description of the decomposition of a Task into sub-Tasks (for decomposition Methods only)*
Realisation	*Description of how the Method is achieved (for action Methods only)*

Figure 2: Description of a method

The reflective activities required for DSTM are presented in Figure 3 (each activity is called a DSTM operation and is represented as a Zola operation).

Operation	Definition	Concerned Slots Task	Concerned Slots Method
Select a Task to achieve	*Select the most relevant Task in a set of not yet realized Tasks*	Input Context	None
Identify possible Methods	*Identify Methods that can achieve a Task*	Associated Methods Expected Results	Results
Identify applicable Methods	*Identify the activatable Methods from a set of Methods*	None	Requirements Input Context
Select a Method	*Select the most relevant Method to achieve a Task*	None	Favourable Context
Activate a Method	*Perform a Method*	None	Decomposition Realisation
Evaluate the state of a Task	*Evaluate if a Task has been achieved by a Method*	Expected Results	None

Figure 3: Reflective activities for DSTM

Figure 4 presents a basic algorithm to schedule these activities.

The architecture obtained is presented in Figure 5. The list of activities proposed in Figure 4 and the algorithm proposed in Figure 5 model one of the different possible ways of performing DSTM. Such an architecture can easily be extended within the same framework by defining other activities and/or other algorithms, with an additional level choosing what algorithm should be used.

The conceptual primitives we have constructed with Zola to represent and operationalise the model are the knowledge types "task" and "method", and the operations "Identify possible methods", "Choose the most favourable method", etc. The operationalisation process requires the definition of the intermediate and low level operations

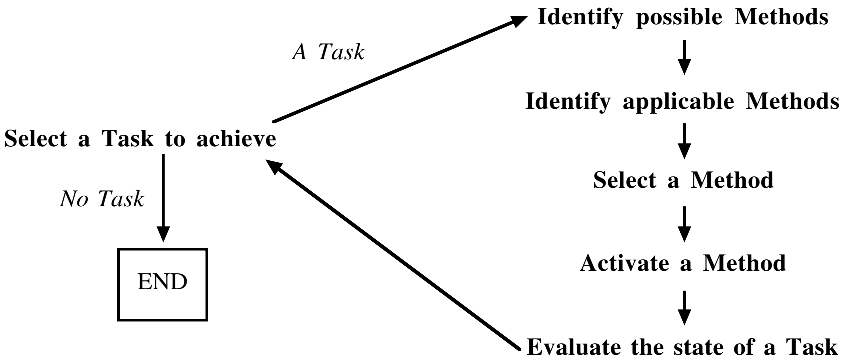


Figure 4: An algorithmic scheduler for DSTM

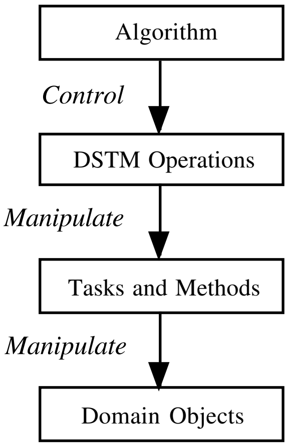


Figure 5: General architecture with an algorithmic scheduler

that implement the high-level operations. As an example, the high-level operation "Identify possible methods" consists of:

1. If possible methods have been defined for the task, the list is directly available.
2. If known methods have been defined and some still remain unused, the list is directly available.
3. If no method is directly defined as possible or the list of unused known methods is empty, a new list of methods is defined by comparing the expected results of the task with the results of the methods defined.

At the bottom-level, manipulations such as "Check if an object slot is empty" are implemented using Zola primitives.

2.3 A reflective approach to scheduling DSTM

2.3.1 Shortcomings of the previous approach

Although sufficient to perform DSTM, the scheduler presented in Section 2.2 has a number of shortcomings:

(i) The definition of the scheduler is model-specific.

(ii) The control of the scheduler is implicit and fixed. The DSTM activities are always considered in the same order, and there is no justification of why an activity is considered.

(iii) It is not possible to choose dynamically (at run-time) how a DSTM activity will be performed.

These shortcomings limit the genericity and the flexibility of the architecture.
For example:

- The scheduler directly manipulates operations defined according to the model, in this circumstance methods and tasks. Tasks and methods are instanciations of the more abstract notions of Objective and Means. An abstract scheduler could be used over object-systems based on some other instanciations.
- The same control is used for all the tasks and methods, and during all the solving. One cannot explicitly model aspects such as: "Some tasks must be absolutely solved at once, and when they are considered one should try all possible methods; some tasks should be re-considered if the selected method fails, and put back into the list of not yet realised tasks".
- The implicit ordering does not permit explanations of the system behaviour.
- The same operation is used to select a method for all the tasks and in all conditions. More flexibility could allow differentiation of the consideration of certain tasks, or to customise solving (for instance, define an alternative operation where the selection is made by the user, and use one or the other according to some criteria).

To overcome these shortcomings we have attempted to model the scheduler at a generic level and to provide more flexibility when defining what DSTM activity should be achieved next.

2.3.2 A Generic and Dynamic Scheduler (GDS)

The scheduler flexibility and genericity can easily be extended by its modelisation as a knowledge base of strategic rules (Cf. Figure 6). The rules specify what action should be undertaken at the DSTM level by the means of a connection layer.

If

Problem Solving Context

An Objective O_i is defined
A set of Means is defined
One of the Means has been tried
The Means failed

Strategy

It has been defined that one should saturate the realisation of the Objective O_i

Then

Action

Choose a new Means for the Objective O_i

Figure 6: A strategic rule

Such a modelisation addresses problems (i) and (ii). The architecture obtained corresponds to Figure 7.

In such an architecture a subsequent problem appears: the modelisation of the connection layer between the Generic and Dynamic Scheduler (GDS) and the DSTM level. In concrete terms, when the GDS asks for the action "Define the objective", in the context of the current DSTM model it must be interpreted as the activity "Select a task to achieve".

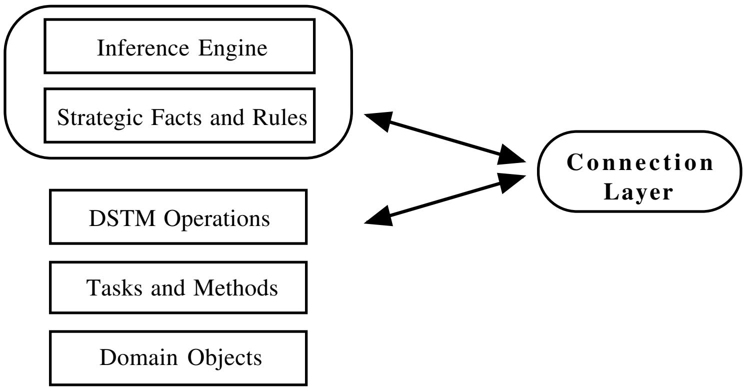


Figure 7: General architecture with a Generic and Dynamic Scheduler

2.3.3 The Connection Layer

The modelisation of the connection layer as a straightforward 1:1 translation between the vocabulary used in GDS and DSTM operations is an inadequate solution. It does not provide more flexibility for how the reflective activities will be performed. How the actions defined by the GDS should be translated into DSTM activities depends on the definition of these DSTM activities. For this reason, the connection layer has been modelled as a reflective activity. It analyses the DSTM operations and selects the more relevant. The high-level operations currently defined for this modelisation are presented in Figure 8. These operations are concerned with DSTM operations and knowledge internal to the connection layer.

Operation	Definition	Concerned Knowledge: Object-System	Concerned Knowledge: Connection System Internal Knowledge
Translate into a Logical Operation	*Translate the Action defined by GDS into a Logical Operation*	None	Dictionnary
Identify the possible Concrete Operations	*Identify a set of Object-System Operations that correspond to the Logical Operation*	Description of the Object-System Operation	Description of the Logical Operation
Select a Concrete Operation	*Select the most relevant Object-System Operation to achieve a Logical Operation*	Description of the Object-System Operation	Description of the Logical Operation
Activate a Concrete Operation	*Perform the concrete Object-System Operation*	None	None
Evaluate the results of a Concrete Operation	*Evaluate a concrete Object-System Operation*	Description of the Object-System Operation	None
Evaluate the results of a Logical Operation	*Evaluate the results of a Logical Operation*	None	Description of the Logical Operation
Translate the results of a Logical Operation	*Translate the results of a Logical Operation in GDS terms*	None	Dictionnary

Figure 8: Reflective activities for the connection layer

We describe here how this modelisation allows the addressing of problems such as (ii) and (iii).

(ii) With every DSTM task has been associated information on how it should be solved, i.e. “saturated” (try all methods until one succeeds) or “re-considered” (if the selected method fails, reconsider the task to achieve). This information is retrieved from the DSTM level by the connection layer and taken into account in the GDS. Accordingly in the context of DSTM, all possible methods will be tried or the task will be re-considered.

(iii) Multiple versions of the high-level operation "Select a method" have been defined (Cf. Figure 9). We will refer to such operations as concrete operations. One is defined previously (Cf. Figure 3), another interacts with the user. With every DSTM task has been associated information on preferences about the way methods should be selected ("select-by-criteria", "select-by-question"). When the connection layer considers the action defined by the GDS, it translates it into a logical operation that specifies the DSTM operation to achieve. With every logical operation is associated an evaluating operation that defines what concrete operation should be used. As an example, the logical operation "Select a method" has an evaluating operation that defines the possible DSTM operations, inspects the current task, and chooses one or the other of the concrete operations according to whether the preference is "select-by-criteria" or "select-by-question".

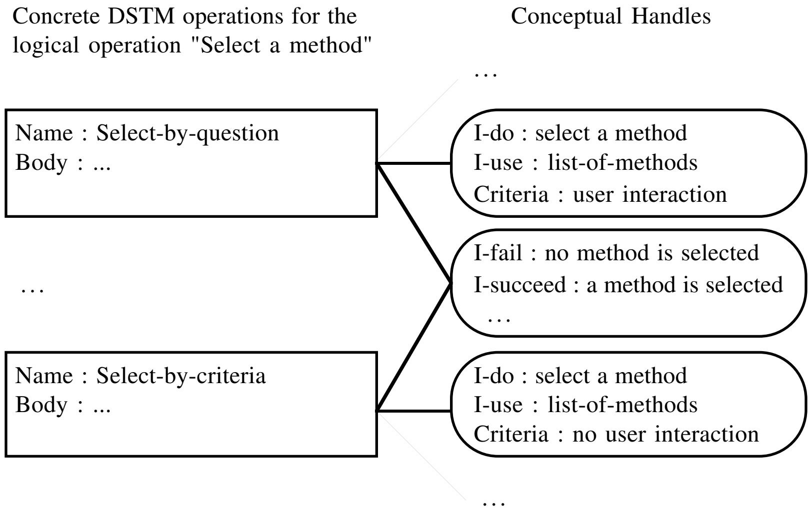


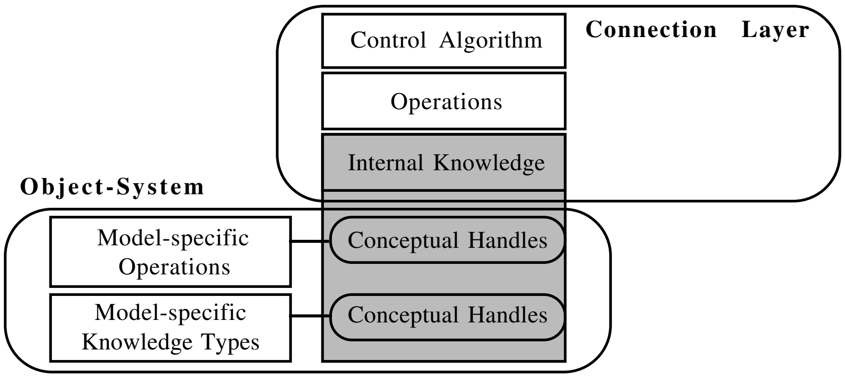
Figure 9: Multiple versions of a high-level operation

The information associated with tasks and DSTM operations is not necessary for DSTM, and thus had not been represented in the initial model. Information about the tasks ("saturate or re-consider", "select-by-criteria" or "select-by-question") can be added by modifying DSTM types of knowledge if one considers that these notions are part of the definition of a task. Information about what a concrete operation does (to be matched with logical operations) and how it proceeds has been modelled using conceptual handles[2] (Cf. Figure 9). Knowledge specific to the connection layer (as the logical and evaluating operations) is represented within this separate layer. The general architecture of the connection layer is presented in Figure 10.

The control of the connection layer (Cf. Figure 11) is a simple algorithm.

The definition of this algorithm is not straightforward. In particular, some choices have to be made about what should be tackled in this algorithm (i.e. by the connection layer) and what should be tackled at the Strategic Level. As an example, in

[2]If the additional information about tasks is not considered as part of the definition of a task, it can also be added as conceptual handles. This is the solution we have adopted (and therefore why conceptual handles connected to the domain layer appear in Figure 10), although we are not completly satisfied with it. This information is not conceptual information in the sense of "conceptual description of an object", but rather additional information for reflective purposes.



Operations from the connection layer manipulate knowledge internal to the connection layer and the conceptual handles attached to the model-specific conceptual primitives (operations and knowledge types).

Figure 10: General architecture with the connection layer

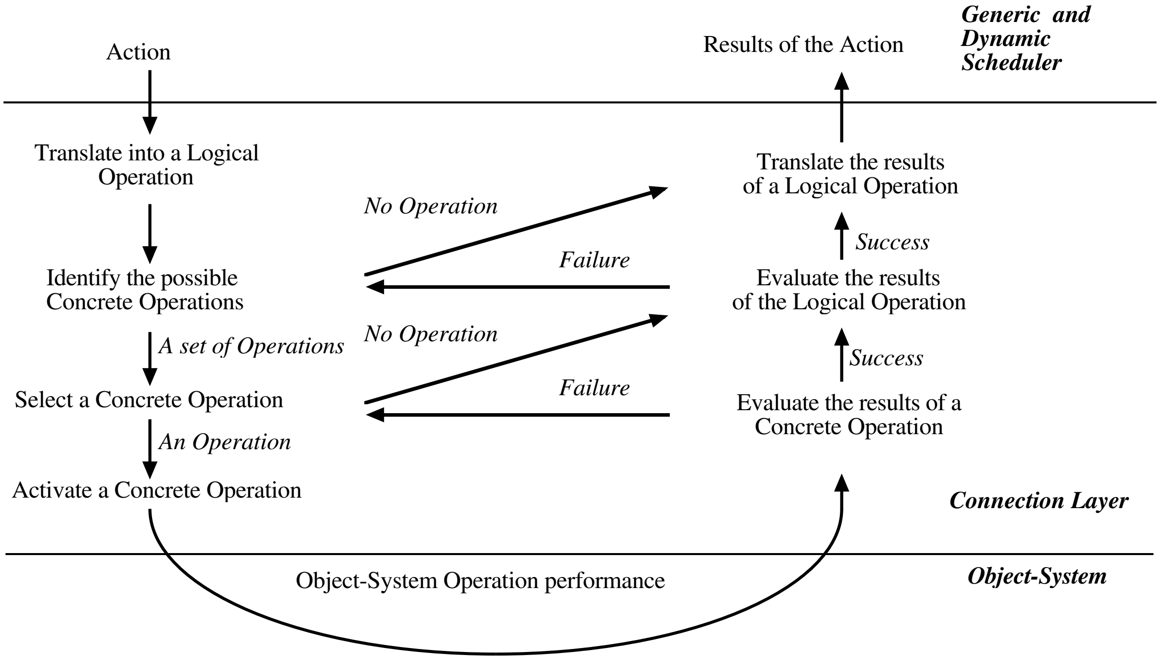


Figure 11: Algorithmic control of the connection layer

the algorithm presented in Figure 11, when a concrete operation corresponding to a logical operation does not succeed, other candidates are tried. In the same way, if a concrete operation succeeds but the logical operation is not considered to be completed, other concrete operations are considered. Such control could be made more flexible and more explicit by moving up these choices to the GDS or modelling more explicitly the connection layer.

3 Discussion

3.1 Advantages for DSTM

The advantages of the approach presented here for DSTM can be summarised as follows:

- Task and method definitions can easily be modified. This is an advantage in comparison with languages where task and method definitions are fixed by the syntax of the language. The representation and operationalisation of the model with Zola enables modification of any part of the model. For example, it can appear necessary to modify how the selection of the most pertinent method is made. This can be undertaken by modifying the description of tasks and methods (addition of new slots) and/or modification of the operations that perform the selection.

- Task and method conceptual descriptions can be extended with conceptual handles, without modifying their definition.
- Tasks and methods are selected at run-time, according to explicit criteria. This is obtained by the reflective activities defined at the DSTM level.
- The way the DSTM activities are controlled can be defined at run-time, according to explicit criteria. This includes the choice at run-time of the operations that select tasks and methods. This is obtained by the dissociation of logical operations and concrete operations and the reflective activities of the connection layer.
- The scheduler's strategy is explicit, flexible, dynamic (the next DSTM activity is defined at run-time), and modelled at a generic level. The fact that the scheduler's strategy is dynamic allows in particular consideration of idiosyncratic notions of the object-system (in this case, if tasks are saturated or not).
- The strategy can be used over object-systems based on different modelisations. This is obtained by the means of the connection layer.

Another benefit of this architecture is that it defines a framework for reflective activities other than control such as "Check if the problem has a solution" or "Verify the object-system consistency". These activities can be modelled in a similar way to DSTM, and the GDS and the connection layer can be extended to control them.

3.2 Problems addressed at a superficial level

A certain number of conceptual problems have been raised and are addressed at a superficial level in our current prototype. Problems currently under deeper investigation are:

- We have supposed that there is a 1:1 correspondence between actions of the strategic level and logical operations of the connection layer. This can easily be extended to a 1:n correspondence. But a 1:0 correspondence is an example of a list of interesting problems (coherence within the DSTM level and between the DSTM level and the connection layer, consistency, solvability, etc.).
- A more accurate analysis of the connection layer, and in particular of what can be defined in generic terms and what is necessarily object-system specific, is necessary.
- We have mainly focused our work on architectural and technical features. The concepts (vocabulary, operations specification) used in the current prototype have not been sufficiently validated from an epistemological point of view. We also consider that our current conception of conceptual handles should be further refined.

3.3 Comparison with the Reflect project

From an architectural point of view, the Reflect project [7] suggests the use of a separate representation of the model of the object-system. The underlying idea is that one can not retrieve knowledge level information from an operational system. A separate representation, that is not defined to perform reasoning (that has not to be "operational") can be made "at the knowledge level". Within our architecture, we can say that the object-system contains an abstract representation of itself. All reflective activities presented previously are based on manipulations of what is used to represent a model[3].

Reflect also suggest the addition of a reflective system to already existing systems. This supposes that one adds some knowledge (in a separate layer) to the existing system for the purposes of reflection. Not surprisingly, this knowledge corresponds to that we have used: for every action of the object-system, definition of the goal, inputs and outputs, requirements, conditions of applicability, assumptions that must hold for the action to apply and cost characteristics [6]. In our prototype, all the knowledge needed for reflection is part of task and method definitions because we have modelled the object-system with the intention of enabling reflection. Our work can be generalised to be used on an already existing system by switching this knowledge to conceptual handles attached to the object-system notions.

4 Conclusion

We have presented a knowledge level reflective approach to dynamic selection of tasks and methods, the advantages of this approach, and how it can be operationalised using Zola. The model presented here has been elaborated by testing and refining successive versions. This method, advocated in [4], is made possible by the use of a high level operationalisation language. We are currently modifying this model to adapt it to the particular needs of an Intelligent Tutoring System.

References

[1] Guerrero-Rojo V., *"MML, a modelling language with dynamic selection of methods"*, in Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop (Banff'95), Banff, Canada, 1995.

[2] Istenes Z., Tchounikine P., *"Zola: a Language to Operationalise Conceptual Models of Reasoning"*, to appear in International Conference of Computing and Information (ICCI'96), University of Waterloo, Ontario, Canada (June 1996).

[3] Jacob-Delouis I., Krivine J.P., *"LISA: un langage réflexif pour opérationaliser les modèles d'expertise"*, in Revue d'Intelligence Artificielle, Vol.9 N°1, pp 53-88, 1995.

[4] Karbach W., Voß A., *"Model-K for prototyping and strategic reasoning at the Knowledge Level"*, in Second Generation Expert Systems, David J.P., Krivine J.P., Simmons R. (editors), pp721-745, Springer-Verlag, 1993.

[5] Punch W.F, Chandrasekaran B., *"An Investigation of the Roles of Problem-Solving Methods in Diagnosis"*, in Second Generation Expert Systems, David J.P., Krivine J.P., Simmons R. (editors), pp673-688, Springer-Verlag, 1993.

[6] Reinders M., Bredeweg B., *"Strategic reasoning as a reflective task"*, International Workshop on Reflection and Meta-Level Architecture, Tama-City, Tokyo, 1992.

[7] van Harmelen F., Wielinga B., Bredeweg B., Schreiber G., Karbach W., Reinders M., Voß A., Akkermans H., Bartsch-Spörl B., Vinkhuyzen E., *"Knowledge-Level Reflection"*, in Enhancing the Knowledge Engineering Process, Contributions from ESPRIT, B. Le Pape and L. Steel editors, Elsevier Science, pp 175-204, Amsterdam, The Netherlands, 1992.

[8] Vanwelkenhuysen J., Rademakers P., *"Mapping a Knowledge Level analysis onto a computational framework"*, in Proceedings of ECAI'90, pp661-666, 1990.

[3] It could be argued that the addition of conceptual information almost corresponds to the definition of a separate knowledge level representation.