Evaluating a Formal Specification Language

Ruiz, F.; van Harmelen, F.A.H.; Aben, M.; van de Plassche, J.

published in
A Future for Knowledge Acquisition Proc. 8th EKAW
1994

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

Proc. of the 8th European Knowledge Acquisition Workshop (EKAW'94)
Steels, Schreiber and Van de Velde (eds.), LNAI 867, pp. 26-45, Springer Verlag 1994

Evaluating a formal modelling language *

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

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

Abstract. 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 $(ML)^2$. In order to analyse the properties of $(ML)^2$ that influence its usability, we designed a set of evaluation criteria. We then applied $(ML)^2$ in two case-studies and scored the language on our evaluation criteria. A separate case-study was devoted to analysing the possibilities for reusing formal model fragment. $(ML)^2$ 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. The case-studies revealed problems with the reuse of formal model fragments. These problems were caused by the (inevitable) ambiguous interpretations of the informal model fragments. Finally, extensive software-support is required when constructing formal specifications. Our case-studies showed that the close correspondence between formal and informal models makes it possible to provide more support (and particularly: different kinds of support) than have traditionally been considered.

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 [7, 10, 22, 19, 21]. (e.g. KARL, K_{bs}SF, FORKADS, $(ML)^2$, MoMo, DESIRE). These modelling languages differ from both the high level informal modelling languages, e.g. as used in KADS [23], and from the directly executable languages [12, 14].

* 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).

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.

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.

1.1 Goal of this Study

The goal of the study presented in this paper was to investigate the usability of one such formal KBS modelling language, called $(ML)^2$. This language has been developed since1990, and specifically aims at formalising the KADS model of expertise. 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 SWI. [4, 24, 18, 8].

1.2 Approach to this Study and Structure of this Paper

In order to analyse the properties of $(ML)^2$ that influence its usability, we proceeded as follows. First of all, we designed a set of criteria to evaluate $(ML)^2$. These are described in section 2. Subsequently, we 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 (described in section 3) were used to score $(ML)^2$ on our evaluation criteria. The results of this evaluation are reported in sections 4 and 5.. Since reusability is an important topic in knowledge acquisition, we also wanted to study the extent to which a library of model fragments could have been used to construct formal expertise models. The result of this third study is reported in section 6. Section 7 discusses the lessons on tool support for $(ML)^2$ that we learned from these studies and section 8 concludes.

1.3 Brief Description of KADS and $(ML)^2$

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 [23] for KADS and [19] for $(ML)^2$.

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 [9].

2 Evaluation Criteria

In this section we describe the 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. For reasons of 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?

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 [2, chapter 3], as well as a set of software-tools to support the construction process [2]. Were these guidelines useful? Were there any guidelines missing? Was the toolsupport useful? Were any tools missing?

3 Case Studies

In this section we describe the three case-studies that were used to evaluate $(ML)^2$ on the criteria from the previous section.

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 the incremental propose-test-revise model described in [20], and its $(ML)^2$ formalisation was taken from [4]. The alternative model was formalised in the language KARL and taken from [13]. 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, taken from [11]. 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. Whereas the first two studies were aimed at evaluating $(ML)^2$ by adapting an existing model or constructing a new $(ML)^2$ model, this third study was aimed at evaluating a library of $(ML)^2$ model-fragments by reusing existing model-fragments for the construction process. The basis of this study was the library of formalised inference schemes described in [1]. This library contains formal descriptions of 41 inference schemes, organised in 16 classes of a refinement hierarchy. This study was divided into three separate substudies:

– In the first substudy, we took a number of formal inferences from the models constructed in the adaptation and construction studies, and we analysed whether these inferences could have been obtained by selecting and adapting inference schemes from the library. The result of this study was a a set of adaptation steps that would yield the desired inferences when applied to schemata selected from the formal library.

– In the second substudy we took a number of simple informal inferences from the conceptual model of the allocation study, and investigated whether a formal version of these inferences could be obtained by first selecting formal schemata from the library and subsequently applying the repertoire of adaptation steps from the first substudy to these inference schemes from the library.
– The third substudy was as the second, but this time for more complex inferences.

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 Evaluation of $(ML)^2$

Since we deliberately choose 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 the office assignment model which was built for the Sisyphus task [15]. 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. We will give a short description of this part of the model.

The office assignment model consists of a propose-test-revise cycle. The propose task generates a complete assignment without considering the constraints. This assignment is tested on the constraints in the test task. If a constraint is violated, a fix is proposed in the revise task. The inference structure of the revise task as it appeared in the conceptual model is shown in figure 1.

The only fix that the revise task proposes for a 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 current, 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 requirements that they violate. The

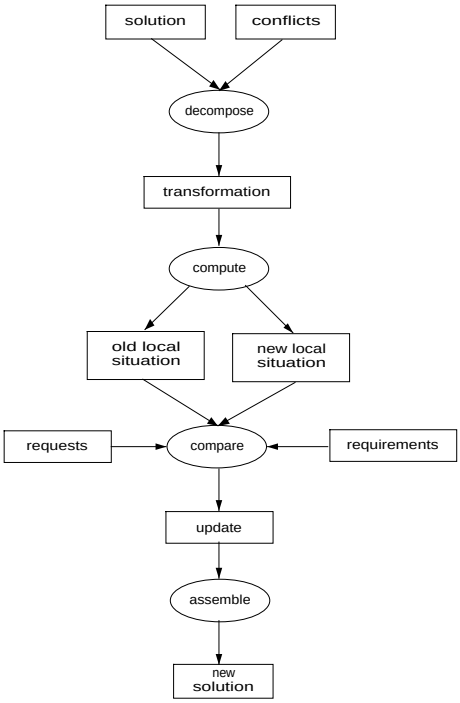

Fig. 1. *the revise task of the office assignment model*

local situation with the least requirement and request violations is passed to `update`. Eventually, this update is used to construct a new solution.

4.1 Incorrect Models can not be Formalised

As the expressivity 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 us to straightforwardly formalise the models, it turned out that these models where 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.

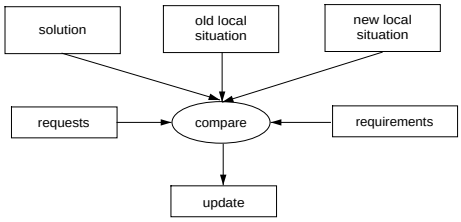

Fig. 2. *the corrected inference structure of compare*

4.2 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 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 methodology 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 the implementation of the office allocation task (also given in [11]) 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 authors 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 from a methodological viewpoint 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 [5]. 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 [5].

The causes of these errors are the vagueness of the model of expertise and the ambiguous description of the library inferences in [5], which are susceptible to various interpretations[3].

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 [5] 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.

4.3 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

[3] In KADS-II this is solved by making a distinction between the type and the name of the inference. The name can be given freely by the domain expert, and the type of the inference maps on an inference from the library.

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.

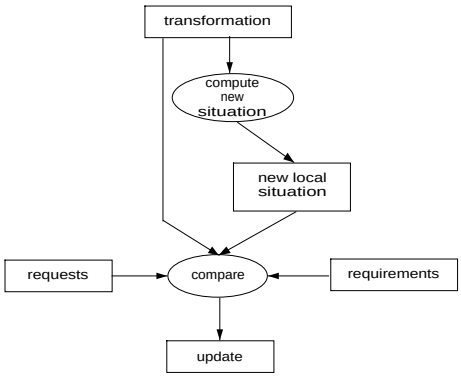

Fig. 3. *the inference structure after removing the computation of the old local situation*

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.

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

4.4 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.

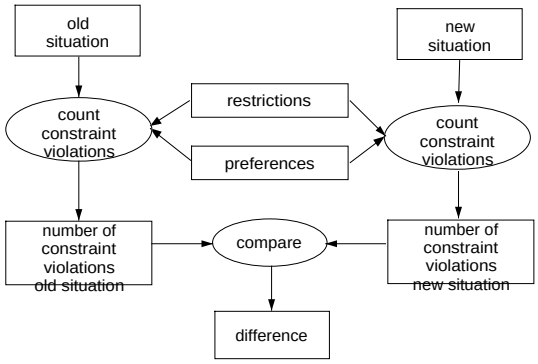

Fig. 4. *the refinement of the inference action* `compare`

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.

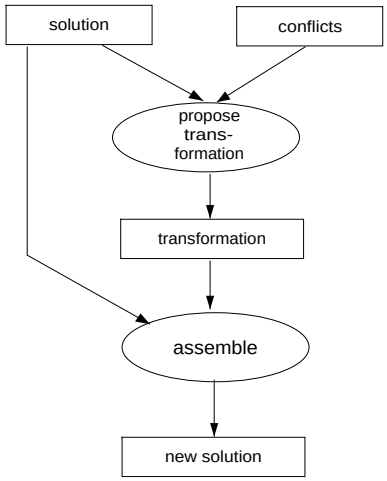

Fig. 5. *the inference structure of revise without the extra test*

5 The Evaluation of the Criteria

Expressivity This criterion, as we have seen before, 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 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 it is 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 do 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 that changes in our model 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 that we can syntactically reuse parts of formal models. The first is with the 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 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.

Here we described reusability with respect to syntactical constructs. In section 6 we will take a look at reusability in a broader context.

Guidelines Generally, the guidelines for constructing a formal model from a conceptual model [2, chapter 3] 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 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 creates an initial formal model. In section 7 we will examine this in more detail.

There are only 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. In the next section we will examine if a formalised library of inferences can support this part of the formalisation process. Another lack in the support of the guidelines is the modularisation of the domain layer.

6 Supporting the Formalisation of the Inferences

Whereas the first two studies were aimed at constructing a new $(ML)^2$ model and adapting an existing model, the third study was aimed at reusing existing model-fragments for the construction process and also to examine if these model-fragments can fill the gap in the support of the formalisation process that we identified in the previous section. This gap concerned the formalisation of the inference actions.

In this section we will evaluate the support for the formalisation of the inferences that is given by the formalised library of inferences. This formalised library consists of adaptable components and not 'ready-to-use' components. These components are called inference schemata [1].

There are two goals for a library of formalised inferences. The first is to remove ambiguity from the conceptual descriptions of the library inferences. The second goal is to support the formal specification of inference actions in a model through the selection and adaptation of inference schemata. The inference schemata can be compared much easier than the informal descriptions, because of their formal character. Therefore, the consequences of selecting one schema instead of another is made more clear.

An inference schema is represented as three constituents: a *precondition*, a *body* and a *postcondition* [1]. The *precondition* describes the conditions under which the inference is applicable. The *body* of the inference schema declaritively describes the operation that is performed by the inference on the input knowledge roles to compute the output knowledge role. The *postcondition* describes the properties that hold after the application of the inference.

The selection of the appropriate inference schema or combination of inference schemata can be done on the basis of the postcondition. We compare the postconditions of the inference action with the postconditions of the inference schemata. The inference schema that matches the postconditions of the inference best is selected.

After the inference schema or combination of inference schemata is selected, it has to be differentiated further. The modifications that we apply are strengthening (making a formula more specific), weakening (the inverse of strengthening thus making the formula more general) and reformulation (a syntactical operation that does not affect the strength of an inference). These modifications can be applied to the whole inference schema or to the separate constituents. It is also possible to strengthen some constituents

and weaken others. These different modifications result in variations of one inference type.

6.1 Finding the Adaptation Steps

In the first substudy, we used the formal versions of some `select` inferences from the two models of the previous case studies to select an inference schema. These `select` inferences select an element from a given set according to a selection criterion. We adapted these general schemata to fit the specific inferences and examined which adaptation steps they had in common. We used the formal version of these inferences so that we could guide the adaptation process by making the adapted inference schema the same as the inference in the formal model.

The six generic adaptation steps that we found were:

1. Instantiation of the selection criterion or principle (strengthening).
2. Unfolding the definition of an object (reformulation).
3. Folding the definition of an object (reformulation).
4. Removal of a parameter from the head of the body (weakening).
5. Addition of a parameter to the head of the body (strengthening).
6. Joining two (or more) inference schemata into one inference (reformulation).

There were some other adaptation steps, but these were not generic.

In the second substudy, we examined if these adaptation steps could be used to formalise a `select` inference starting from the conceptual description. This resulted in a formal specification of this inference that was better than the original formalisation. The new formalisation corresponded better to the conceptual description of the inference. So it seems that using the library of inference schemata and the adaptation steps we found is a good way to formalise simple select inferences.

6.2 Applying the Adaptation Steps on Complex Inferences

The goal of the third substudy was the same as for the second one, but now we tried to formalise more complex inferences. Using the formalised library by selecting inference schemata and applying the adaptation steps that we found in the previous stage did not succeed for the formalisation of these complex inferences. The problems that we encountered had two causes. The first was that the interpretation of the *names* of the inferences in the models did not match with the standard interpretation. Therefore, it was not possible to rely on the name of an inference for the selection of an inference schema. This is not a severe problem, as selection can also take place on the basis of the contents of the inference action that has to be formalised.

The second cause is more severe, and was that the *contents* of inferences did not match with any inference schema or combination of inference schemata in the library. This made it impossible to formalise these inferences with the help of the formalised library. As these inferences were not primitive inferences, we had to change our inference structure to make it possible to formalise it using the formalised library. This modification of the inference structure is a creative process and is difficult to support. As a consequence, the formalisation of these inferences cost more effort than expected.

Conclusions regarding Reusability. We can conclude from the previous section that the first goal of the formalised library of inferences, namely to remove ambiguity from the descriptions of the library inferences, has been realised successfully, as it was not difficult to find the corresponding inference schema of an inference if there was one.

The second goal of the formalised library has not been realised yet. The formalised library could not give enough support for the formalisation of inferences by *selecting* and *adapting* inference schemata. The cause for this is that the informal model is constructed using the informal library of inferences. The elements from this informal library have various different interpretations. This makes the mapping on the formal library elements difficult, as the formal library incorporates one specific interpretation of the informal library elements.

In other words: not all interpretations of inferences in the conceptual model could be found in the library (the second goal), but those elements that could be found in the library were sufficiently disambiguated (the first goal).

The differing interpretations of the standard inferences in the conceptual model endanger reusability of individual inferences, as it is not clear what a certain inference does. They also endanger reusability of entire interpretation models, as it is probably not known which interpretations of the inferences are used in a model and it is therefore questionable if it is reusable.

As reusability is a crucial aspect of the KADS methodology, this problem will have to be solved. As a solution to this problem, we propose that conceptualisation and formalisation must be viewed as an iterative and not a sequential process. The conceptualisation phase sets up only the global structure of the model, which is then further refined during conceptualisation/formalisation cycles. If formalisation of a model is not possible with the formalised library, then the inference structure must be modified in another conceptualisation phase (and maybe consulting the domain expert). This modified structure is then formalised.

There are some aspects to this solution that need further research. It is obvious that formalisation should follow up the conceptualisation phase sooner, but it is not clear how much sooner this is and how this affects the separate phases. Also research is needed for determining the best support for the formalisation using the inference schemata.

7 Requirements and Evaluation for Software Support

The amount of detail required to ensure the completeness and correctness of a formal model results in two difficulties with formal specification. First, the specifications tend to be big and second, adding the required detail is not always trivial.

Tools and methodology should help in overcoming these difficulties as much as possible. In the following we will analyse the required functionality of software tools, and discuss a number of tools that we used during the experiments. We give some suggestions for more tool support, where we found it lacking during the experiments.

7.1 Current Software Support Tools

Formal specification consists of three phases:

1. Building an *initial* formal specification on the basis of the informal model,
2. *Refining* the formal model until it is sufficiently detailed (e.g. enough to serve as a design document, or such that a certain amount of trust is endowed in the specification),
3. *Analysing* the model for its correctness and completeness.

These phases are often cyclic. Typically, phase 2 depends strongly on the results of phase 3. For each of these phases software support is required. The sheer size and complexity of an average formal specification make it impracticable to write it down and evaluate it with pen and paper. We will describe the required functionality of the tools that support these phases, and we will indicate where such support is already available and evaluated in the course of our evaluation of $(ML)^2$.

The $Si(ML)^2$ toolset [2, chapter 2] has been developed in the course of the KADS-II project. $Si(ML)^2$ consists of a number of tools written in Prolog that support the required functionality sketched above. The toolset uses the facilities of GNU Emacs-19 for interfacing and editing. This toolset has been used during all the experiments and will therefore be discussed in more detail. We will discus the required tool support for the three phases in formal model construction, and indicate which functionality is realised in $Si(ML)^2$.

Initital Formal Model Construction. The result in KADS of the conceptual modelling phase is a structured expertise model. This model is the basis for subsequent formalisation. As we have seen, a large number of guidelines have been developed to help generating an initial formal model on the basis of the expertise model. These guidelines can to a large degree be automated. Many of the guidelines suggest a number of default formalisations, and decisions that are taken in some guidelines determine the decisions that are to be taken in other guidelines. An automated tool can keep track of implications of applying guidelines in a certain way, can avoid errors introduced by manually applying the guidelines and and can help to preview the results. The *transformation tool* included in $Si(ML)^2$ realises this functionality, thereby reducing the number of errors introduced by applying the guidelines manually. The informal constructs that cannot be formalised automatically are inserted as comments at the appropriate places in the formal model, making the specification self-contained and well-documented. For all of the guidelines the tool presents defaults. In combination with an adjustable level of user-interaction, the tool can generate a skeletal model automatically or semi-automatically. The tool became available during the experiments, and is therefore not included in the evaluation. The activities during the case-studies, however, clearly showed the need for such a tool, and in part inspired its construction.

Formal model refinement. The transformation tool generates the initial formal specification. This initial specification contains "holes" where the informal model consists of informal, unstructured text. Formalising these parts is a creative enterprise, and can not be fully automated. However support is still required and feasible. First of all, we need good editing support that can be used to manage the complexity of the formal model. Second, we need support for reusing formal specification fragments, such as the inference schemes.

To manage the inherent complexity of the formal model we need to selectively display parts of the formal model. For instance, when we are sufficiently satisfied that the signatures of the formal modules are complete, we may want to hide them in the editor, such that we can focus on those parts that are still incomplete. Normalisation of the layout and pretty printers make it possible to keep the specification readable.

The $(ML)^2$ *editor* in $Si(ML)^2$ is a customization of GNU Emacs. The editor supports *selective display* which turned out to be very useful for keeping an overview of the specification. Besides the standard navigation facilities offered by Emacs, the editor makes it possible to go through an $(ML)^2$ specification in sensible jumps, e.g., to go to the next module or layer. The editor also supports hypertext navigation through the specification, e.g., to follow import links, or to jump to the place where some term is defined. As the editor constitutes the main interface to all functionality of $Si(ML)^2$, the editor provides functionality to select parts (modules, layers, axioms, etc.) of the specification and subsequently perform an action upon them (e.g., to parse it, to pretty-print it). The pretty printer can be used to normalise layout of the specification in the editor, and to generate LaTeX text.

Apart from managing the complexity of the specification, tool support is required for reusing formal specification fragments stored in a library. Currently the library of inference schemes is not supported by any software tool.

Analysing the specification. We can analyse the (intermediate) results of the fomal specification in various ways. We want to verify the syntactical correctness and to perform static analysis, such as type checking, connections between modules in the specification etc. A third way to analyse the specification is dynamic assessment of the problem solving competence of the formal model.

The $Si(ML)^2$ *parser* is a context-free definite clause grammar parser. The EMACS interface can visualise the parsing process, by using the cursor to display the parsing point. This facilitates finding the cause of a parsing-error. $Si(ML)^2$ includes a *parser generator*, that takes as input a BNF language definition in LaTeX and produces a Prolog parser for that language. This ensures that the definition of the language and the parser for the language are always consistent with each other.

After a text has been parsed, and syntactically accepted, the static semantics of the specification are checked, such as correct typing, valid import relations, defined terms etc. The *checker* generates error messages, which can be interpreted by the editor, allowing to jump to the cause of the error. The tool also heuristically suggests modifications to the specification that would remedy the errors found.

$Si(ML)^2$ contains a *theorem prover* (based on PTTP [16]), which can be used to simulate the execution of a task, and to prove properties of the specification. This part of the toolset was not evaluated in the course of this research.

7.2 Suggestions for Further Support

In the above we described the required functionality for software support tools, and how the $Si(ML)^2$ toolset realises this functionality. There is still some functionality lacking which should be incorporated in a toolset to support the construction of formal specifications.

The main lack in the Si(ML)^2 toolset was *graphical visualisation* and editing of the formal model. We found ourselves sketching the structure of the formal model by pencil and paper, to maintain an overview over the specification. Experiences with tools that do offer graphical support (such as TheME [3]) show the benefits of graphical support is for the construction process of formal specifications.

Another improvement of the toolset would be the ability to show the *evaluation status* of parts of the model. The toolset supports incremental parsing of the specification, but does not show which modules are syntactically accepted, or found to be (in)correct by the type checker.

The selective display that Si(ML)^2 supports is based on the syntactic structure of the specification. One could envisage the support of various *views* on the formal specification. For instance, there may be a view that is more understandable to the domain expert, or a view that is tuned towards the programmer of the target KBS. Such views would not only require hiding detail, but would also require different representations of parts of the formal model. Examples are graphical representations or informal textual *summaries* of specification fragments, or specific syntactic sugar. The editor and parser already support inclusion of *comments* in the specification, and the transformation tool includes CML text as comments, so that could be a starting point.

As remarked above, software support for the selection and adaptation of inference schemes is also neccessary. A library tool requires functionality for selecting and adapting the inference schemes. Given the formal structure of the inference schemes, we envisage theorem proving support for selection and for suggesting adaptations. A library tool should also support the extension of the library with fragments and with adaptation steps.

8 Conclusions

In this paper we investigated the usability of the formal modelling language (ML)^2. We did this 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. In section 5 and 6 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. The case-studies revealed problems with the reuse of formal model fragments. These problems were caused by the (inevitable) ambiguous interpretations of the informal model fragments.

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. In section 6 we have sketched how the problems with reusability of model-fragments might be solved, but further study is certainly needed in this area.

Related Work A recent study of the use of formal methods in industry [6] 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 methodology, and especially designed for that methodology. The notations in $(ML)^2$ are tuned towards knowledge engineers that are familiar with the KADS methodology, and avoid the in depth knowledge required for understanding the underlying formal logics. Second, the embedding of $(ML)^2$ in the KADS methodology allows more informed toolsupport.

Limitations and Future Work The first limitation of the research outlined in this paper is that we concentrated mainly on the transformation of the informal model to the formal model. As 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. The last topic that needs to be further researched concerns the reusability. The solution that we proposed in this paper is that the relation between conceptualisation and formalisation should not be sequential but iterative. Further research is necessary to examine the effect of this lifecycle to the separate processes.

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

References

1. M. Aben. CommonKADS inferences. Report KADS-II/M2/TR/UvA/041/1.0, June 1993.
2. M. Aben, J. Balder, and F. van Harmelen. Support for the formalisation and validation of KADS expertise models. Report KADS-II/M2/TR/UvA/63/1.0, January 1994.
3. J. R. Balder and J. M. Akkermans. TheME: an environment for building formal KADS-II models of expertise. In *Proceedings of the 12th Int. Conf. on Expert Systems and their Applications*, Avignon, 1992. Also in: AI Communications, 5(3), 1992.
4. J. Balder, F. van Harmelen, and M. Aben. A KADS/ML^2 model of a scheduling task. In Treur and Wetter [17], pages 15–44.

5. 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.
6. 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.
7. D. Fensel, J. Angele, and D. Landes. Knowledge representation and acquisition language (KARL). In *Proceedings 11th International workshop on expert systems and their applications*, pages 821–833, Avignon, France, May 1991.
8. J. Fox. On the soundness and safety of expert systems. *AI in Medicine*, 5:159–179, 1993.
9. D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol. II*, pages 497–604. Reidel, Dordrecht, The Netherlands, 1984.
10. L. in 't Veld, W. Jonker, and J.W. Spee. The specification of complex reasoning tasks in K_{BS}SF. In Treur and Wetter [17], pages 233–256.
11. J. Kamps and N.J.E. Wijngaards. $(RP)^2$: A KADS–solution for the office assignment problem. student report, July 1992.
12. W. Karbach, A. Voß, R. Schukey, and U. Drouwen. Model-K: Prototyping at the knowledge level. In *Proceedings Expert Systems–91*, pages 501–512, Avignon, France, 1991.
13. D. Landes, D. Fensel, and J. Angele. Formalizing and operationalizing a design task with KARL. In Treur and Wetter [17], pages 105–142.
14. Marc Linster. Linking modeling to make sense and modeling to implement systems in an operational environment. In Thomas Wetter et al., editors, *Current developments in knowledge acquisition: EKAW92, Lecture Notes in AI*, vol. 509, Springer-Verlag, 1992.
15. A. Th. Schreiber. Sisyphus'91: Modelling the office assignment domain. In M. Linster, editor, *Sisyphus'91: Models of Problem Solving*, Arbeitspapiere der GMD 663, chapter 11. GMD, Sankt Augustin, Germany, 1992.
16. M. E. Stickel. A prolog technology theorem prover: implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4(4):353–380, 1988.
17. J. Treur and Th. Wetter, editors. *Formal Specification of Complex Reasoning Systems*, Workshop Series. Ellis Horwood, 1993.
18. M. van 't Holt. Modelling of visual perception for a recognition task in noise analysis. Master's thesis, Dept. of Information Theory, Fac. of Electrical Engineering, Technical Univ. of Delft, August 1993. (in Dutch).
19. F. van Harmelen and J. R. Balder. $(ML)^2$: a formal language for KADS models of expertise. *Knowledge Acquisition*, 4(1), 1992.
20. I. van Langevelde, A. Philipsen, and J. Treur. An example reasoning task description. In Treur and Wetter [17], pages 7–14.
21. J. Walther, A. Voß, M. Linster, T. Hemman, H. Voß, and W. Karbach. MoMo. Technical Report Arbeitspapiere No. 658, GMD, Juni 1992.
22. T. Wetter. First-order logic foundation of the KADS conceptual model. In B. J. Wielinga, J. Boose, B. Gaines, G. Schreiber, and M. van Someren, editors, *Current trends in knowledge acquisition*, pages 356–375, Amsterdam, The Netherlands, May 1990. IOS Press.
23. B. J. Wielinga, A. Th. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–53, 1992.
24. R. Wols. Knowledge acquisition, modelling and formalisation for METEODES. Master's thesis, Dept. of Information Theory, Fac. of Electrical Engineering, Technical Univ. of Delft, July 1993. (in Dutch).