Zola: a Language to Operationalise Conceptual Models of Reasoning

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

Abstract

This paper presents Zola, a language that allows the operationalisation of conceptual models of expertise. Zola is more specifically designed to support the definition of conceptual models by knowledge level prototyping. It does not include constructions specific to a particular methodology (as tasks or roles), but provides general primitives that allow the construction of the conceptual primitives specific to the conceptual model elaborated from the expertise. Zola provides three main levels of representation: domain knowledge, operations (that can manipulate the domain knowledge), and strategies (algorithms). Zola is a reflective language: operations can analyse other operations. This allows the construction of reflective operations that can give the modelling team some feed-back on the implemented model.

After a short presentation of the objectives of the project underlying Zola, we describe how Zola has been used to operationalise a conceptual model of reasoning for a fault diagnosis system. We then summarise the advantages of Zola to elaborate a conceptual model of a KBS.

Keywords

Knowledge Engineering, Conceptual Model of Reasoning, Operationalisation language, Knowledge Level Prototyping.

Introduction

Zola is a language that allows the operationalisation of conceptual models of expertise. Zola was defined to support the Mapcar project, which aims to provide help to a modelling team defining a conceptual model of expertise by knowledge level prototyping. Therefore, although it can be used to operationalise conceptual models defined in another manner, Zola has a certain number of specificities bound to the Mapcar project objectives. Section 1 presents our approach to knowledge level prototyping and why it requires a specific language, what specifications have been defined for Zola, and a general overview of the language. Section 2 presents how Zola allows the operationalisation of a conceptual model that has been defined for a fault diagnosis expert system for robots in plasturgy. Section 3 discusses the advantages of Zola as a language to elaborate conceptual models. The point of view adopted in this paper is to emphasise the language structures and the features that make it appropriate for knowledge level prototyping ; a more formal presentation can be found in [5].

1 Objectives, specifications and general overview of Zola

1.1 Objectives

The conceptual model of a Knowledge Based System is the description at an abstract level of the problem solving expertise: high-level actions that will be used during solving, different types of knowledge that have to be used, general strategy. It describes the problem solving method independently from its implementation (what is usually called a description "at the knowledge level" [12]), and allows the study of the expert system behaviour at a level that is comprehensible to human experts. The following list of the possible uses of a conceptual model has been elaborated in the EuroKnowledge project [1]: "specify the requirements of one or more KBSs", "drive model-based knowledge acquisition", "assist evaluation of the correctness and/or completeness of knowledge to improve communication", "document knowledge in an unambiguous manner, e.g. corporate knowledge assets".

Different approaches to the definition of the conceptual model of a KBS have been proposed as part of Knowledge Acquisition methodologies. They can roughly be classified as bottom-up approaches (essentially based on data abstraction, as in Macao [2]) or top-down approaches (essentially based on model refinement, as in Kads [3]). In practice, data abstraction is always necessary. According to the methodologies, there is more or less guidance for this abstraction and, more importantly, the aim can be to obtain a complete conceptual model, a framework of the model, or only to acquire enough relevant knowledge to select and refine generic models.

The approach we have adopted in the Mapcar project is the elaboration of conceptual models by data abstraction[1]. More precisely, we want to construct a framework that allows knowledge level prototyping, i.e., the definition of a model by iteratively testing and refining a first version of the model until it gives satisfaction to the modelling team[2]. Such an approach allows modelling in a short time scale the effective activity of human experts. Defining a running prototype allows the experts to judge and refine the model under construction. Moreover, constructing the prototype gives a concrete idea of what can be expected from the expert system. Although developing a complete KBS by prototyping has well known disadvantages, knowledge level prototyping is a valuable approach for the elaboration of the conceptual model [7,8,10].

To avoid low-level implementation details, knowledge level prototyping must be supported by an appropriate operationalisation language. Such a language must allow easy representation of the model concepts. A certain number of languages, such as Model-K [7] or OMOS [8], have been defined to operationalise the conceptual models obtained following Kads methodology. These languages provide ready-to-use primitives that correspond to that of the methodology. In a bottom-up approach such as the one adopted in the Mapcar project, providing such modelling primitives is not possible, as the structure of the conceptual model to operationalise is not known. To tackle this problem, we have constructed the Zola language.

1.2 Specification of the language

Zola, as an operationalisation language, shares some specifications common to most of such languages:

- ability to define the inference layer (domain knowledge and its manipulation) and the strategy layer (how to control solving) with a minimum of implementation constraints.
- prototyping facilities (easy definition and easy modification of the model components).
- respect for the structural correspondence principle, i.e., the fact that the conceptual model is preserved in the implementation [9].

[1] Note that this does not mean that the final model will not be selected or derived from a predefined library, but that the first steps of the modelisation will consist in understanding and modelling the domain expert process, this process being continued up to the definition of a complete conceptual model.

[2] We refer here to the different people (experts, knowledge engineers, etc) who can be involved in the process of elaborating the conceptual model.

The most specific features of Zola are:

- Universal structures

  In general, operationalisation languages are dedicated to a particular methodology. As an example, Model-K [7], that has been defined to operationalise Kads models of expertise, provides conceptual primitives corresponding to the methodology ("Tasks", "Knowledge Roles", etc.), thus allowing a direct mapping between the conceptual model and the language primitives.

  As we want to operationalise conceptual models defined by data-abstraction, i.e. models whose components cannot be anticipated, Zola does not provide ready-to-use conceptual primitives. Such primitives would implicitly enforce a particular modelisation. The language proposes basic structures that can be combined to define the conceptual primitives necessary for a particular model. In other words, Zola primitives are not supposed to be used as modelling primitives. Zola primitives allow an easy construction of the conceptual primitives specific to the conceptual model elaborated from the expertise[3]. Prototyping with Zola requires a step more than prototyping with a language such as Model-K. This step is the elaboration of the conceptual primitives that correspond to the expertise being modelled.

- Different uses of the Knowledge Base

  A Knowledge Base represents knowledge. As long as the conceptual model is not definitely fixed, it should be possible to change the way the domain knowledge is used by the inference layer. It should also be possible to manipulate domain knowledge with objectives other than solving the problem, as for example presenting different views of system knowledge during the validation process. For this purpose, Zola enables the dissociation of knowledge from how it is used, and allows the definition of different ways of manipulating the knowledge.

- Domain-specific analysis tools

  In a prototyping approach, the implemented model represents a partial and/or provisory understanding of the domain expertise. As the model evolves, the modelling team can have some difficulty to control what the system can and cannot do, or what will happen if this or that is changed in the model. The experts need assistance in seeing the functionality of these changes, in discovering their possible errors and in understanding them better. Some informal but useful validation work can be done by analysing the implemented model and giving

[3] It can be argued that any structure can be considered as a "conceptual primitive", and that a language cannot be "neutral". We agree that a "neutral" modelling language has no sense. Zola is not a modelling language, its primitives are not supposed to be used to model behaviour directly.

feed-back to the modelling team, and this feed-back can play an important role in the prototyping cycle. Part of this analysis may be domain independent and rely on general mechanisms (checking the completeness of domain concepts, etc.). We think that model-specific analysis tools are also necessary, and that such tools can be defined as specific KBS. Zola allows the definition of reflective KBS tools that can analyse model components, i.e., the different types of knowledge and actions.

The way the elaboration of a conceptual model is considered in the Mapcar project is summarised in Figure 1. A first "paper" version of the model is elaborated and operationalised using Zola. It can be tested on case studies and analysed using model-specific or model-independent tools. Given this feedback, the modelling team can modify the model. As the conceptual model is preserved in the implementation, a modification of an aspect of the conceptual model corresponds to a modification of an identified part of its representation in Zola.

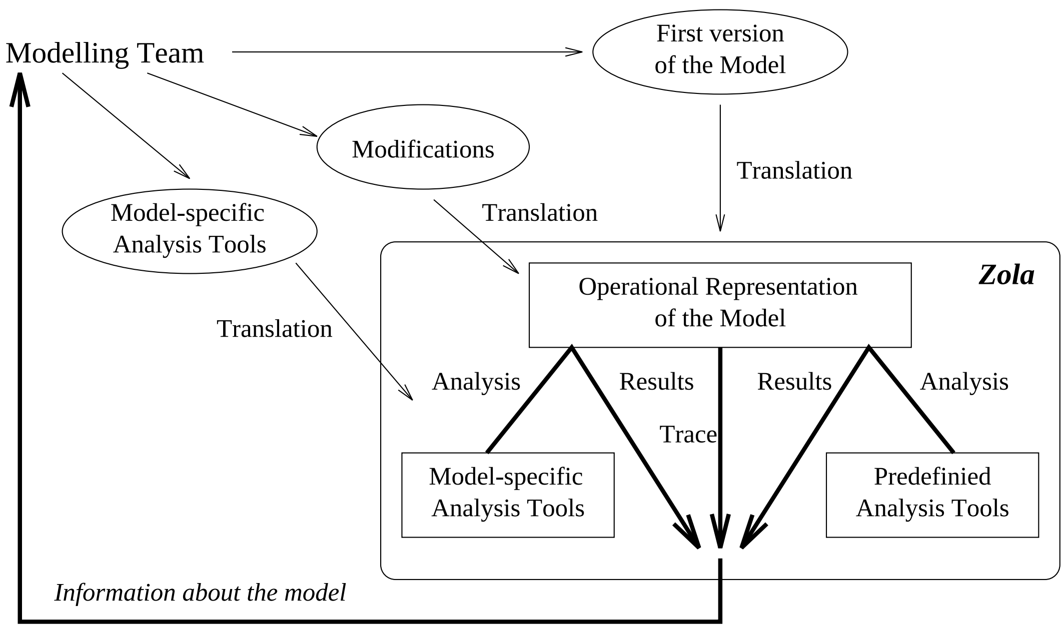


Figure 1: General approach in the Mapcar project

Note that Zola has been constructed to support knowledge level prototyping as presented here, i.e. to allow the construction, representation and testing of a conceptual model by means of a prototype. There is no claim that Zola allows representation in the best possible way of all the domain knowledge (one only needs a small representative amount of domain knowledge for testing purposes, and the fact that its representation in the prototype is not the most pertinent is not critical), nor that Zola should be used to implement the final system.

1.3 General overview of Zola

The Zola language proposes four levels of representation, each of them corresponding to a specific sub-language:

L3 for domain knowledge representation. L3 allows the definition of domain specific types of knowledge. These types are represented as frames that can be instantiated by the domain basic knowledge. Types are structured in a hierarchy.

L2 for operational knowledge representation. L2 allows the definition of operations that manipulate the objects defined with L3. To represent these manipulations, the language proposes a set of predefined primitives that correspond to control structures (While, If-Then-Else, etc.) and L3 manipulation primitives (Match, etc.). Operations are defined by instantiating control structures with operations defined previously or manipulation primitives. Operations can be attached to a knowledge type. If an operation is not defined for a type it can be inherited following the hierarchy of types.

L1 for strategy representation. L1 allows the definition of algorithms over L2 operations. L1 simply corresponds to Lisp. This allows the definition of the overall structure of the high-level system operations without having to use Zola primitives.

L4 for additional conceptual information. L4 allows the representation of additional information, and more particularly conceptual 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 representation of a conceptual model. Conceptual information can be represented as conceptual handles, i.e. information attached to a knowledge type or an operation, or as conceptual description, that is not attached to another particular component. As an example, "what an operation does" or "how it proceeds" are two items of "conceptual information" that cannot easily be retrieved from the description of the operation in L2. This information can be added to the operation with conceptual handles.

Zola is a reflective language that allows both knowledge level reflection and computational reflection.

Knowledge level reflection is the capacity to analyse a dedicated abstract representation of the conceptual model [9] to enhance the flexibility of the system. This is made possible by reasoning on the description of the knowledge types, the operations, and the additional conceptual information represented with L4. Zola knowledge level reflection capacities are not emphasised in this paper ; see [6] for a focus on this aspect of the language.

Computational reflection is the capacity to analyse the code of a system operationalised with Zola. This is made possible by the fact that the L2 and L3 structures are isomorphic. This means that L2 operations can manipulate not only domain knowledge (L3), but also operations (L2). Using this reflective capacity allows the building of model-specific analysis tools as particular strategies relying on particular operations. This allows the construction of analysis tools that can calculate some pertinent information on the knowledge base and the operations that manipulate it. An example of such analysis is presented in Section 2.

1.4 Using Zola to represent and operationalise a conceptual model

Representation. The representation of a conceptual model with Zola corresponds to:

1. Definition of the knowledge types with L3 and, if necessary, of conceptual handles about these types.
2. Definition of the high-level operations (i.e. operations that make sense at a knowledge level) with L2 and, if necessary, of conceptual handles about these operations.
3. Definition of the control structure. This can be achieved using L1 if the control structure is a simple algorithm on high-level actions, or can be modelled as a specific KBS.
4. Definition of additional conceptual information on the model on its own (i.e. not related to specific components).

Operationalisation. The operationalisation of a conceptual model with Zola corresponds to:

1. Definition of the knowledge types instances with L3.

2. Definition of the technical information required for the operationalisation (including knowledge types that are not necessary for the representation of the model).
3. Definition with L2 of the intermediate and low-level operations that describe how to perform the high-level operations.

Note that during knowledge level prototyping, the elaboration of the model is performed together with its semi-formalisation using the language syntax and its simulation by an operational prototype. The conceptual difference we make between representation and operationalisation does not necessarily correspond to two different phases ordered in time during the prototyping process.

Zola conception benefits from lessons learned from the Mapcar language [11]. This previous language was based on the use of a single formalism (production rules) to represent the domain layer and the inference layer. It demonstrated that the reflectivity allowed by this uniformity could give some interesting results, but also that production rules as a formalism remain too poor to avoid syntactic tricks.

Zola is implemented in Lisp. A (French) interface has been developed using the Ilog Aida/Masai generator.

2 Elaboration of a conceptual model with Zola: Fiona

In the context of the Fiona[4] project we defined and operationalised a conceptual model of fault diagnosis for robots in plasturgy [10]. The aim was to define a high-level conceptual model that allows communication between the different people who have to cope with diagnosis problems: company customers, company trainers and customer technicians. It was decided to elaborate a "cognitive" model, i.e., a model that would allow solving corresponding to that of an efficient human expert.

The fault diagnosis process appeared to be too complex to be implemented using a classical expert system. We could have undertaken the modelling using a generic model, using for example the library proposed by the Kads methodology [3]. This approach was rejected for two reasons. First, we wanted to obtain over a short period of time (two months) a first operational prototype. Using a general methodology would not have permitted the attainment of this objective. Second, we wanted to define an "original company model". This means that the model is not necessarily the best solving model, it must take into account human aspects (it must be understandable and reproducible by different types of persons). To reach this objective,

[4]Funded by the COMETT project 6617Cb and the "Pays de la Loire" region.

knowledge level prototyping appeared to be the appropriate approach: abstraction of the conceptual model by analysing the company experts problem solving activity, construction of a prototype to refine the model and provide a concrete understanding of the project to the company staff[5].

The definition of the model required several prototyping loops: elaboration of version i of the conceptual model, operationalisation, test, and eventually specification of the version i+1. The final model is based on a set of high-level actions that are performed successively following the general algorithm presented in Figure 2.

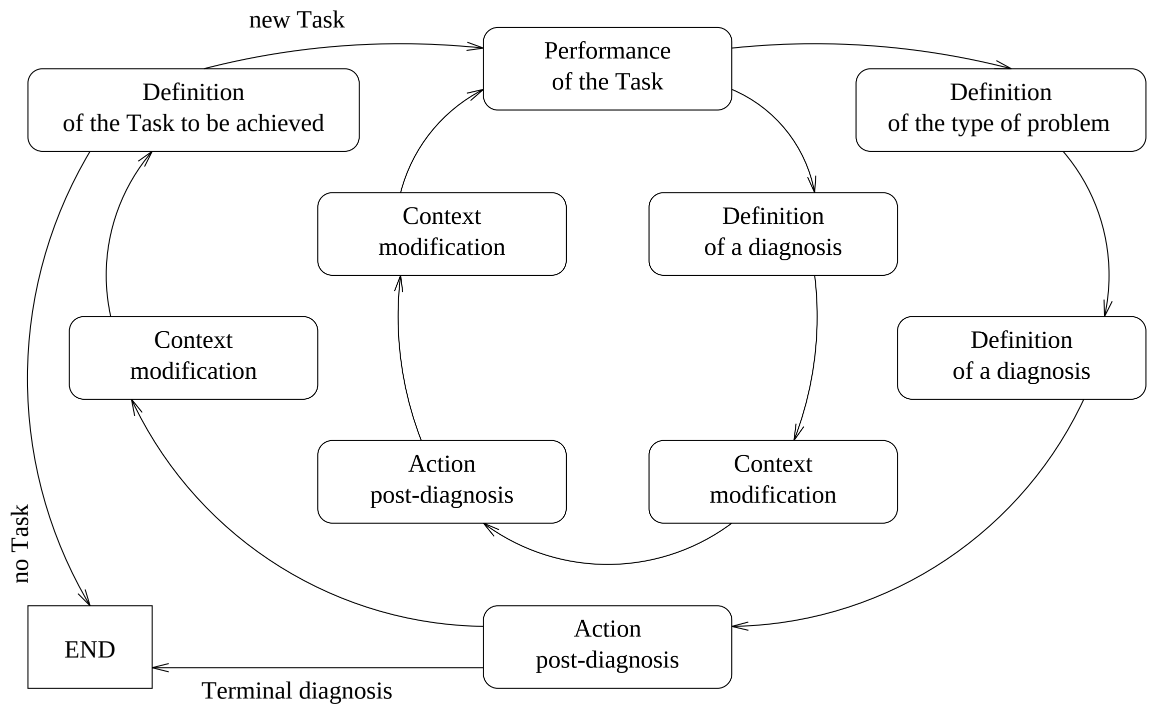


Figure 2: Problem Solving overview

This model is based on the following notions:

Task. A Task corresponds to a concrete action to be achieved, typically a set of manipulations and interpretations (Ex.: "define the gap between where the robot should have stopped and where it actually stopped"). Control knowledge defines what tasks should be achieved next. To every Task corresponds some Task realisation knowledge (procedures and expert knowledge) that represents how to achieve the task.

[5]The model was originally elaborated using the Mapcar language [10,11], re-engineered and improved with Zola.

Problem types. Problem types allow an abstract classification of the possible failures.

Diagnosis states. According to the symptoms, a particular failure can become "possible". It can then become "established", "discarded" or "suspected"; in this case it will either be "confirmed" or "refuted".

Post-diagnosis actions. This defines actions to be carried out after the modification of a diagnosis state such as turning back a switch which was modified to test a suspected fault which was finally discarded, or giving technical information after confirmation of the definitive diagnosis, etc.

Cognitive context. A cognitive context is the expert cognitive framework at a particular state of its diagnosis process[6].

Every high-level action ("Definition of the task to be achieved", "Task performance", "Definition of a diagnosis", etc.) corresponds to the manipulation of specific knowledge (control knowledge, task realisation knowledge, diagnosis knowledge, etc.). The different notions are tightly linked. For instance, in order to define the next task to be achieved (control knowledge) one examines the current type of problem, the current cognitive context, the states of the possible diagnosis, the tasks that have already been achieved and the basic data.

The solving strategy is as follows (Cf. Figure 2). The system first defines what new task should be achieved. During the task performance, it checks whether the cognitive context evolves, and if it does, whether or not it allows a direct diagnosis, and following post-diagnosis actions. When the task is achieved, i.e., all the system knowledge for this task has been used, the system checks subsequently whether it can focus on a particular type of problem, whether it can define a diagnosis, and if so, achieve post-diagnosis actions, whether the cognitive context evolves, and finally what new task should be achieved. If a terminal diagnosis is defined the process ends after the post-diagnosis actions. A typical problem solving case requires 10 to 20 loops (i.e., 10 to 20 tasks to be performed) before making a terminal diagnosis.

A more precise presentation of the model and of how it has been defined using a knowledge level prototyping approach can be found in [10].

2.1 Domain knowledge: L3

In the Fiona model there are a certain number of basic notions: problem type, task, diagnosis, etc. These notions are used to define some active knowledge representing

[6]This notion is not necessary to solve the problem, but is absolutely required to solve it as human experts do, and has been modeled for this reason.

when there is a new problem type, a new task, a new diagnosis state, etc. according to the solving state. A state is denoted by an information set: tasks already achieved, states of the possible diagnoses, current cognitive context, etc.

To represent the different notions of the model ("Problem type", "Context", "Task", etc.) we have created the abstract type "Concept". The different basic notions are defined as subtypes of the "Concept" type (Cf. Figure 3). One of these is the "General-Rule" type, which allows the definition of the different types of active knowledge. The model uses six different types of rules ("Task rules" that represent control, "Context rules", "Diagnosis rules", etc.)[7].

The representation of states (Ex.: list of tasks already achieved[8]) and sets of active knowledge (Ex.: list of "Task rules") requires structuration constructs. This can be achieved by using the predefined "Collection" type. For example the list of tasks already achieved is represented using a collection of tasks. The different Rule Bases are also represented as collections.

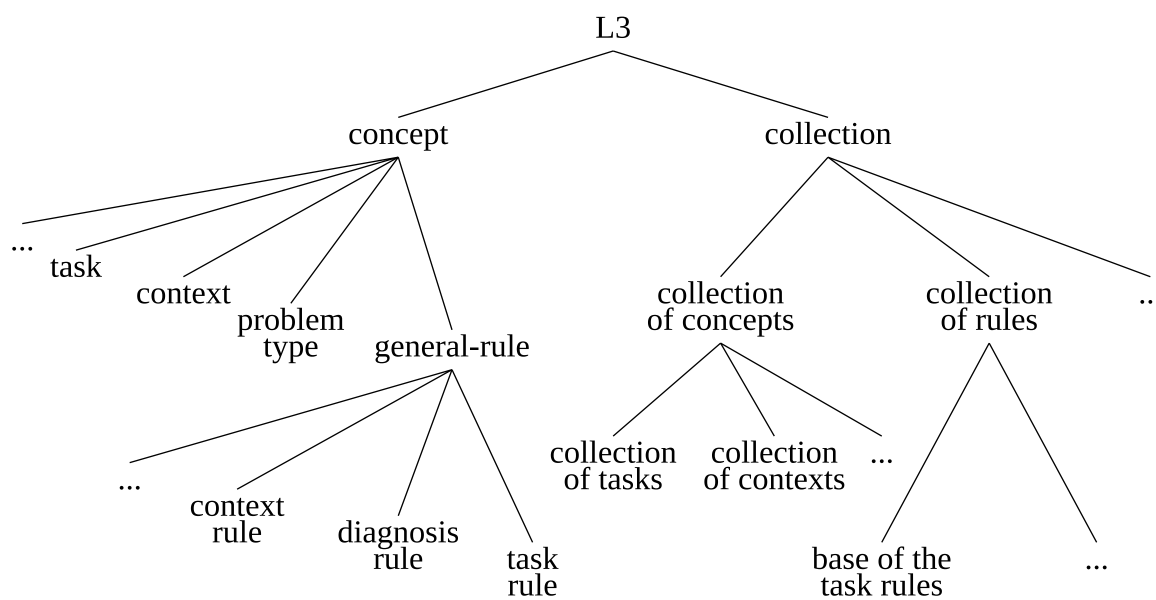


Figure 3: Hierarchy of knowledge types

A knowledge type is defined as a set of fields. Every type inherits the fields of its parent-type. A field can be instantiated by a literal, a list of literals, an instance of a knowledge type or a list of instances of a knowledge type.

[7]The term "rule" is used for historical reasons, in fact they are frames.

[8]This is an example of what we have called "technical information required for the operationalisation" in Section 1.4.

As an example, the type “Task-rule”, that allows the representation of when a new task should be undertaken (field “New-Current-Task”), has a list of specific fields that refer to the current “Problem type”, “Cognitive context”, “Actual diagnosis”, “Situation”, “New task”, and “Task to realise before” ; it inherits the fields “Comment” and “Explanation” from its parent-type “General-rule” (Cf. Figure 4).

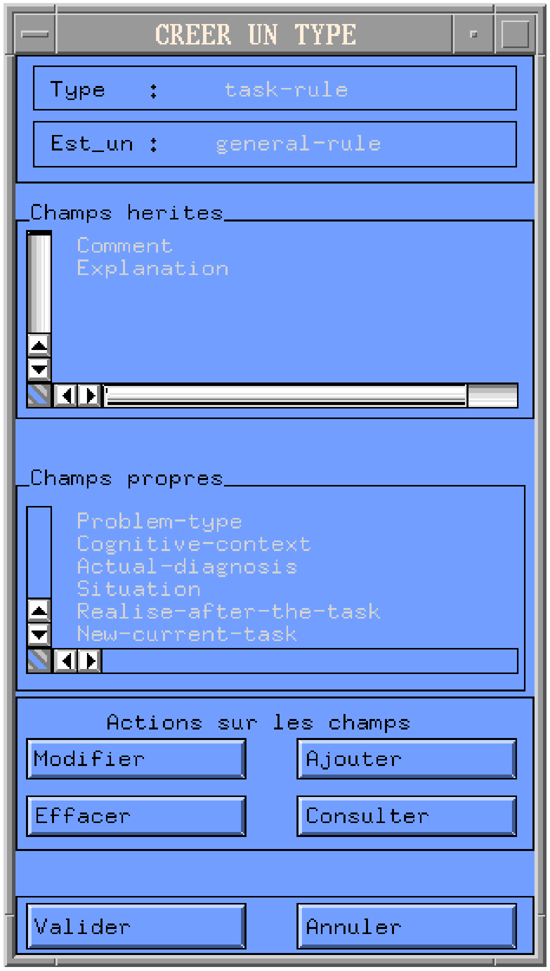


Figure 4: The knowledge type “Task-rule”

When a knowledge type is defined, instances representing concrete objects of the domain can be defined by valuing the fields (Cf. Figure 5).

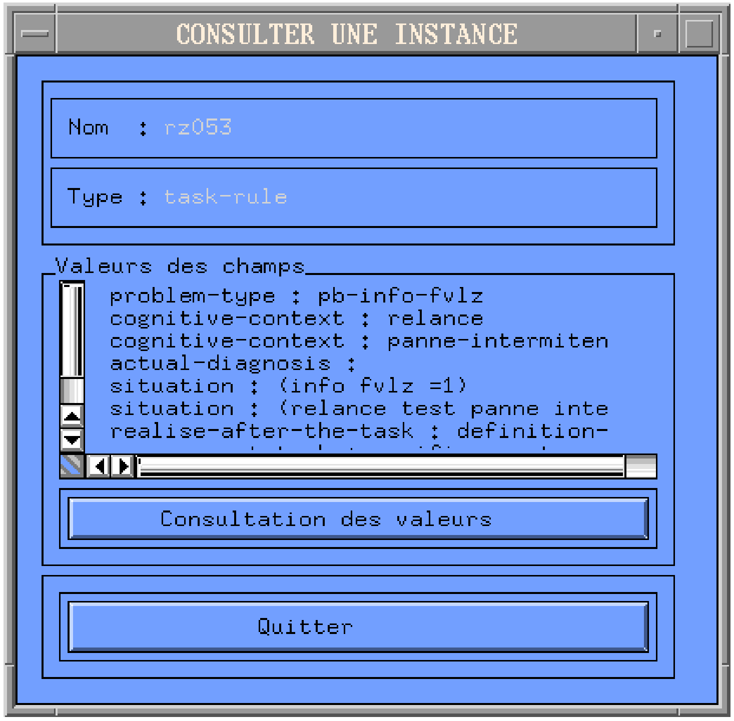


Figure 5: The "rz053" instance of the Task-rule type

2.2 Operational knowledge: L2

Zola proposes a set of predefined operation profiles. These profiles are control structures ("If-Then", "While", "For-all", etc.) and L3 manipulation primitives ("Compare", "add-to-list", etc.). These operation profiles are represented (as in L3) by a set of fields (Cf. Figure 6).

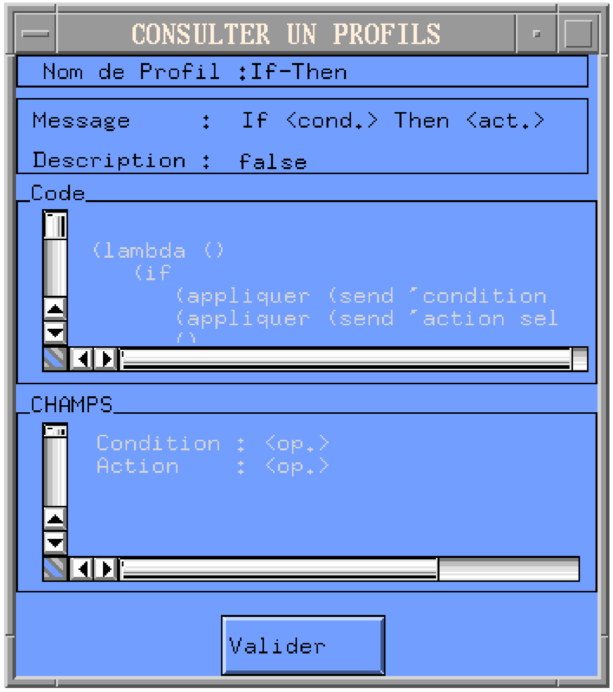
An operation is defined by instantiating a profile with another operation, manipulation primitives or L3 knowledge instances.

Figure 7 presents the "Test-Rule" operation defined for the "Task-rule" type.

As to how they are used to operationalise the model, Zola operations can be classified on different levels.

- High-level operations are those of the model such as "Definition of the current task". In general such an operation uses different types of domain knowledge. Therefore, it is not bound to a particular domain type, but to the default value "L3".

- Intermediate-level operations are defined when decomposing high-level operations for their operationalisation. At some level these operations can be bound



The two fields to value the "If-Then" profile are "condition" and "action".

Figure 6: The "If-Then" profile

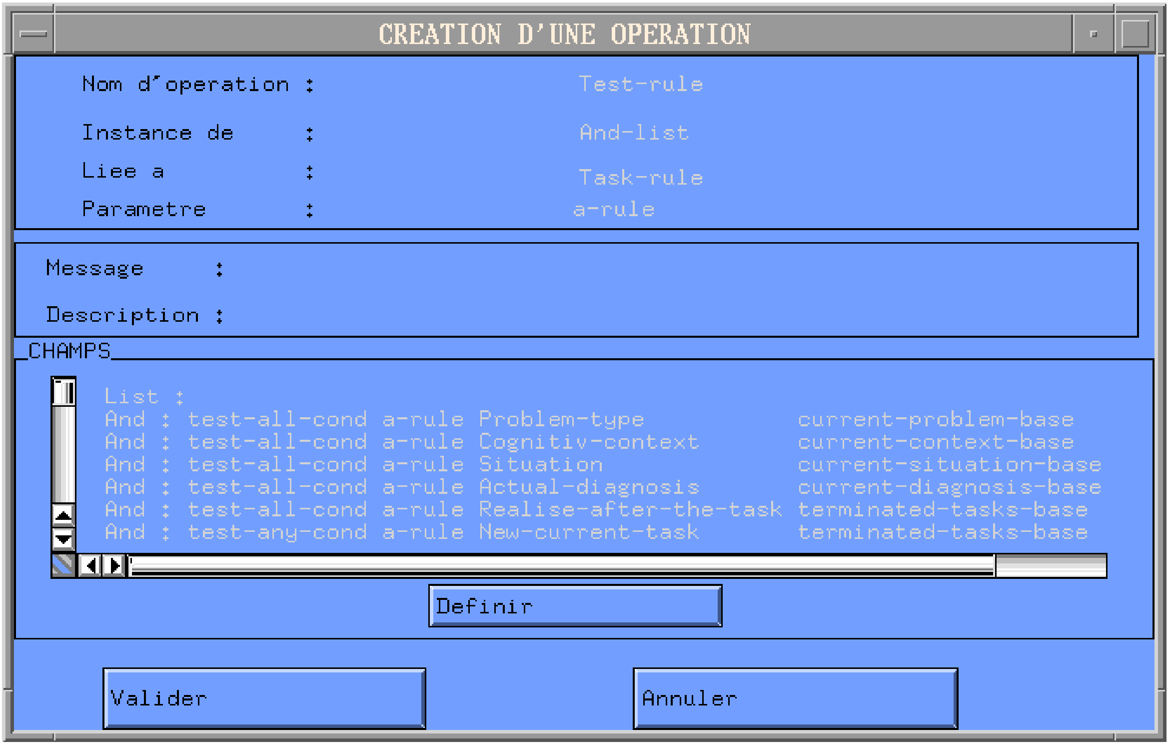
to a specific L3 type (this allows the use of inheritance of operations).

- Low-level operations are operations closely linked to the domain types ; they directly reference L3 fields. As an example, to every type of rule is attached a "Test-rule" and a "Fire-rule" operation: their description directly references the rule fields.

In other terms only high-level operations make sense at the knowledge level. Low and intermediate-level operations are operationalisation features. Intermediate operations can reference a domain type but not its fields. Low-level operations are completely domain dependent.

As mentioned in Section 2, Zola allows computational reflection. One can define with L2 reflective operations that can manipulate other operations. This capacity is not necessary to run the Fiona model, but we have used it to define tools that analyse the model.

As an example, using a reflective operation, we analysed the knowledge and the operations used in the internal loop during Task Performance. Roughly presented, the reflective module analyses the internal loop high-level actions and defines what



The applicability test of the "task-rule" knowledge type is an instance of the profile "and-list". The "and-list" primitive executes the list of operations (the "list" field), and is evaluated to True if all of the operations of the list are evaluated to True. Here the operations are the manipulation primitives "test-all-cond" and "test-any-cond" used on the parameter ("a-rule") and knowledge instances that represent the current state of solving ("current-problem-base", etc.). It tests whether for the given task rule, the problem type is associated (respectively the cognitive context in which this knowledge is defined, the actual situation, state of the possible diagnosis, the tasks to accomplish before) to the collection of actual problem types (respectively the current cognitive context, the state of the actual diagnosis, and the tasks already done) and then test whether the task is finished.

Figure 7: The "Test-rule" operation

domain knowledge can make this loop modify the performance of a Task, and all the Tasks of the knowledge base are then analysed according to this to detect if the loop influences or not the performance of the Task. This allowed the definition of different types of Tasks. Some do not depend on either of the notions that can evolve during the internal loop, some depend on these notions but are always performed with the same values for the corresponding slots, some vary with these values. Interpreting these results by inspecting these different tasks appears relevant at the model level. Tasks from the first set are basic manipulations, that depend on the robot technology. On the contrary, tasks of the second and third set are bound to the point of view adopted in the modelisation, i.e. to the human expert model. And, for the tasks of the second set, the notions are referenced “for information”, just to note that this task only has some sense in this context.This analysis made apparent the fact that some Tasks are never concerned with the manipulations undertaken in the loop, and why they are not concerned. The results led us to propose a taxonomy of the tasks. Although this taxonomy did not appear explicitly in the first modelisation obtained from the knowledge extracted during the interviews, it is significant at the knowledge level.

2.3 Strategies for Fiona

L1 allows, if necessary, the definition of a control algorithm for the execution of L2 operations. A strategy corresponds to a Lisp function that can use L2 operations as simple Lisp functions. In the same way a strategy can reference other strategies. An L2 operation can be used by different strategies.

As Fiona strategy is fixed, it has been represented as an algorithm (Cf. Figure 8).

Figure 8: Extract from Fiona’s general strategy (Cf. Figure 2).

3 Conclusions

Zola is a language that allows the operationalisation of the conceptual model of a KBS. It is not dedicated to a specific theory of how such a model should be structured, except for the (commonly accepted) dissociation between domain knowledge and how it should be manipulated. If a general methodology such as Kads is used, the operationalisation language should be selected among those whose syntactic constructions have been defined according to the methodology concepts. We think that to allow the elaboration of a conceptual model following a bottom-up approach, a language that proposes conceptual primitives, introduces a bias in the modelisation. This is why Zola does not propose conceptual primitives, but constructs that allows the definition of conceptual primitives according to the studied model.

Zola succeeds in representing and operationalising a specific model such as the one elaborated in the Fiona project. Every notion of the model can be explicitly represented in the prototype. The fact that the model representation can easily be modified allows a knowledge level prototypical approach, i.e. to present a first version of the model, run the model to highlight its drawbacks, and specify its evolution. The development of a concrete prototype that reifies the model is of great help for the human experts, who can be confronted with the concrete behaviour that corresponds to the current conceptual model of the system. Using a high-level operationalisation language avoids the major inconveniences of prototyping in knowledge engineering, i.e. compilation of different types of knowledge, low-level analysis constrained by implementation features, mixing conceptual notions and implementation features. The Zola capacity to define reflective tools allowing a rational study of the model also gives some pertinent feed-back to the modelling team, and contributes significantly to the modelisation process.

To conclude, we can say that the Zola language allows the putting into practice of the approach proposed in Figure 1. Using Zola, the conceptual model of a KBS can be elaborated by abstraction from the human experts whilst putting emphasis on modelling, and not on operationalisation. Once the conceptual model is defined, it can serve as a basis for the construction of the complete KBS.

4 References

[1] AI Watch, "Special issue on the EuroKnowledge Project", Vol. 4 N8, August 1995.

[2] Aussenac-Gilles N., "How to combine data abstraction and model refinement: a methodological contribution in MACAO", in "A future for Knowledge Acquisition", LNAI N 867 (proceedings of EKAW'94), pp262-282.

[3] Breuker A., Schreiber A., Wielinga B., “KADS: a modelling approach to knowledge engineering”, in 'Knowledge Acquisition and Learning', pp 92-116, 1992.

[4] Fensel D., van Harmelen F., “A Comparison of Languages which Operationalise and Formalise KADS Models of Expertise”, The Knowledge Engineering Review, vol. 9, pp 105-146, 1994.

[5] Istenes Z., “Zola: Manuel de Référence”, (in French) to appear as an internal rapport of IRIN, 1996.

[6] Istenes Z., Tchounikine P., Trichet F., “Using Zola to model Dynamic Selection of Tasks and Methods as a Knowledge Level Reflective activity”, to appear in EKAW'96 Positions Papers (Europeen Knowledge Acquisition Workshop), Nottingham, England, 1996.

[7] Karbach W., Voss A., “Model-K for prototyping and strategic reasoning at the knowledge level”, in “Second Generation Expert Systems”, J.P.David, J.P.Krivine, R.Simmons (editors), pp721-745, Springer-Verlag, 1993.

[8] Linster M., “Linking modelling to make sense and modelling to implement systems in an operational modelling environment”, proceedings of EKAW'92 (LNAI N5999), pp55-74, 1992.

[9] Reinders M., Vinkhuyzen E., Voss A., Akkermans H., Balder J., Bartsch-Sprl B., Bredeweg B., Drouven U., van Harmelen F., Karbach W., Karsen Z., Schreiber G., Wielinga B., “A conceptual modelling framework for knowledge-level reflection”, AI Communications Vol.4 No 2/3, pp74-87, 1991.

[10] Tchounikine P., Choquet C., “Fault diagnosis expert system for robots: a knowledge level prototyping experience”. Proceedings of the Seventh International Conference on Software Engineering and Knowledge engineering (SEKE'95), Rockville-Washington, USA, pp 268-274, 1995.

[11] Tchounikine P., Istenes Z., Rideau S., Trichet F., “Mapcar: a production rule reflective language for expert knowledge representation”, proceedings of ECAI'94 Workshop “Formal specification methods for knowledge based systems”, pp63-77, Amsterdam, Holland, 1994.

[12] Van de Velde W., “Issues in Knowledge Level Modelling”, in “Second Generation Expert Systems”, J.P.David, J.P.Krivine, R.Simmons (editors), pp211-231, Springer-Verlag, 1993.