Verifying and Validating a Task/Method Knowledge-Base

Francky TRICHET & Pierre TCHOUNIKINE
IRIN
Université de Nantes & École Centrale de Nantes
2, rue de la Houssinière - BP 92208
44322 Nantes cedex 03 FRANCE
trichet@irin.univ-nantes.fr

Abstract

In this paper, we present how verification and validation tools can be of effective help when constructing the problem-solving model of a Knowledge-Based System (KBS) in the context of the Task/Method modeling paradigm. The problem-solving model of a KBS defines how expert-knowledge should be modeled and organised in order to solve problems. Constructing such a model in the context of the Task/Method paradigm consists in adopting a Task/Method Knowledge Representation Language (KRL) and using it to express a Problem-Solving Method. DSTM is a flexible framework that allows prototyping the problem-solving model of a KBS. The prototyping process concerns both the custom-built KRL and the PSM expressed within this KRL. Prototyping the KRL aims at constructing a satisfactory high-level Task/Method language which is both a modeling language and an implementing language. Prototyping the PSM aims at defining the PSM itself (i.e., defining the Task/Method knowledge-base). In order to support such an approach, DSTM provides verification and validation tools. DSTM verification tools aim at checking if the Task/Method knowledge-base respects some given modeling constraints. DSTM validation tools aim at checking if the prototype fits the needs, i.e., if the current problem-solving model allows emulating the expected problem-solving behaviour.

1 Introduction

Second generation KBSs have popularised modeling problem-solving in terms of tasks and methods [4]. Problems and sub-problems are modeled as tasks and possible means to solve a problem are modeled as methods. Such a modeling allows the definition of different means to reach an objective (different methods for a task) and the expression of static strategies (static decomposition of tasks into sub-tasks) or opportunistic strategies (dynamic selection of tasks and methods at run-time, according to the problem-solving context). Using task and method notions appears as an adapted way for describing Problem-Solving Methods[1] at an abstract implementation-independent level. Although with some differences, the task and (for some of them) method modeling primitives are used as a high-level language in works such as the Generic Tasks Theory [2], Commet [16], the Task framework [14] or the works of [1].

Constructing a KBS within the Task/Method modeling paradigm consists in (1) adopting a Knowledge Representation Language (i.e., a language that proposes task and method modeling primitives and selection mechanisms) and (2) defining the knowledge-base according to the adopted KRL (i.e., defining the effective tasks and methods by using the current KRL as a modeling guide). This two-step process can possibly be performed in a sequential way. However, a lot of modeling problems generally appear when confronting the adopted representation language with the expert-knowledge. Therefore, a prototyping approach where several versions of the couple KRL/knowledge-base can be tested and refined until fixing the final KRL is preferable. In order to enable such a process, we have constructed DSTM (*Dynamic Selection of Tasks and Methods*), a flexible framework that allows prototyping the problem-solving model of a KBS in the context of the Task/Method paradigm [17]. The prototyping process concerns both the current custom-built KRL and the PSM expressed within this KRL. Prototyping the KRL aims at constructing a satisfactory high-level Task/Method language

[1] In this paper, a Problem-Solving Method (PSM) is a way to solve problems. It is not necessarily a generic PSM, i.e., a way to solve a type of problem that can be used over different domains. The description of a PSM at an abstract level is a description of what actions are to be performed and how, at a level that ignores implementation issues.

which is both a modeling language and an implementing language. Prototyping the PSM aims at defining the PSM itself (i.e., defining the Task/Method knowledge-base that allows emulating the expected problem-solving behaviour).

The flexibility provided by DSTM allows the modeling-team[2] to model all the relevant specificities of the studied expertise. However, as one can modify the KRL (i.e., the modeling primitives and the selection mechanisms), the complexity of the KBS construction is increased. In order to help the modeling-team to deal with this complexity, we have enhanced the DSTM framework with verification and validation tools.

DSTM verification tools are concerned with checking if the knowledge-base respects some given modeling constraints (evaluation of the *structure/form* of the KBS in order to answer the question "Am I building the system right?" [10]). These tools can (1) check if the domain tasks and methods respect some syntactical constraints defined on the task and method modeling primitives, (2) analyse the influence of missing knowledge (e.g., an instance with an omitted slot value) according to how this knowledge can be manipulated by the selection mechanisms and (3) check if the domain tasks and methods respect some integrity constraints (e.g., "*each task must be associated with at least one method*") explicitly defined by the way of a graphical language based on the Entity-Relationship model.

DSTM validation tools are concerned with checking if the prototype fits the needs, i.e., if the model allows emulating the expected problem-solving behaviour (evaluation of the *substance/content* of the KBS in order to answer the question "Am I building the right system?" [10]). These tools aim at providing the modeling-team with a synthetic understanding of how, given the current selection mechanisms, the tasks and methods of the knowledge-base can interact. In other words, DSTM validation tools aim at explaining the consequences of the current confrontation knowledge-base/custom-built KRL, consequences that the modeling-team cannot simply retrieve by itself [18]. Each validation tool is concerned with an explanation objective which emphasises an interesting Task/Method modeling feature that requires a complex process to be presented (e.g., "*Analyse how the achievement of tasks preceding T_i influences the selection of a method for T_i*" or "*Analyse how the achievement of T_i influences the selection of methods for the following tasks*"). Therefore, we call what these tools calculate "explanations dedicated to the modeling-team".

The characteristic of the DSTM verification and validation tools is that they are meta-tools in the sense introduced in [5]. They are based on a reflective analysis of the problem-solving model explicitly represented in the prototype and thus adapt themselves to the adopted KRL. Therefore, they can be used without further programming work.

The remainder of this article is organised as follows. In order to keep the paper self-content, we first present the architecture of the DSTM kernel (i.e., the operational kernel that allows constructing the Task/Method knowledge-base and the underlying custom-built KRL). More details about this kernel can be found in [17]. In section 3 and section 4, we present (respectively) the DSTM verification tools and the DSTM validation tools. These sections are illustrated with examples from the construction of the KBS embedded in the Emma educational system [3]. In section 4, we highlight the interest of building verification and validation tools based on a reflective analysis of the couple knowledge-base/KRL, we discuss the scope of our approach and we compare it with related work.

2 Constructing a KBS within the DSTM framework

2.1 Modeling principles

The DSTM framework is based on a structuration of the Task/Method modeling paradigm in four layers: the modeling primitives, the abstract notions, the high-level actions and the control. This four-layer architecture is associated with a limited-interaction principle: the control is defined on the high-level actions, a high-level action is based on an abstract notion and an abstract notion denotes a possible state of a task or a method during resolution.

The following abstract notions have been retained: *applicable Task* (a task that can possibly be achieved), *achieved Task* (a task that has been achieved by a method and whose objective is attained), *unsuccessfully considered Task* (a task for which all the possible methods have been considered and none of them allows it to be achieved), *pending Task* (a task which is neither an *achieved Task* nor an *unsuccessfully considered Task*), *candidate Method* (a method that can achieve a task), *applicable Method* (a method that can be fired) and *favourable Method* (a method that is particularly relevant).

These abstract notions are used as selection criteria by the high-level actions that perform the different selection mechanisms. The following high-level actions have been retained:

[2] The *modeling-team* is composed of one or several domain-expert(s) and one or several knowledge-engineer(s).

- *Select a task*: selects a task that can be achieved from a set of not yet achieved tasks. This action is based on the *applicable Task* abstract notion.
- *Identify candidate methods*: identifies the methods that can achieve a task. This action is based on the *candidate Method* notion.
- *Identify applicable methods*: identifies the methods that can be used from a set of methods. This action is based on the *applicable Method* notion.
- *Select a method*: selects a method from a set of methods. This action is based on the *favourable Method* notion.
- *Evaluate the state of a task*: evaluates the state of a task after the activation of a method. This action is based on the *achieved Task*, *unsuccessfully considered Task* and *pending Task* abstract notions.

Different types of control (e.g., static decomposition of tasks or dynamic selection of tasks and methods) can be defined over these high-level actions. As an example, a dynamic selection of tasks and methods can be modeled by a simple (and intuitive) sequential algorithm over all the high-level actions.

2.2 The DSTM operational kernel

The architecture adopted for the DSTM kernel respects the modeling principles underlying the DSTM framework and therefore is a four-layer one. It has been implemented above the Zola language [9]. This language provides a set of primitives which allows an explicit representation of the modeling primitives and the underlying selection mechanisms (i.e., the abstract notions and the high-level actions).

The modeling primitives level

The basic definitions that have been retained for the task and method primitives are as follows. A task is defined by its results and the context in which it can be achieved. If known, methods that can achieve it can be associated. A method is defined by the results it produces, the context in which it can be fired and the knowledge required for its achievement. If known, the description of when it is particularly relevant can be added.

These definitions are explicitly represented in the kernel as Zola knowledge-types. These knowledge-types are structured in an inheritance hierarchy and are described by a set of slots (e.g., *Expected-Results*, *Input-Context* and *Associated-Methods* for the task knowledge-type). Each slot can be associated with syntactical constraints. These constraints are concerned with the *mandatory filling* of the slot (the slot S_i has to be filled in when defining a new instance of the knowledge-type), the *type of the slot* (the slot S_i has to be filled in with a *SET-OF* instances of a particular type, with a *Boolean* value, etc.) and the *domain* of the slot (the slot S_i has to be filled in with a value belonging to a list of predefined ones or to an interval).

What makes DSTM kernel originality is that these structures can be customized according to the definition adopted for the considered expertise.

For instance, in Emma[3], the basic definition of a task is not totally satisfactory. First, the relevance of a task is defined by taking into account aspects related to mathematical constraints as well as aspects related to the PSM that is to be taught. Second, some interpretation knowledge must be attached to every task. Such knowledge is used to interpret (at an abstract level) the results of a task once it is achieved. One dissociates necessary interpretations (interpretations that *must* be considered when the task is achieved) from possible interpretations (interpretations that *can* be considered when the task is achieved). Finally, some tasks are decomposed into sub-tasks but, at the moment this decomposition is achieved, one cannot yet define what these tasks effectively are. For instance, at the first step of a resolution, the task "Deal with an exercise" is decomposed into "Analyse the problem", "Formalise the problem", "Solve the problem" and "Examine the results". When this decomposition is achieved, one cannot yet know if "Formalise the problem" corresponds to the task "Formalise an optimisation problem" or to the task "Formalise a statistic problem", as this depends on the results of the task "Analyse the problem".

These differences lead to the adaptation of the basic definition of a task as adopted in DSTM kernel. This is achieved by modifying some of the slots (e.g., *Input-Context* is changed into *Activation-Context* that denotes the pertinence from the mathematical point of view, and into *Pre-Conditions* and *Resources* that denote the relevance from the taught method point of view) or adding some new slots (e.g., *Necessary-Interpretations* and

[3]Emma is an educational system that aims at training students in the practice of linear programming as a technique to solve concrete problems (for example economic problems). In order to be able to solve problems and to analyse students' resolutions, Emma embodies a KBS whose problem-solving model has been defined within the DSTM framework [3].

Possible-Interpretations). In order to respect the teacher's vocabulary, a task has been renamed an activity. Two specialisations of the activity primitive have been defined: the *prototype activity* primitive (activities that are planned but cannot yet be explicited and therefore do not have associated methods) and the *concrete activity* primitive (effective activities that can be associated with methods). During resolution, when a prototype activity is selected, the concrete activity to be used to "instantiate" it is defined according to the context.

The abstract notions level

The basic definitions that have been retained for the abstract notions are founded on the basic definitions that have been retained for the modeling primitives. For instance, the definition adopted for the abstract notion *candidate Method* is: "a method M_i is a *candidate Method* for the achievement of a task T_i if and only if M_i has explicitly been defined as achieving T_i or if M_i produces the results that are expected for T_i".

These abstract notions are represented as Zola operations defining a logical connector (*and*, *or*, *not*) over a set of sub-operations. Sub-operations can be expressed as the application of predefined primitives (e.g., *check-domain-knowledge*, *belongs-to* or *subset*) over the different slots of the modeling primitives. As an example, figure 1 presents the Zola operation used to represent the *candidate Method* abstract notion.

```
(operation     'name                'candidate-Method
               'profile             'or
               'parameters          '(method-1  task-1)
               'body
               '(
                          (belongs-to method-1 'NAME task-1 'ASSOCIATED-METHODS)      OP1
                          (subset task-1 'EXPECTED-RESULTS method-1 'RESULTS)         OP2
               )
)
```

A Zola operation is constructed from a set of predefined profiles that correspond to control structures (e.g. *While* or *If-Then-Else*), logical connectors (e.g. *And* or *Or*) or manipulation primitives (e.g. *Match, Belongs-to* or *Subset*). Thus, an operation is an instance of a predefined profile. The operation `candidate-Method` is an instance of the profile *Or* (text in bold font corresponds to the syntax of the Zola language). It is defined as a disjunction (`or` profile) of two sub-operations, *OP1* and *OP2*. It takes as input a Method `method-1` and a Task `task-1` (`method-1` and `task-1` are formal parameters) and returns 'true' if one of *OP1* or *OP2* returns 'true'. *OP1* checks if the name of `method-1` belongs to the list of methods associated to `task-1`. *OP2* checks if the results expected for `task-1` are a subset of the results produced by `method-1`.

Figure 1: Representing an abstract notion in Zola

According to the modifications that have been made at the modeling primitive level, the basic definitions of the abstract notions must be adapted and/or some others must be created. Such modifications can easily be made through the graphical interface provided by the DSTM kernel. For instance, in Emma, the *applicable Task* abstract notion must be modified into the *applicable Activity* abstract notion. An activity A_i is an *applicable Activity* if it satisfies the mathematical constraints (modeled by the slots *Activation-Context* and *Resources*) and the constraints related to the taught method (modeled by the slot *Pre-Conditions*). First, two support notions[4] have been defined, `verify-mathematical-constraints activity` and `verify-taught-method-constraints activity`. Then, the *applicable Task* abstract notion has been modified by changing its logical connector and its sub-operations (cf. figure 2).

The high level actions level and the control level

The high-level actions correspond to selection mechanisms based on criteria that are denoted by the abstract notions. These actions are represented as control operations over the operations that implement the abstract notions. For instance, the operation `Identify-applicable-methods` takes as input a set of methods M1 and returns as output the set of methods M2 that can be fired, i.e., that respect the criterion denoted by the abstract notion *applicable Method* (represented by the Zola operation `applicable-Method`). This is explicitly represented in Zola by an operation whose body is: `match set-of-methods set-of-applicable-methods applicable-Method`. The primitive (`match set1 set2 criterion`) constructs `set2` by selecting the items from `set1` that respect `criterion`.

The modification of the abstract notions can require some detailed adaptations of some of the high-level actions and/or the definition of new high-level actions. All the high-level actions have the same structure, their differences only stand in their signature and the abstract notions they manipulate. For the DSTM high-level actions that are reused in Emma (such as *Select an applicable task*), the only modification is to introduce the Emma abstract notions. This is a direct advantage of the limited-interaction principle. For new actions such

[4]When defining an abstract notion, predefined primitives (e.g., *check-domain-knowledge*, *belongs-to* or *subset*) and support notions can be used. The interest of defining a support notion can be to simplify the description of an abstract notion and/or to reify a pertinent aspect of the modeling (as abstract notions do). In the present case, these support notions have been introduced because they correspond to modeling notions.

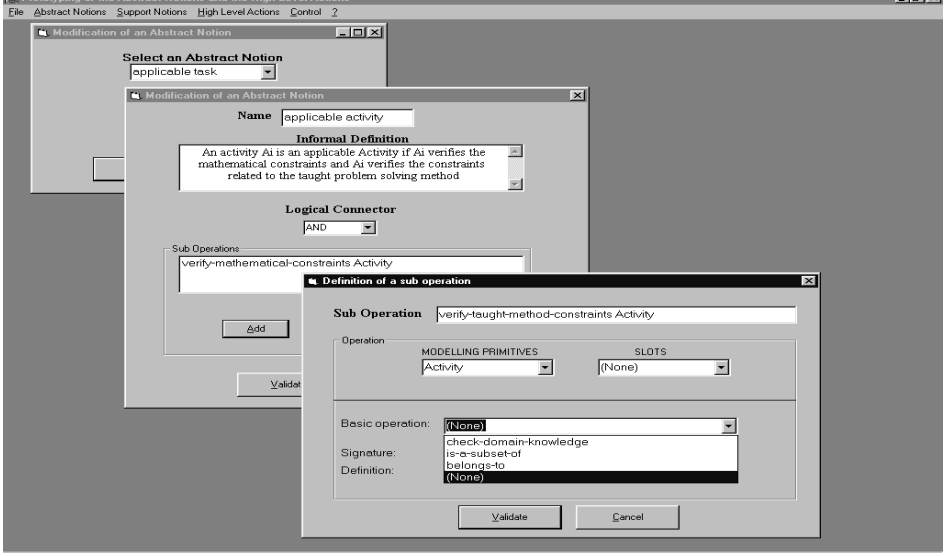


Figure 2: Modifying the *applicable Task* basic definition into *applicable Activity*

as *Identify possible concrete activities*, the high-level actions directly proposed by the kernel can be used as patterns.

The control over the high-level actions is represented by a Lisp function which performs a sequential and iterative launching of the high-level actions (see [17] for more details).

2.3 The DSTM kernel: assessment

DSTM operational kernel allows the modeling-team (1) to define knowledge-types used to represent the definitions adopted for the task and method modeling primitives, (2) to define operations used to implement the selection mechanisms (i.e., the abstract notions and the high level actions) and (3) to construct the Task/Method knowledge-base (instantiation of the task and method knowledge-types). This kernel has been developed above the Zola langage [9]. Zola is not the only language that can be used to implement the DSTM kernel, but it proposes a set of properties that are essential for implementing such a kernel, in particular:

- The capacity to construct and modify representation structures that correspond to modeling primitives.
- The capacity to construct operations that explicitly represent the definition adopted for the abstract and support notions. The explicitness is an important point. If the encoding of an abstract notion or a support notion is a black-box or is defined with low-level features, these notions cannot be easily modified nor analysed by reflective modules.
- The capacity to construct Zola operations that can analyse other Zola operations[5].

These properties allow constructing a prototype which reifies the problem-solving model. The modeling primitives, the abstract notions and the high-level actions are explicitly represented in the implementation. Such explicitness makes the evolution of these notions easy and straightforward. Moreover, the capability to construct operations that can analyse other ones allows constructing reflective modules that can analyse the current prototype and produce relevant information on the current problem-solving model. In the next sections, we present how we have greatly benefit from these properties to develop the DSTM verification and validation tools.

3 DSTM verification tools

Verification as intended here aims at making a structural evaluation of the KBS. This kind of verification is based on a set of structural properties [12]. In our approach, these properties can be explicitly mentioned by

[5] With the Zola language, one can construct an operation O_i which takes as input an operation and retrieves its functionality by analysing its structure (i.e., its body).

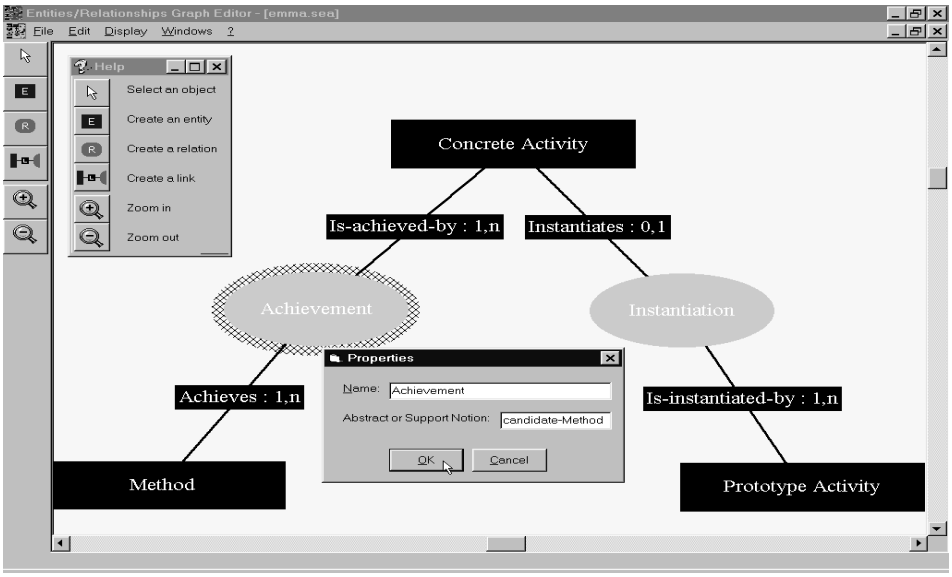
the modeling-team or implicitly retrieved by the system according to how the slots of the modeling primitives are manipulated by the selection mechanisms.

3.1 Explicit constraints on the model

First, as we have mentioned in section 2.2, the slots of the modeling primitives can be syntactically constrained. For instance, in Emma, the *Post-Conditions* of an Activity are supposed to be *a set of strategic facts*[6]. Analysing the knowledge-base and checking such constraints can help in discovering some structural errors.

Second, some integrity constraints on the modeling components can be defined by the use of (binary) relationships. A relationship allows the representation of (1) what role plays the notions of the model involved in the relationship and (2) the maximal and minimal cardinalities. Figure 3 presents examples of relationships defined on the Emma model. As an example, the *Instantiation* relationship can be used to state (1) that a prototype activity must be instantiable by at least a concrete activity (minimal cardinality on the *Is-instantiated-by* role) and (2) that a concrete activity can instantiate at most one prototype activity (maximal cardinality on the *Instantiates* role). This allows putting into evidence violations of quantitative constraints such as "*the prototype activity A_i cannot be instantiated by any concrete activity*" or, using the *Achievement* relationship (cf. figure 3), "*the method M_j is not defined as achieving any task*".

The relationships, defined by the use of the graphical Entity-Relationship language, are automatically translated into Zola representation structures. The roles of a relationship are represented as Zola operations which are automatically constructed according to the abstract notion (or the support notion) underlying the relationship. To do that, we use the reflective capabilities provided by the Zola language (i.e., the capability of constructing operations which analyse the structure of other operations and construct new ones structurally equivalent). For instance, the *Is-achieved-by* role is represented by a new operation automatically derived from the operation used to implement the abstract notion *candidate Method*, which is the abstract notion mentioned by the modeling-team when defining the *Achievement* relationship (cf. figure 3). Such a reflective analysis allows checking if the knowledge-base respects the quantitative constraints (i.e., the minimal and maximal cardinalities applied on the roles of the relationships) without further programming work.



A rectangular box represents an entity. A rounded box represents a relationship. The roles (e.g., *Instantiates* or *Is-instantiated-by*) denote qualitative constraints that must respect elements of the corresponding set to be members of the relationship. Cardinalities denote quantitative constraints on the roles. A relationship is associated with an abstract notion (or a support notion). This notion denotes the qualitative constraints underlying the roles of the relationship.

Figure 3: Two integrity constraints on the Emma model

[6]For modeling the domain knowledge of Emma, we use different types of facts and in particular *domain facts* that describe the state of the solution such as "it is a problem with two variables" or "the objective function is linear" and *strategic facts* that describe aspects of the reasoning process such as "the problem type is defined" or "all the constraints are defined".

3.2 Implicit constraints on the model: influence of missing knowledge

The different tasks and methods of the knowledge-base are manipulated by operational selection mechanisms (cf. figure 1). This defines implicit constraints on how the slots of the tasks and methods must be filled in. For certain mechanisms, the slots that are manipulated must be filled in, an empty slot conducting to an execution error. For some other manipulations, an empty slot can conduct to an erratic behaviour of the system; for instance, an empty *Resources* slot can conduct a method to be never considered as applicable.

In the context of a framework whose mechanisms are fixed, what slots are manipulated and how is known. Therefore, the constraints that the domain knowledge must respect in order to be manipulable by these mechanisms can be explicited and a static syntaxic parser can be hard-encoded in the system. In our prototyping context, the mechanisms are not considered as fixed. They can be modified whilst not modifying the knowledge-base, or *vice-versa*. Therefore, the link between the mechanisms and the effective tasks and methods cannot be *explicited*, it must be *calculated* from the current prototype.

In order to calculate this link, we use Zola reflective capacities. As said before, Zola permits the construction of operations that can analyse other Zola operations. This capability can be used to retrieve what operations manipulate a given slot and how they manipulate it. For instance, analysing the source-code of a high-level action will lead to analyse the source-code of the operation that implements the underlying abstract notion (and of the support notions if any), and then to analyse what slots of the modeling primitives are manipulated. This allows the system to highlight that (for instance) a particular method has no domain knowledge associated to its *Input-Context* and that this can be problematic because, *given the current selection mechanisms*, (1) the high-level action that performs the selection of methods uses the abstract notion *applicable Method* as a selection criterion and (2) a method M_i is an *applicable Method* if the domain knowledge associated to its *Input-Context* is verified.

4 DSTM validation tools

Validation as intended here aims at checking if the current problem-solving model fits the needs, i.e., allows emulating the expected behaviour (a functional evaluation of the KBS [12]).

4.1 Needs for a synthetic presentation of tasks and methods interactions

An intrinsic difficulty of a Task/Method model is that the behaviour of the system is defined by the interactions of the different tasks and methods. When the knowledge-base contains numerous tasks and methods, one of the most important difficulties is to keep in mind a synthetic understanding of the influence of the characteristics of all the tasks and methods on these interactions. Defining a new task (resp. method), modifying the characteristics of some already existing tasks or modifying one of the abstract notions used in the selection mechanisms can influence the overall system's behaviour.

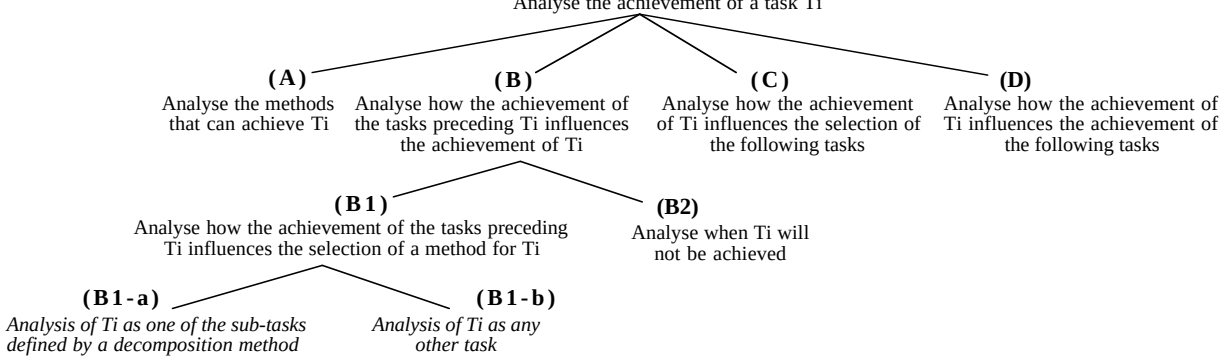
Therefore, when analysing the knowledge-base, questions such as "what is the influence of how task T_i is achieved on the rest of the solving ?" appear recurrently. The pertinent answer to this question is the one that fits the underlying intention, i.e., analysing if the current problem-solving model allows emulating the expected behaviour. The considered task must be analysed from the point of view of its interactions with other tasks, and therefore with the methods that can be used to achieve it and its following tasks. This requires confronting the knowledge-base (the effective tasks and methods) with the selection mechanisms. We use the expression "explanations dedicated to the modeling-team" to denote that what we want is to explain to the modeling-team some consequences of the current couple knowledge-base/custom-built KRL. These explanations can highlight problems in the knowledge-base (e.g., a mistake in the filling of a slot) or problems in the KRL (e.g., the definition of what is an *applicable Method* is not satisfactory).

For constructing these explanations, we have developed automatic tools (that we call explanation modules) that propose synthetic points of view on the current problem-solving model (i.e., the tandem KRL/knowledge-base). Each explanation module is related to an explanation objective.

4.2 An explanation objective: *Analyse the achievement of a task T_i*

Figure 4 presents the explanation objective "*Analyse the achievement of a task T_i*" and how it can be decomposed. Such an explanation objective is a general framework that must be adapted to take the eventual specificities of the adopted modeling into consideration. This adaptation is a two-step process.

First, what should be presented must be adapted to the general Task/Method modeling choices (e.g., "*a task can be associated with multiple methods or with only one method*" or "*methods associated with a same task*



Every node from the first level (A, B, C, D) denotes an explanation objective that can contribute to a synthetic view on the achievement of a task Ti. Nodes from this level can be exploded into sub-objectives. As an example, B1 and B2 denote two different explanation objectives that are connected with B. B1 analyses the influence of preceding tasks on how Ti will be achieved; B2 analyses when Ti will not be achieved, i.e., it is no longer necessary to achieve it (its objectives have already been produced) or it cannot be achieved (the context in which its required resources cannot be produced or none of its possible methods can be used). On the contrary, B1-a and B1-b are not two different objectives, but two different ways to attain B1 objective. If a method defines that a task Ti should be achieved by successively achieving sub-tasks Ti1, Ti2 and Ti3, there is a specific link between these Tij tasks; therefore, when analysing (for example) Ti2, one must highlight its interactions with Ti on one hand, with Ti1 and Ti3 on another hand. Note that what *preceding* and *following* tasks are is related to the adopted modeling and that this explanation objective can only be used on tasks whose order can be accessed, either because it is explicit (e.g., in the case of hierarchical decompositions) or because it can be calculated (e.g., by analysing pre and post conditions).

Figure 4: The explanation objective "*Analyse the achievement of a task*"

can have different influences on the rest of solving or not"). Some sub-objectives only exist in some particular contexts. For instance, (A) is only to be examined if a task can be achieved by multiple methods. Some other modeling choices require a reformulation of some of the sub-objectives. As an example, when a task can be achieved by multiple methods, two different cases occur: (i) what method is used to solve a task has no influence on the rest of the solving or (ii) methods have side-effects[7]. How to attain the B1 objective is very different in (i) and (ii) contexts. In context (ii), the aim is to put the interactions between different methods used to achieve different tasks into evidence, such as for example the fact that achieving task T_i by method M_x can influence the selection of the method that will be used to achieve task T_j. For this purpose, what results will be obtained according to the possible methods for T_i must be defined and compared to the selection criteria of the possible methods for T_j. In context (i), the only point is *if* T_i is achieved and not *how* it is achieved.

Second, how to attain an explanation objective must be defined in the context of the particular specificities of the current problem-solving model. Figure 5 presents how (B1) explanation objective is achieved for the Emma model. As an example of adaptation, one can notice that, as in Emma methods do not have side-effects, the definition of the results of the preceding tasks (B112 in figure 5) does not take the existence of different possible methods into account. One can also note the influence of the use of different types of knowledge in the domain model and the different possible status (necessary or possible) of the interpretation knowledge attached to the tasks (cf. a1, a2, b1, b2 nodes in figure 5). Finally, the analysis of "*how the results of the preceding tasks influence the selection of a method*" (B12 node) is done according to the slots *resources*, *selection context* and *favourable context* which are used as selection criteria in the Emma model (cf. section 2.2).

4.3 Reusing the DSTM framework for implementing the explanation modules

In order to be able to dynamically (and automatically) adapt the explanation to be produced according (1) to the general Task/Method modeling choices and (2) to the relevant specificities of the considered problem-solving model, we have implemented the explanation modules by reusing the DSTM framework. Thus, an explanation module corresponds to a dynamic selection of explanatory tasks and explanatory methods according to a given explanation objective and to the adopted modeling choices.

Adaptation to the Task/Method modeling choices

Each node from the general decomposition of the explanation objectives (cf. figure 4) is represented by an explanatory task and the different ways to reach an explanation objective are represented by explanatory methods. Therefore, the final explanation is constructed by dynamically selecting explanatory tasks and explanatory methods. An *explanatory task* is defined by its underlying explanation objective, the context

[7] they all can be used to achieve the task but some of them can produce additional knowledge that can influence the rest of the solving (i.e., the selection or achievement of tasks and/or methods).

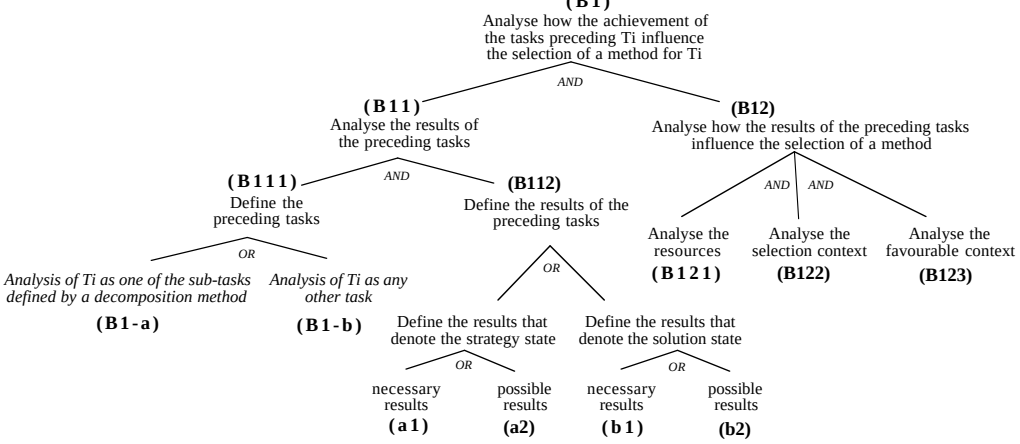


Figure 5: Reformulating the explanation objective for the Emma model (partial)

in which it is relevant to be studied[8] and a set of explanatory methods. Two types of *explanatory methods* are distinguished: *operational method* and *decomposition method*. A *decomposition method* specifies how a task is divided up into sub-tasks and an *operational method* specifies how a task without further sub-tasks can be achieved. The operational methods produce texts. The overall explanation is obtained by compiling these texts. To each method is associated a modeling context underlying the different Task/Method modeling choices. Thus, an explanatory method describes a way to reach an explanation objective in a particular modeling context. Figure 6 presents the explanatory task "Analyse the candidate Methods of a Task" and its associated explanatory methods (this explanatory task corresponds to the (A) node of the figure 4).

Task Analyse-candidate-methods
Objective "Analyse the candidate methods of a task"
Selection-context (Add-New-Task(Ti) ∨ Modification-Abstract-Notion(candidate-Method) ∨ ...)
Associated-methods (Present-the-candidate-method, Analyse-candidate-methods-without-side-effects, Analyse-candidate-methods-with-side-effects)

Method Present-the-candidate-method
Modeling-context (One-Task-One-Method)
Body *Zola-operation*

Method Analyse-candidate-methods-without-side-effects
Modeling-context (One-Task-Muliple-Methods ∧ Methods-Without-Side-Effects)
Sub-tasks (Research-Candidate-Methods, Compare-Candidate-Methods/Selection-Criteria)

Method Analyse-candidate-methods-with-side-effects
Modeling-context (One-Task-Muliple-Methods ∧ Methods-With-Side-Effects)
Sub-tasks (Research-Candidate-Methods, Compare-Candidate-Methods/Selection-Criteria, Compare-Candidate-Methods/Produced-Results)

The explanatory task `Analyse-candidate-methods` aims at comparing the differents methods of a given domain task. This task is relevant when a new task has been defined or when the definition adopted for the *candidate Method* abstract notion has been modified (`Selection-context` slot). Three explanatory methods can be used for achieving it. The first one (`Present-the-candidate-method`) is an operational method which is applicable in the context of a Task/Method modeling where one task is associated with only one method (`Modeling-context` slot). The second one (`Analyse-candidate-methods-without-side-effects`) is a decomposition method which is applicable when one task can be associated with multiple methods which have no side-effects on the rest of the solving. It decomposes the task `Analyse-candidate-methods` in two sub-tasks: `Research-Candidate-Methods` and `Compare-Candidate-Methods/Selection-Criteria`. The first sub-task aims at retrieving the different candidate methods and the second one aims at comparing these candidate methods according to their selection criteria. The third method (`Analyse-candidate-methods-with-side-effects`) is also a decomposition method which is applicable when one task can be associated with multiple methods which can have side-effects on the rest of the solving. In this context, the candidate methods have also to be compared according to the results they can produce. This is why the explanatory task `Compare-Candidate-Methods/Produced-Results` belongs to the list of the sub-tasks decomposition of the method `Analyse-candidate-methods-with-side-effects`.

Figure 6: An explanatory task and its explanatory methods

[8]This context is related to the different modeling actions that have been performed by the modeling-team within the DSTM framework. A modeling action can be concerned with the customisation of the DSTM operational kernel (i.e., the prototyping of the KRL) or the refinement of the current knowledge-base (i.e., the prototyping of the PSM). Examples of such actions are "*modification of the characteristics of the task modeling primitive*", "*definition of a new abstract notion*" or "*definition of a new method in the knowledge-base*".

Adaptation to the relevant specificities of the problem-solving model

As one can see from figure 6, an operational method is associated with a Zola operation (`Body` slot). As said before, these operations produce parts of the final explanation (parts which are linked together thanks to the selection process provided by the DSTM framework).

Most of these Zola operations reuse those that represent the abstract notions. For instance, in order to compare the different methods that can be used to achieve a task, these methods must first be retrieved. This is achieved by the Zola operation associated with the operational explanatory method `Search-candidate-Methods`. This operation directly reuse (as a selection criterion) the Zola operation that implements the *candidate Method* abstract notion. This is a direct advantage of the explicit representation of the selection mechanisms.

Some parts of the explanation require an analysis of the current selection mechanisms. For instance, in order to emphasise when a method will be preferred to another (the explanatory task `Compare-Candidate-Methods/Selection-Criteria` presented in figure 6), the different slots used as selection criteria must first be retrieve. To retrieve these slots, the explanatory task `Compare-Candidate-Methods/Selection-Criteria` is associated with an operational method whose Zola operation performs a structural analysis of the operation that implements the *applicable Method* abstract notion. One can notice that the same reflective analysis is required when analysing how the results of the tasks preceding a task T_i influence the selection of a method for T_i (cf. B121, B122, B123 nodes in figure 5).

However, few parts of the explanation require information that cannot be calculated from a reflective analysis of the current problem-solving model. For such cases, we have defined specific abstract notions dedicated to our explanation purpose. These abstract notions have to be defined by the modeling-team when customising the DSTM kernel. For instance, the explanatory task `Define-results-task` (cf. B112 node in figure 5) is associated with an operational method whose Zola operation uses a specific abstract notion called *task results*.

Figure 7 presents an episode of what is produced by the module "Analyse how the achievement of tasks preceding T_i influences the selection of a method for T_i" in the context of the Emma project. Additional information on our approach of explanations dedicated to the modeling-team can be found in [18].

Resources analysis
The method Graphic-method can only be used if the task Constraint-definition has been achieved previously

Selection-context analysis
The method Define-solution-by-equations can only be used if (1) the task Isoquant-traces has been achieved previously and (equations.defined) is produced during its achievement and (2) the task Formalisation-of-the-mathematical-optimisation-situation has been achieved previously or the task Variable-definition has been achieved previously and (nb-variables>2.true) is produced during its achievement

Favourable-context analysis
None of the possible Methods have a favourable context defined

The task that is analysed is *Solution-determination*. *Graphic-method* and *Define-solution-by-equations* are two of the methods that can achieve it. What is emphasised here is how the achievement of the tasks preceding *Solution-determination* influences the selection of one of these methods.

Figure 7: Information produced by an explanation module (excerpt)

5 Discussion

5.1 Scope of our approach of Verification and Validation

Structural verification is currently the most advanced aspect of KBS evaluation, and it is mainly focused on production rules [8]. In this paper, we advocate that the problematic of verification and validation of a Task/Method KBS has to be studied because of the increasing development of such systems. The approach we propose is based on a reflective analysis of the problem-solving model explicitly represented in the prototype. This allows constructing tools that automatically adapt themselves to the current problem-solving model. Therefore, our verification and validation tools can be considered as metatools [5].

Information provided by DSTM verification tools is mainly dedicated to the knowledge-engineers. Anomalies detected by theses tools generally emphasise a problem in the *structure* of the Task/Method knowledge-base. Information provided by DSTM validation tools can highlight a misunderstanding or a distortion of the knowledge supplied by the experts and/or a lack of precision in the knowledge expressed by the experts. In other words, they facilitate the evaluation and the refinement of the *content* of the Task/Method knowledge-base (evaluation that is made in collaboration with the domain-experts which play an introspective role in order to validate the PSM underlying the knowledge-base). For instance, in the context of the Emma project, the results produced by our explanation modules have pointed out several deficiencies in the problem-solving behaviour

initially described (in an informal way) by the teachers, and thus have helped the teachers to progressively refine the PSM they would like to explicit and transmit.

5.2 Comparison with knowledge-acquisition tools

Knowledge-acquisition tools have been studied in different works such as Omos [11], Salt [13], Expect [7] or MetaKit [6].

Omos, Salt or Expect essentially aim at supporting the instantiation of the problem-solving model with expert-knowledge. In works such as Omos or Salt, the system can analyse the knowledge-base according to how knowledge is used in the strategy, the "role" of knowledge in a PSM [15]. Such tools can provide help such as "*there is a lack of knowledge concerning this aspect of the PSM*". Expect goes a step further in this way by allowing an automatic derivation of such knowledge-acquisition tools by a reflective analysis of its knowledge-base. Expect can consider any strategy defined in the representation formalism it proposes, when Omos or Salt only accept modifications of the knowledge-base. MetaKit essentially aims at verification issues. A dissociation is introduced between macro verification and micro verification. Macro verification is used to check the overall structure of a project, for example that a leaf task of the decomposition tree is not performed by a decomposition method. Micro verification is used to check details, for example that each method is typed.

Our verification tools essentially address the phase where the knowledge-base is constructed in a process guided by the adopted KRL. They aim at discovering errors in the knowledge-base. The tools that check explicit constraints can be compared to MetaKit verification tools. The advantage of our approach is that the constraints to be checked are explicitly defined (using the graphical Entity-Relationship language) by the modeling-team, and not predefined and hard-encoded. The tools that analyse implicit constraints are based on an analysis of the source-code, in a similar way to Expect. In a system such as Salt, the PSM and what knowledge is supposed to be acquired is known. Knowledge-acquisition tools can therefore be constructed according to these specifications. In DSTM, how knowledge is manipulated can be modified and therefore the knowledge-acquisition tools (related to the role that plays knowledge in the PSM) must be based on a reflective analysis of the source-code.

Our explanations address both phases of the elaboration of the problem-solving model. They do not aim at stating "*there is a problem in the knowledge-base*", but "*here is some information on how the current KRL and the current knowledge-base interact*". In other terms, they are not automatic debugging tools, but tools that provide the modeling-team with information that is susceptible to highlight a problem with the current KRL (the definition adopted for the modeling primitives and/or the selection mechanisms are not satisfactory) or with the current knowledge-base (errors when filling the slots of one or several tasks or methods).

5.3 Current direction of the work

First, as presented in section 3, our verification tools are essentially based on constraints defined by the modeling-team in an explicit way (by the use of the Entity/Relationship representation language) or an implicit way (by the definition of the selection mechanisms which necessary imply some constraints for the tasks and the methods of the knowledge-base). As pointed out by several works, one can state some predefined constraints according to the representation formalism used for implementing the KBS [12]. For instance, in the context of a rule-based representation formalism, one can define (a priori) a set of structural properties that would be part of the structural verification such as *redundant rules*, *conflicting rules* or *circular rule chains*. We are currently studying such structural properties in the context of the Task/Method modeling paradigm. A first study seems to indicate that the constraints generally used in a rule-based formalism can be reused and reformulated in a Task/Method formalism. For instance, it would be interesting to check if the knowledge-base does not contain *conflicting methods* (two methods are in conflict when they have been defined as achieving a same task T_i but produce conflicting results for T_i) or *circular tasks* (a task T_i is a circular task when it belongs to the sub-tasks decomposition performed by one of its associated methods).

Second, as said before, the DSTM framework can recommend the modeling-team to use a specific validation tool. This advice is based on the different modeling actions that have been done previously. However, when constructing the explanations, our system does not take the background of the DSTM session into account. Therefore, if the same explanation module is launched twice, the same final explanation will be produced. We are currently studying (and modeling via the explanatory tasks and methods) the adaptation of the explanations to how the modeling-team is secured with the knowledge it is managing. Such a model of the modeling-team (which would be constructed dynamically during a DSTM session) could be used to select new explanatory methods which, for instance, would allow shortcuts in the final explanation when the modeling-team would be supposed to master a particular modeling notion (e.g., the *candidate Method* abstract notion).

References

[1] R. Benjamins. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications*, 8(2):93–120, 1995.

[2] B. Chandrasekaran and T.R. Johnson. Generic tasks and task structures: History, critique and new directions. In J.M. David, J.P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*, pages 232–272, Berlin, Germany, 1993. Springer-Verlag.

[3] C. Choquet, P. Tchounikine, and F. Trichet. La modélisation de la méthode de résolution de problèmes dans le système Emma. In *Actes des Journées francophones EIAO de Cachan, (In french)*, pages 263–275. Hermès, 1997.

[4] J.M. David, J.P. Krivine, and R. Simmons. *Second Generation Expert Systems*. Springer-Verlag, 1993.

[5] H. Eriksson and M. Musen. Metatools for knowledge acquisition. *IEEE Software*, 10(3):23–29, 1993.

[6] S. Geldof, A. Slodzian, and W. Van de Velde. From verification to life cycle support. *IEEE Expert*, 11(2):67–73, 1996.

[7] Y. Gil and C. Paris. Towards method-independent knowledge acquisition. *Knowledge Acquisition*, 6(2), 1996.

[8] U. Gupta. *Validating and Verifying Knowledge-Based Systems*. IEEE Computer Society Press, 1991.

[9] I. Istenes and P. Tchounikine. Zola: a Language to Operationalise Conceptual Models of Reasoning. *International Journal of Computing and Information*, 2(1):689–706, 1997.

[10] N. Juristo. A common framework for conventional and knowledge based software validation and verification. In *9th International Conference on Software Engineering and Knowledge Engineering (SEKE'97)*, pages 287–294, Madrid, Spain, 1997.

[11] M. Linster. Integrating conceptual and operational modelling: a case study. *Knowledge Acquisition*, 5:143–171, 1993.

[12] B. Lopez, P. Meseguer, and E. Plaza. Knowledge based systems validation: A state of the art. *Artificial Intelligence*, 3(2):58–72, 1990.

[13] S. Marcus and J. McDermott. Salt: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–37, 1989.

[14] C. Pierret-Golbreich. *TASK : un environnement pour le développement de systèmes à base de connaissances flexibles*. Habilitation à diriger des recherches, Rapport de Recherche LRI-1056, (In french), 1996.

[15] C. Reynaud, N. Aussenac-Gilles, P. Tchounikine, and F. Trichet. The notion of role in conceptual modeling. In *10th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW'97)*, number 1319 in Lectures Notes in Artificial Intelligence, pages 221–236. Springer-Verlag, 1997.

[16] L. Steels. Components of Expertise. *Artificial Intelligence*, 11(2):29–49, 1995.

[17] F. Trichet and P. Tchounikine. Reusing a Flexible Task-Method Framework to Prototype a Knowledge-Based System. In *9th International Conference on Software Engineering and Knowledge Engineering (SEKE'97)*, pages 192–199, Madrid, Spain, 1997.

[18] F. Trichet and P. Tchounikine. Structured explanations as a support to model problem-solving in a Task-Method paradigm. In G. Grahne, editor, *Sixth Scandinavian Conference on Artificial Intelligence (SCAI'97)*, number 40 in Frontiers in Artificial Intelligence and Applications, pages 131–142, Amsterdam, Holland, 1997. IOS Press.