Evaluating a formal KBS specification language

van Harmelen, F.A.H.; Aben, Manfred; Ruiz, Fidel; van de Plassche, Joke

published in
IEEE expert
1996

DOI (link to publisher)
10.1109/64.482959

document version
Publisher's PDF, also known as Version of record

IEEE Expert, Vol. 11, No. 1, pp. 56-62, Feb. 1996

Evaluating a formal KBS specification language

Frank van Harmelen[1], Manfred Aben[1], Fidel Ruiz[2], Joke van de Plassche[2]

[1] SWI, University of Amsterdam, Roetersstraat 15
1018 WB Amsterdam, The Netherlands
email: frankh@swi.psy.uva.nl.
[2] NICI, University of Nijmegen, The Netherlands

Abstract. In recent years, the knowledge engineering community has begun to explore formal specification languages as a tool in the development of knowledge-based systems. These formal knowledge modelling languages have a number of advantages over informal languages, such as their precise meaning and the possibility to derive properties through formal proofs. However, these formal languages also suffer from problems which limit their practical usefulness: they are often not expressive enough to deal with real world applications, formal models are complex and hard to read, and constructing a formal model is a difficult, error prone and expensive process. The goal of the study presented in this paper is to investigate the usability of one such formal KBS modelling language, called $(ML)^2$. $(ML)^2$ is strongly based on the structure of the knowledge-models used in the KADS KBS development method. We first designed a set of evaluation criteria. We then applied $(ML)^2$ in two case-studies and scored the language on our evaluation criteria. $(ML)^2$ scored well on most of our criteria. This leads us to conjecture that the close correspondence between the informal KADS models and the formal $(ML)^2$ models avoids some of the problems that traditionally plague formal specification languages.

1 Introduction

Formal modelling languages have begun to play an increasing role in the knowledge acquisition community in the last few years, as witnessed by a steady stream of proposals for such formal languages for KBS modelling. A number of these languages are reviewed in [3] and [5]. These modelling languages differ from both the high level informal modelling languages, e.g. as used in KADS [8], and from directly executable languages.

Various authors have argued the advantages of such formal modelling languages: they reduce the vagueness and ambiguity of informal descriptions, they allow for validation of completeness and consistency through formal proofs, and they bridge the gap between the informal model and the design of a system.

[1] The research reported here was carried out in the course of the KADS-II project. This project is partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 5248. The partners in this project are Cap Gemini Innovation (F), Cap Gemini Logic (S), Netherlands Energy Research Foundation ECN (NL), ENTEL SA (ESP), Lloyd's Register (UK), Swedish Institute of Computer Science (S), Siemens AG (D), Touche Ross MC (UK), University of Amsterdam (NL) and Free University of Brussels (B).

However, these advantages come at a price: as is well known from software engineering, these formal languages suffer from problems which severely limit their practical usefulness: they are often not expressive enough to deal with real world applications, formal models are complex and hard to read, and constructing a formal model is a difficult, error prone and expensive process.

In this paper we investigate the usability of one such formal KBS modelling language, called $(ML)^2$ [7]. This language has been developed since 1990, and specifically aims at formalising the KADS model of expertise [8]. We conducted this study at a point when the language definition had become stable, and when the language plus a set of tools to support its use had been applied in a number of cases both inside and outside our research group.

In order to analyse the usability of $(ML)^2$, we proceeded as follows. First of all, we designed a set of criteria to evaluate $(ML)^2$. We then performed a small case-study, constructing an expertise model in $(ML)^2$, in order to try out and refine our evaluation criteria. Subsequently, $(ML)^2$ was used to construct a second model which formed the basis for our language evaluation. These case-studies were used to score $(ML)^2$ on our evaluation criteria.

We assume that the reader of this paper has a basic knowledge of the structure of KADS expertise models. Knowledge of the $(ML)^2$ language is helpful but not required for reading this paper. In order to remind the reader of the central notions of KADS and $(ML)^2$, we give a very brief description of both. More detailed descriptions can be found in [8] for KADS and [7] for $(ML)^2$.

2 Evaluation Criteria

As a first step we designed a set of evaluation criteria that we used to evaluate the usability of $(ML)^2$. Although the main aim of this study was to evaluate a specific formal language, we believe that this list of criteria can be of general use in similar evaluation studies.

Expressiveness. A first concern is whether our language was expressive enough. Were certain things impossible to express? Were some things difficult to express?

Frequency of Errors. One of the problems with formal specifications is that their construction is an error prone activity. What were the most common errors made when using $(ML)^2$? What was the frequency of these errors? Can we identify why these errors occurred so frequently? Can we find a way to avoid these common errors?

Redundancy. To achieve compactness and maintenance, redundancy should be avoided in formal specifications. Was redundancy present in our formal models? Can we identify different types of redundancy? Where does the redundancy occur? Can we think of ways to avoid it? What were the most frequently used constructions in our language? Can we remove or simplify these frequently occurring constructions?

KADS and (ML)^2

A KADS expertise model consists of three layers: domain, inference and task layer (for the purposes of this paper we ignore the strategic layer). The *domain layer* contains a description of the domain knowledge of a KBS application. This description should be as much as possible independent from the role this knowledge plays in the reasoning process. In (ML)^2, such use-independent descriptions of domain knowledge are formalised as a set of theories in order-sorted first-order predicate logic.
The *inference layer* of a KADS expertise model describes the reasoning steps (or: inference actions) that can be performed using the domain knowledge, as well as the way the domain knowledge is used in these inference steps. In (ML)^2, the inference layer is formalised as a meta-theory of the domain layer, and each inference action is represented by a predicate which is axiomatised in an order-sorted first-order theory. The inputs and outputs of an inference action (called knowledge roles) correspond to arguments of these predicates. These roles (terms in the meta-theory) are described in domain-independent terminology, which is connected to domain specific predicates in the domain layer by a naming relation which is specified as a rewrite system (so called lift-operators). The relations between the inference steps through their shared input/output-roles are represented in KADS by a dependency graph among inference steps and knowledge roles. Such a graph is called an inference structure, and specifies only data-dependencies among the inferences, and not the order in which they should be executed.
This execution order among the inference steps is specified at the *task layer*. For this purpose, KADS uses a simple procedural language with primitive procedures to execute inference steps and predicates to test the contents of knowledge roles. These procedures can be combined using sequences, conditionals and iterations. This procedural language is formalised in (ML)^2 through quantified dynamic logic.

Locality of Change. Since formal specifications will have to be refined and maintained, it is important that changes to a formal model remain local. Do changes propagate through the formal models? If so, what were the causes for global changes, and can they be avoided?

Reusability. Reusability of model fragments and of entire models is an important goal in knowledge acquisition. Do our formal models enable reusability?

Guidelines and Tool-Support. In earlier research, we have developed a set of guidelines on how to construct (ML)^2 models, as well as a set of software-tools to support the construction process. Were these guidelines useful? Were there any guidelines missing? Was the toolsupport useful? Were any tools missing?

3 Case Studies

The following case-studies were used to evaluate (ML)^2 on the criteria described above.

Adaptation Study. In this study we took an already existing (ML)^2 model of a simple scheduling task, plus an alternative model of the same task described in a different

language. The task was an incremental procedure for constructing a time schedule, which had been formalised in $(ML)^2$ before we started our study. The alternative model was formalised by others in the language KARL. This alternative model contained a much more elaborate version of the revision subtask. The goal of this study was to adapt the given $(ML)^2$ specification to have the same elaborate subtask as specified in the KARL model, and to observe the effects of these changes on the model as a whole.

Construction Study. In this second study, we performed the process for which $(ML)^2$ is intended: we took an informal conceptual model and constructed the formal version of this model in $(ML)^2$. The particular conceptual model was a simple allocation task. It allocates employees to offices on the basis of a given set of constraints by choosing a complete allocation and subsequently fixing the constraint-violations that occur.

Reusability Study. A third study was aimed at evaluating a library of $(ML)^2$ model-fragments by reusing existing model-fragments for the construction process. In this paper, we do not further report on this study. Details can be found in [4].

We carefully chose all inputs to these case-studies to be of a high quality. The inputs for the adaptation study were reviewed publications, and constructed by experts. The input for the construction study was highly rated by KADS experts. This ensured that any problems found by or during formalisation would not be due to flaws in the conceptual model that could reasonably have been avoided.

4 The Evaluation of the Criteria

Expressivity This criterion is concerned with the ability of $(ML)^2$ to describe KADS models of expertise. We encountered no problems that indicate that parts of KADS models of expertise are impossible to express in $(ML)^2$. The problems concerning the expressivity that we encountered were all the result of errors in the models. The limitations of the formal language reveals errors: something that can not be expressed, is often an error. Therefore, we can conclude that $(ML)^2$ is suitable for describing KADS models of expertise. This is not a very surprising conclusion as the structure of $(ML)^2$ depends strongly on the structure of the conceptual modeling language that is used to describe the models of expertise. It is therefore in principle clear how objects in the conceptual description must be mapped onto $(ML)^2$ constructs.

Errors in the Formalisation Process Not many errors were made during the formalisation process: in the beginning approximately three errors per page of specification text were made, in the last formalisation only approximately one and a half error per page specification text was made. The most frequent errors are typing errors. Most of these can be located easily with the available tools (a parser and a checker for the syntax of $(ML)^2$). It seems therefore that the formalisation process in $(ML)^2$ is not an error prone activity. We suspect that this is mostly due to the fact that the formalisation process is strongly guided by the conceptual model.

Redundancy and Repetition We can distinguish redundance within models and repetition between models. Redundancy within models occurs if some piece of knowledge has to be represented more than once in the same model. The major disadvantage of this kind of redundancy is that it is difficult to modify the knowledge in a model such that it remains consistent. Fortunately, this kind of redundancy was not present in the conceptual models that we studied, and was therefore also not present in the corresponding $(ML)^2$ models.

Repetition between models occurs when the same piece of knowledge has to be represented for all the models. This kind of repetition is especially present in the task layer of $(ML)^2$ models. Although this kind of repetition is not harmful, it is very time-consuming to generate these parts of the model. However, it is not possible to remove these parts as they are necessary for checking the correctness of the model.

A possible solution would be to generate these parts automatically by a tool. We have found templates for modules that could be used for this purpose (initial and intermediate knowledge role and task programs). Especially the template for task programs is suitable for this. This does not remove the redundancy from the model, but it saves much time in the formalisation process and is not hard to realise.

Locality of Changes With locality of changes we mean the amount of actual changes that are necessary to fix an error in an $(ML)^2$ model or to modify an $(ML)^2$ model. In other words, how far the changes propagate through the model. Therefore, this criterion is important for determining the applicability of $(ML)^2$ in practice. Reusability makes it necessary to (partly) modify an existing model for the formalisation of another model. Also it is inevitable that errors will be made while building formal models. However, we do not want changes in our model to affect significant parts of the rest of the model.

The modifications that we applied in the models all had a local character. The reason for this is the restricted interaction between the various layers, and the modularisation within the layers. Changes in the form of the domain knowledge affect only the lift-rules in the inference layer and changes in the task layer are confined to the modules in the inference and task layer which are part of the modified task. Altogether, we can conclude that modifications in the formal model tend to be local.

Reusability Can we reuse (parts of) the specifications of models for the formalisation of other models? Because modifications in the formal model tend to be local, this reusability depends for a great deal on the similarities between models.

There are two ways to syntactically reuse parts of formal models. The first is with templates that we have found for inference and task layer modules. These templates consist mostly of complete modules and not of smaller portions of a module. This is caused by the grainsize of the reusable elements within KADS conceptual models. The templates we found during the formalisation of the office-assignment model were templates for initial and intermediate dynamic knowledge roles, task programs and the task-definitions module. The modules that belonged to one of these classes of modules all had the same structure.

The second way of reusing model fragments are general domain theories for various kinds of knowledge like arithmetic and set theory, and property axioms for relations like

transitivity, reflexivity and symmetry. A library can be constructed from these general theories, which can then be used in other models by selecting the necessary modules from the library and inserting them in the domain layer.

Guidelines Generally, the guidelines for constructing a formal model from a conceptual model were clear and easy to follow. The guidelines are quite extensive; there are over 60 of them. These guidelines are organised in different groups, which affect different parts of the model. The first group prescribes how to transform the structure of the informal conceptual model into a skeletal formal model. The second group of guidelines gives suggestions on how to add additional structure to the formal model and how to specify the signatures, the axioms and the lift-rules.

The main problem with the guidelines was the fact that their application by hand is very time-consuming. We suggest two ways of improving this process. First we can use *templates*, i.e., reusable syntactic constructs that are essentially compilations of the guidelines. A typical example is the combination of a number of guidelines which each suggest a part of a module in the formal specification into a single template that gives the default structure of the entire module. The advantage of such templates, is that larger parts of the specification are generated quickly, the disadvantage is that these larger templates are less generally applicable.

An alternative way to improve the formal model construction process is automated support by a software tool that takes an informal model and semi-automatically creates an initial formal model. Our results with this approach are described in [6]

There are two issues that are not handled by the guidelines. The formalisation of the primitive inference actions is one of them. The guidelines generate the overall structure of the formal model (for instance the connections in the inference structure), but within this structure, the inference actions are left as "gaps" that must still be filled in. Another lack in the support of the guidelines is the modularisation of the domain layer.

5 How formalisation reveals errors

Since we deliberately chose our input models to be of high quality, it is remarkable that the formalisation of the office assignment model revealed many errors in it. We must therefore conclude that formalisation reveals certain aspects that can not be seen in the conceptual model. A logical question is then how formalisation reveals these errors.

We found four ways that formalisation helps in finding errors. The first is that because of the error, a part of the model can not be described in $(ML)^2$. The second is that the detailed examination of the model that is necessary for the formalisation reveals the error. In this case, the erroneous part of the conceptual model can be formalised, but this gives a different interpretation to the model than the intended one. The third is that the formal specification reveals the grainsize of inference steps, which may then turn out to be too complex to be primitive, or too simple for inclusion in the formal model. The final and fourth way of finding an error is that formalisation may reveal redundant parts in the specification, which are repetitions of other parts of the model.

We will take a look at the errors that were found by the formalisation of an office assignment task. This task attempts to assign a given set of employees to a given set of

offices under a given set of constraints. We will use one part of this model to illustrate the errors that can occur, namely the revise task.

The office assignment task

The office assignment task takes the following inputs:

- a floorplan of a building indicating the layer of a set of office rooms;
- properties of these offices (e.g. the size of the offices, which offices have computer connections, which offices are noisy);
- a set of employees that must be allocated to the offices;
- properties of each employee (e.g. are they smokers or not, are they a manager, a secretary or a programmer, what projects do they work on)
- constraints that are emposed on the office assignment, such as: smokers must not share with non-smokers, a manager must be close to the secretary, a programmer needs a computer connection, etc.
- requests which should be fulfilled if possible (e.g. mr. X wants to share an office with mrs. Y, the secretary would like a quiet room).

Given these inputs, the office assignment task is to find an allocation of all employees to an office in such a way that no constraint is violated, and that as many requests as possible are fulfilled.
The task is established through a simple propse-test-revise cycle: the propose subtask generates a complete assignment (i.e. a room allocation for each employee) without considering the constraints and requests. The test substask then tests for any violations of constraints and requests in this proposed solution. If a constraint turns out to be violated, a fix is proposed in the revise task.
Since the revise task is used to illustrate the error-types that were uncovered through formalisation, we describe it in some more detail. Fig. 1 shows the inference structure of the revise task.
The only fix that the revise task proposes for a candidate solution that violates a constraint is an exchange of two employees. Such an exchange is called a transformation. Such a transformation is constructed using the set of conflicts (= the assignments that violate constraints) and the currently proposed, incorrect solution. This transformation is used to compute the old local situation, i.e. the two assignments that are going to be switched, and the new local situation, i.e. the two assignments after the exchange. These two situations are compared with respect to the number of requests and constraints that they violate. The local situation with the least constraints and request violations is used to construct a new proposed solution.

5.1 Incorrect Models can not be Formalised

As the expressiveness of any formal language is less than that of natural language, there are certainly models that are expressible in natural or informal language, but that are not expressible in a formal language. Ideally, the formal language would allow a way to write down all methodologically correct models, and disallow all methodologically incorrect models. In our case studies we found that in all cases where $(ML)^2$ precluded

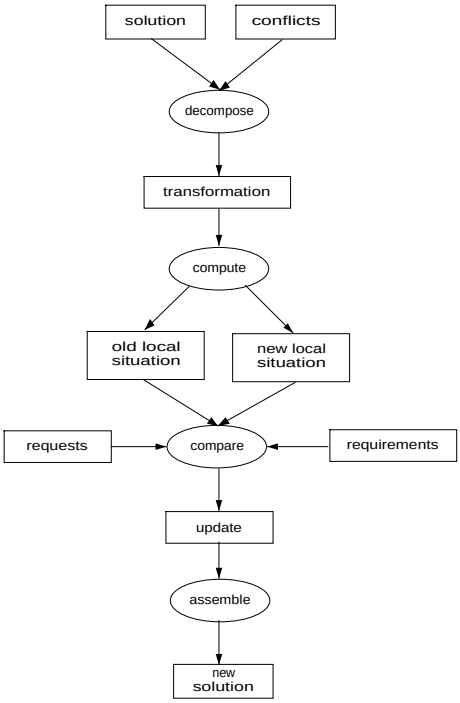

Fig. 1. *the revise task of the office assignment model*

us to straightforwardly formalise the models, it turned out that these models were methodologically unsound.

Missing dependencies are the best example of errors that were revealed because it was impossible to formalise the conceptual model. Missing dependencies occur when an output knowledge role of a primitive inference action can not be computed from its input knowledge roles. Sometimes it is difficult to see from the conceptual descriptions what the contents of the knowledge roles is. Therefore, it is easy to make these kinds of errors. Such an error can only be solved by finding the right knowledge roles that contain this knowledge and making this an input to the inference action.

One example of such a missing dependency is concerned with the inference action `compare`. This inference action compares the two local situations with respect to the constraints. However, there are also assignments that are not restricted to these local situations, but involve also other assignments of the solution. As a consequence, we need the knowledge role `solution` as an input to the inference action `compare`. The improved inference structure is shown in figure 2.

5.2 Detailed Examination reveals Errors

The detailed examination that is necessary for the formalisation of a model often reveals errors. In this case, the erroneous part of the conceptual model can be formalised, but

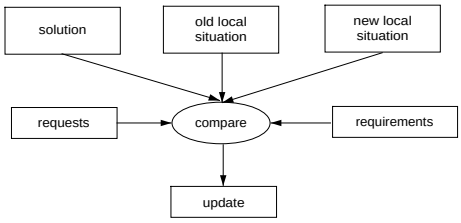

Fig. 2. *the corrected inference structure of compare*

this gives a different interpretation to the model than the intended one.

An obvious objection to the use of formal methods to find imperfections in the informal model would be that taking a closer look at the informal model would have revealed the errors anyway. Our response to this is twofold. First, we took high quality informal models as a starting point: experts in the KADS method did not find flaws in the model. Second, formal methods are a tool for having a closer look at the model, providing rigorous support where informal methods apparently fail. Another objection would be to say that implementation of the informal model would have revealed the imperfections anyway. In fact, a detailed look at an implementation of the office allocation task shows that the implementation is unfaithful to the informal model at exactly those places where we found the imperfections in the informal model. Indeed, the programmers discovered the errors, but since they encoded a correction in the implementation, the implementation does not correspond to the specification anymore, a situation that is highly undesirable as it hampers maintenance and documentation.

The errors that were revealed in this way are confusing names of inferences and control knowledge that was modelled on the inference layer instead of on the task layer.

Names of Inferences are Confusing. This is one of the most frequent errors. The involved inferences had names that were inconsistent with the conceptual descriptions of these inferences in a KADS reference document [1]. The names of the inference action's did not match with the description of the inference actions that took place. This can be very confusing, because it is not clear what inference should be formalised: the inference which is described in the conceptual model or the inference with the same name in the reference document.

The causes of these errors are the vagueness of the model of expertise and the ambiguous description of the library inferences in the reference document, which are susceptible to various interpretations.

An example of a confusing name of an inference is `decompose`. The description of this inference action in the conceptual model says that a transformation is proposed which could solve the conflict. This proposition is done by selecting a conflict (= an assignment that violates a constraint) from the set of `conflicts`, and an assignment from `solution` that is different from the selected conflict. These two are used to construct a transformation. However, the definition of decompose according to the reference document is to choose a set of components from some composite structure. This definition does not match with the conceptual description of the inference action. It seems that

the decompose in the conceptual model is a combination of two selects, for selecting a conflict and an assignment, and an assemble, for constructing a transformation.

The Inference Layer contains Control Knowledge. This kind of error occur because control dependencies between modules are represented on the inference layer instead of on the task layer. Often these dependencies are described as conditional actions in the inference action: the decision *when* an inference has to be applied is also part of the inference. Such an error can be solved very easily by modeling this dependency on the task layer.

5.3 Formal Specification reveals the Grainsize of Inferences

In the informal model, specification stops where the knowledge engineer is not interested in further detail. Formalising the informal model by definition adds more detail to the informal model, and reveals differences in complexity of the various inferences in the informal model. Some inferences become very complex, others turn out to be trivial.

Inferences may be Trivial. As formalisation gives us the means to compare inferences and their relative complexity, it can help identifying inferences which "do not do anything", i.e., are formally redundant. In these inferences, the output is equal to the input (mostly there is only one input knowledge role). The cause of these trivial inferences are the vague conceptual descriptions of the input and output knowledge roles, which make it difficult to see the similarities between the two. When such a trivial inference is found it is a modelling decision to remove it or not. There may be conceptual reasons to maintain such inferences. As the inference is trivial the removal of the inference will have no effect on the rest of the model. These errors are revealed clearly in the formal specification of an inference action as it has no body.

The `compute` inference is such a trivial inference. This inference computes two local situations, the old one and the new one. The old local situation consists of the two assignments that are going to be switched, the new local situation of the two assignments after the switch. The computation of the old local situation is, however, redundant as this situation consists of the same assignments as those of the transformation. Removing this output of the inference results in the inference structure that is shown in figure 3.

Inferences may be too Complex. When by formalisation an inference turns out to be too complex, it is usually an indication that the conceptual model is not understood well enough. Although it is a modeling decision when to stop decomposing a model, formalisation may show that not all "leaves" of the model are equally primitive. This is usually an indication of complexities in the model that are overlooked in the informal expertise model. If we would have chosen to formalise these inferences, then the formal specification would have been at least two to three times larger than specifications of other inferences. Often it is possible to identify separate parts in the inference. It is better to split up the complex inference in these different parts. This makes the inference structure more clear.

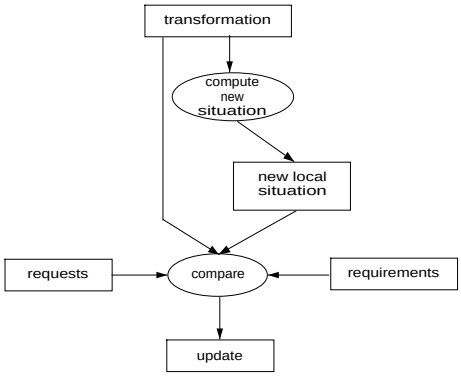

Fig. 3. *the inference structure after removing the computation of the old local situation*

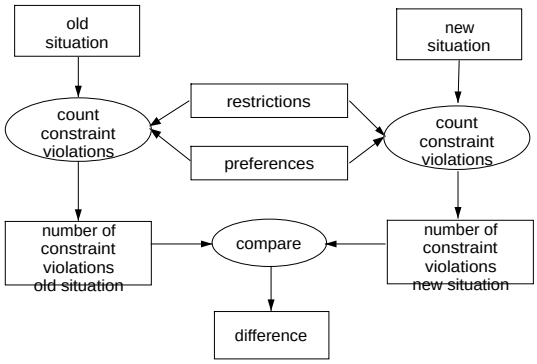

Fig. 4. *the refinement of the inference action* `compare`

An example of such a complex inference is shown in figure 4, where the inference action `compare` compares the number of violations of an old and a new situation. This inference action can be split up in counting the number of violations for the new situation, counting the violations of the old situation, and comparing these two. This is shown in figure 4.

5.4 Formal Specification reveals Redundancies

Formal specifications also reveal redundancy by establishing that two model fragments are identical, and the inference structure could be improved by removing one of them and restructuring the model.

This can be seen in the revise task of the office assignment model, as it does not completely correspond with the overall interpretation of a propose-test-revise cycle. Normally, in such a cycle, the test part would examine if the application of a transformation that is proposed in the revise task yields a better solution. However, we can see in figure 1 that the revise part of the office assignment model incorporates another test

phase. This makes the test part of the office assignment model redundant. The removal of this extra test from the revise task results in the inference structure that is shown in figure 5.

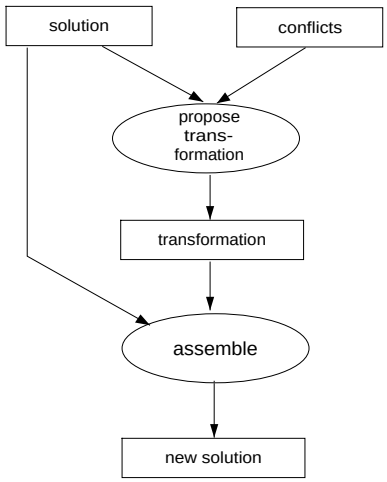

Fig. 5. *the inference structure of revise without the extra test*

6 Conclusions

In the above, we have evaluated the usability of the formal modelling language $(ML)^2$ by designing a set of evaluation criteria and applying these criteria to $(ML)^2$ in a number of case-studies. The evaluation criteria were: expressivity, frequency of errors, redundancy, locality of changes, reusability and guidelines/support. We have seen that $(ML)^2$ scored well on most of these criteria. We contribute these relatively positive results to the close connection that exists between the structure of the informal KADS models and the formal $(ML)^2$ models.

Concluding, we can say that formal specification for KBS modelling is both desirable and possible. Formalisation is *desirable* because it reveals errors in conceptual models, even when these were thought to be of high quality. Formalisation is also *possible*, but only when extensive support is given in the form of guidelines and software-tools. The close structural relation between informal and formal model makes it possible to give such extensive support for a number of aspects of the formalisation process.

Besides these positive conclusions concerning desirability and feasibility of using formal languages, there is also a negative conclusion from our work. Our case studies have revealed difficulties with the reuse of model fragments. The ambiguous interpretations of the informal model fragments endangers the reuse of both formal and informal model fragments. Further study is certainly needed in this area.

Related Work A recent study of the use of formal methods in industry can be found in [2] The recommendations of this study can be summarized as follows. First, the acceptance of formal methods must be improved by integrating the formal methods in a broader methodology and by developing notations that can be used by non-logicians. Second, tools must be robust, and tool support in validation is especially weak. $(ML)^2$ attemps to address both these problems. First, $(ML)^2$ is embedded in the KADS method, and especially designed for that method. The notations in $(ML)^2$ are tuned towards knowledge engineers that are familiar with the KADS method, and avoid the in depth knowledge required for understanding the underlying formal logics. Second, the embedding of $(ML)^2$ in the KADS method allows more informed toolsupport.

Limitations and Future Work The first limitation of the work described in this paper is that we concentrated mainly on the transformation of the informal model to the formal model. Another goal of the formal model is to bridge the gap between informal model and design model. Future research should focus on the question whether the transformation of the informal model to the design model through the formal model is easier and/or gives better results than the direct transformation of the informal model to the design model. The second limitation is that the two models that we used had a restricted domain layer because of the precise knowledge that is necessary for the task. Therefore, our evaluation focussed mainly on the inference layer. Formalisation of a model that has a more elaborate domain layer could focus more on the evaluation of the formal specification of the domain layer.

Acknowledgements. We are grateful to Anjo Anjewierden and Gertjan van Heijst for their critical comments on an earlier version of this paper.

References

1. J. A. Breuker, B. J. Wielinga, M. van Someren, R. de Hoog, A. Th. Schreiber, P. de Greef, B. Bredeweg, J. Wielemaker, J. P. Billault, M. Davoodi, and S. A. Hayward. Model Driven Knowledge Acquisition: Interpretation Models. ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd, 1987.
2. D. Craigen, S. Gerhart, and T. Ralston. An international survey of industrial applications of formal methods. Technical report, U.S. Department of Commerce, Technology administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA, March 1993.
3. D. Fensel and F. van Harmelen. A comparison of languages which operationalise and formalise KADS models of expertise. *The Knowledge Engineering Review*, 9:105–146, 1994.
4. F. Ruiz, F. van Harmelen, M. Aben, and J. van de Plassche. Evaluating a formal specification language. In L. Steels, A.Th. Schreiber, and W. Van de Velde, editors, *A Future for Knowledge Acquisition, Proc. 8th EKAW*, number 867 in Lecture Notes in Artificial Intelligence, pages 26–45. Springer-Verlag, 1994.
5. J. Treur and Th. Wetter, editors. *Formal Specification of Complex Reasoning Systems*, Workshop Series. Ellis Horwood, 1993.
6. F. van Harmelen and M. Aben. Structure preserving specification languages for knowledge-based systems. *International Journal of Human Computer Studies*, 1995. (Formerly Journal of Man Machine Studies).

7. F. van Harmelen and J. R. Balder. $(ML)^2$: a formal language for KADS models of expertise. *Knowledge Acquisition*, 4(1), 1992. Special issue: 'The KADS approach to knowledge engineering', reprinted in *KADS: A Principled Approach to Knowledge-Based System Development*, 1993, Schreiber, A.Th. *et al.* (eds.).
8. B. J. Wielinga, A. Th. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–53, 1992. Special issue 'The KADS approach to knowledge engineering'. Reprinted in: Buchanan, B. and Wilkins, D. editors (1992), *Readings in Knowledge Acquisition and Learning*, San Mateo, California, Morgan Kaufmann, pp. 92-116.