TheME: an Environment for Building Formal KADS-II Models of Expertise

John Balder and Hans Akkermans

Software Engineering & Research Department
Netherlands Energy Research Foundation ECN
P.O. Box 1, NL-1755 ZG Petten (NH), The Netherlands
e-mail: {balder,akkermans}@ecn.nl

COMMONKADS is a well-known methodology for the development of knowledge-based systems. In this methodology one constructs so-called models of expertise as a basis for the development. A new feature with respect to older versions of the KADS methodology is a formal version of these models, whereby models of expertise are expressed in a special logical language called (ML)^2 that is tailored to the needs of KBS methodology. Our paper presents TheME, an environment for constructing and manipulating these formal models of expertise. We consider the role of formal methods in conceptual knowledge modelling and briefly introduce the ingredients of the formal framework. Next, we discuss the rationale behind the TheME environment that supports the formalisation process. The functionality of the environment is described and illustrated in detail and experiences and future developments are discussed. TheME constitutes one of the tools of the COMMONKADS knowledge engineering workbench currently under construction.

Keywords: formal knowledge modelling

This paper has een awarded with the EUROLANG Award for the best paper presented at the 12th International Conference Avignon'92, Artificial Intelligence, Expert Systems, Natural Language. See Information on page 147.

1. Introduction: the Role of Formal Methods in Conceptual Modelling

KADS is a widely used methodology for KBS development. It may be said to be a de facto standard, especially in Europe. A fundamental idea underlying KADS is that KBS development is to be seen as a modelling activity. One of the most important models is the so-called model of expertise which models the knowledge used in reasoning in conceptual "real-world" terminology. To this end, KADS provides a general framework: it distinguishes various categories or "layers" of knowledge (domain, inference, task and strategic knowledge) and provides a vocabulary for further characterising different types and roles of knowledge within these categories and for the links between them. We will not further discuss here the KADS framework but refer to the extensive literature, e.g., (Wielinga and Breuker, 1986; Breuker and Wielinga, 1989; Wielinga et~al., 1989; Schreiber et~al., 1991).

In the Esprit-II project P5248 "KADS-II" currently

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 the KADS-II project are Cap Gemini Innovation (FR), Cap Gemini Logic (CH), Netherlands Energy Research Foundation ECN (NL), ENTEL SA (ES), IBM France (FR), Lloyd's Register (UK), Swedish Institute of Computer Science (SE), Siemens AG (DE), Touche Ross MC (UK), University of Amsterdam (NL) and Free University of Brussels (BE). This paper reflects the opinions of the authors and not necessarily those of the consortium.

underway, an extended and improved version of this methodology is being developed called the COMMONKADS methodology. One of the objectives of the project is to develop a state of the art conceptual modelling language based on a unification of exsiting approaches towards knowledge level modelling (Wielinga et~al., 1992). A new aspect of COMMONKADS is that it introduces the possibility to employ formal methods for the modelling of knowledge. We believe that there are several good reasons to do so (Balder and Akkermans, 1990,Akkermans et~al., 1992, because the conceptual model of expertise plays a crucial role in KBS development, but currently has major limitations in that it is informal and hence can be ambiguous and not sufficiently precise for further analysis and design.

We can summarise the role of (informal) conceptual models as follows:

- The conceptual model describes the target problem-solving behaviour of the artefact (or more generally the expertise of any intelligent agent), but does so in conceptual ("knowledge-level") terms of the domain expertise itself. It thus abstracts from KBS implementation aspects.
- Accordingly, the conceptual model plays a crucial role as a communication vehicle between the knowledge engineer and the domain experts and users, as it is expressed in a vocabulary familiar to them.
- It thus provides the conceptual basis for the KBS design — no more and no less. Of course, in some cases conceptual features straightforwardly map onto computational methods. However, usually this is not the case: it is not in conventional software and it would therefore be highly unlikely to be the case for knowledge-based systems. Consequently, additional activities are necessary before KBS design (*i.e.*, decision making with respect to computational methods to be used in the KBS) and implementation can meaningfully commence.

It is here that formal methods come into play. The general role of formalisation is *to enhance the process of conceptual modelling* in various ways:

- Formal methods yield additional means for making conceptual models *more precise and less ambiguous*. We note that it is possible and useful to formalise only parts of a conceptual model, notably those parts that are unclear or that are critical to the functioning of the KBS, for example due to external requirements. We also note that the price one usually has to pay for increased detail and precision is greater complexity of the model.
- A (partly) formalised conceptual model provides more and better handles for *inspecting its consistency and correctness*.
- Hence, it is important as part of the technical *documentation* of a KBS and valuable for *review actions* as undertaken in quality assurance systems.
- Upon mechanising the used formal language by introducing theorem-proving capabilities, it becomes possible to carry out *simulations of the problem-solving behaviour* of a model. Simulation helps to *test the adequacy* of a conceptual model and is a widely used method in engineering design in general.
- As a result of the above, formal modelling provides useful *feedback to conceptual modelling* and generates a *more solid KBS design basis*.

As part of the COMMONKADS methodology, a formal language has been developed for knowledge modelling. It is based on a variety of standard logic elements but in a way that matches the conceptual COMMONKADS framework. This language is called $(ML)^2$ and has been described in (Akkermans et. al., 1992; van Harmelen and Balder, 1992 and various Esprit reports available from the authors; a brief sketch is given in the next section. The present paper focusses on TheME, an environment that supports formal knowledge modelling with $(ML)^2$ and is part of the COMMONKADS knowledge engineering workbench currently under construction. The design requirements and rationale of TheME are described in Sec. 3. In Sec. 4 its functionality is explained and illustrated in some detail. Finally, we discuss experiences from formal knowledge modelling with TheME and indicate directions for further work.

2 The $(ML)^2$ Language

The formal modelling language $(ML)^2$ for KADS conceptual models has been described elsewhere in several recent papers, see (Akkermans et~al., 1992; van Harmelen et~al., 1990; van Harmelen et~al.

1990; van Harmelen and Balder, 1992. The formal modelling language has been designed such that the various knowledge types distinguished in the conceptual modelling language of COMMONKADS can be expressed by means of different formal constructs. An $(ML)^2$ description is basically a structure of logical theories. The choice of both the logic and the structure has been derived from the nature of the various elements ocurring in the COMMONKADS framework, as briefly indicated below.

Layered framework. COMMONKADS distinguishes different categories or "layers" of knowledge: the domain, inference, task and strategy layers. This layered structure — a prominent feature in the KADS methodology for KBS development — recurs in $(ML)^2$, each layer consisting of a set of logical modules of a certain structure. Thus, in our formal $(ML)^2$ models the various layers occur as separate components of the knowledge model, in a one-to-one relation to the layers of the conceptual model of expertise. This is also implemented in TheME. Some examples are found further on in this paper.

Domain layer. Domain knowledge is modelled in $(ML)^2$ as logical theories. Each theory consists of a signature and a set of axioms. The logical language is usually first-order order-sorted predicate calculus (order-sorted FOPC), but it can be extended to include, for example, modal operators. Domain knowledge can be formally structured, since $(ML)^2$ allows for the modular combination of separate subtheories (for example corresponding to separate domain concepts in the conceptual model of expertise). This modular structuring and recombination is done by means of simple meta-theoretic operators, such as the *import* operator which generates the union of two theories. In this respect $(ML)^2$ is the KBS equivalent of algebraic specification languages for conventional software (Bergstra et~al., 1990).

Inference layer. Also the knowledge at the inference layer is modelled in terms of a modular order-sorted FOPC. The modular structure is such that the well-known inference structure diagrams in KADS and in $(ML)^2$ are identical. Primitive inference actions and dynamic knowledge roles[1] are thus visible in $(ML)^2$ as separate components, with their own specific format: dynamic knowledge roles are modules consisting of signature only (defining the language used at the inference layer), while primitive inference actions are theory modules characterised by a single leading predicate identical to the name occurring in the conceptual inference diagram.

Task layer. In COMMONKADS the task layer contains control and procedural knowledge which cannot be naturally modelled in terms of FOPC. Therefore we use at the task layer a different logical language, viz. quantified dynamic logic (QDL, a multimodal logic) that is able to speak about the execution of (here: inference) actions and has built-in notions of sequence, iteration and selection. This QDL is a superset of the language used at the inference layer such that the task structures of KADS can be written down formally. Thus, it is possible to formally express procedural knowledge and have a declarative semantics.

Links between layers. In $(ML)^2$ the relation between the domain and inference layer is a meta-relation. This is a natural solution since the domain layer describes the content of expert knowledge, while the inference layer speaks *about* the inferential use of the domain knowledge. The link between these layers is specified in $(ML)^2$ by a so-called *lift*definition. It gives domain theories a name at the inference layer by establishing a naming relation. This mapping is achieved via sets of rewrite rules. A particular feature is that we do not use the conventional quotation or structural-descriptive names, but instead employ a user-definable naming. This type of naming relation makes it possible to give *meaningful* names to domain knowledge elements that express the *role* that they play in the inference process. In addition to meaningful naming, $(ML)^2$ employs so-called reflection rules in order to establish object-meta-knowledge relationships. For a further technical discussion the reader is referred to the above-cited papers on $(ML)^2$. Interestingly, the link between the inference and task layers in $(ML)^2$ is not a meta-relation, but an embedding relation: the FOPC language used at the inference layer is embedded in the QDL language of the task layer. Thus, the control knowledge of the task layer is to be seen as an *addition* to the inference knowledge.

In brief, our formal modelling language has a multi-level and multi-language structure (hence the name

1. Primitive inference actions and dynamic knowledge roles are the COMMONKADS equivalents, as defined by the Esprit-project "KADS-II", for the former knowledge sources and metaclasses. In the remainder of this paper we will conform to the KADS-II terminology, but users of the earlier versions of KADS may simply replace this with knowledge sources and metaclasses without being led into misinterpretations.}

$(ML)^2$). As such, it has some logical features in common with systems like Socrates (Jackson et~al., 1989), REFLOG (Lavrac and Vassilev, 1989) and FOL (Weyhrauch, 1980), but in contrast to these $(ML)^2$ is tailored to the COMMONKADS KBS methodology.

3 Rationale for TheME

When a knowledge engineer is constructing a formal knowledge model, we can distinguish two main activities. One is the actual design of the model, the other one is a task that is more administrative in nature, namely, taking care that the formal knowledge model is (stays) syntactically correct and well-formed. In practice, the second activity can be very complex and time-consuming. Most of the latter activity is well defined, however, and can easily be performed by a machine. This is the rationale behind TheME.

TheME is not a tool in the sense that it can perform a predefined task. Rather it can be seen as a very accurate co-worker that keeps track of all the nitty-gritty details of the formal model while the user (knowledge engineer) can concentrate on the design component of the task. (We actually observed this phenomenon when two people work together in constructing a formal model. Most of the time one is doing the design while the other keeps track of the formal details.) This "co-worker" only speaks up when the integrity of the model is violated. But it can be consulted if the user wants to know certain details about the model. Besides this, another important feature is to offer only that functionality to the user that is appropriate in a certain context.

An idea that very much influenced the architecture of TheME was that of so-called *self-contained contexts*. When the user carries out a certain task he switches to the context most appropriate for that task. This context must contain all necessary information (and functionality) to perform the task. Only in this way can the user do the things he wants to, without having to go to other contexts where the functionality s/he wants is available. Thus, user actions are minimalized.

3.1 The Process of Formal Modelling

The functionality of TheME was deduced from observing actual model development and partly it was done by introspection. This resulted in the following observations.

- Humans use several levels of abstraction and frequently shift between these levels. Sometimes several levels are used at the same time.
- Items in the model are viewed externally and internally, i.e. sometimes items are viewed as atomic entities with respect to their role in a larger structure and sometimes the internal structure of an item is inspected.
- Different cross sections of the knowledge model are needed. Sometimes these are cross sections like in the COMMONKADS methodology, i.e. layers. Sometimes these are cross sections that show items selected according to certain properties like: all knowledge modules that have a certain component in their signature.
- At higher levels of abstraction, that is at levels above the level of logic statements, *graphical* representations are the basic means to represent information.
- There is a constant need to present the same information in different ways. For example, graphical as well as textual. In situations where building the formal model is done without a supporting environment it can be observed that a lot of time is needed to keep these different representations of the knowledge consistent. Often, errors occur because different representations are assumed to be consistent while in fact they are not.

All the above mentioned observations together with experiences with a first prototype of this environment called StrucTOOL (Balder and Akkermans, 1990) led to general requirements for a supporting environment for building formal knowledge models. Summarising we can formulate the requirements as follows.

- Visualisation of the model and user interaction through this visual metaphor.
- Support for different points of view.
- Different levels of aggregation, e.g. *KADS layer* level, *module* level, *axiom* level.
- Continuous checking of formal consistency of the model.
- *Where-what-how* support.
- Restrict formal constructs to the COMMONKADS models.

TheME was built in an evolutionary way starting from these requirements. Further requirements were defined while developing the environment. Most of the time these were functionalities not yet thought about because they were too complex or too time-consuming for a human to perform. Much in the line of thought: ... *but if we do have all this information on line we could easily do* .... In this category one finds for example:

- Tracing the origin of syntax elements, i.e. finding out where, i.e. in which knowledge module, they were originally defined.
- Finding out where in the knowledge model a certain item is used and with what meaning.
- Finding out declarations that are redundant, i.e. never used

In the Section 4 we will describe the functionality of TheME as developed from these requirements. Before that we will discuss in more detail the concept of well-formedness.

3.2 Well-formedness of a formal model

TheME helps to guarantee the mathematical well-formedness of the model. To accomplish this the system starts with the empty model ($\emptyset$) which of course has all the well-formedness properties we want. Thereafter the system checks all additions to the model, whether from file or from user interaction. Modifications to a module are effected by *adding* this modified module to the model, replacing the existing module. Additions are only authorised if the new model is still consistent[2]

Below we will describe what kind of checks are necessary to keep a model well-formed:

Syntax. The first thing to check is the syntax of the description of the knowledge modules. A complete syntax description for $(ML)^2$ has been defined in (van Harmelen and Balder, 1992).

Declaration. The signature of a knowledge module defines the language in which axioms from the theory can be expressed. Therefore the elements of an axiom have to be defined (in the signature) before they can be used (in the axioms). Furthermore, within a signature, there also is a dependency: sorts used in the declaration of constants and predicate- and function schemas must have been declared beforehand.

Context. In the present mathematical framework we distinguish two principal ingredients: (logical) theories which represent a chunk of knowledge and lift-definitions which define a meta-level naming for object-level items. Although there is some overlap in the type of elements that can occur in both elements, there are also differences. The context, theory or lift-definition, determines which constructs are allowed.

Typing. The formalism used to describe the logical theories is order-sorted logic, which implies that all constants (variables) are of a certain sort. Likewise axioms are instances of predicate schemata which describe the sort of the elements of an axiom. There must be consistency between the sort of elements in a predicate and the sorts defined in the schema of the predicate.

4 The actual environment

In this section we will describe the functionality of the actual environment. First we give some general properties and after that we go top-down through the environment, starting with the *model*-level, through the *KADS-layer*-level down to the *module*-level. We present some examples to illustrate the considerations mentioned in the previous section.

In constructing TheME we tried to establish a common "look-and-feel" for all parts of the system. This "look-and-feel" is governed by the following simple rules:

- All visible items that have an internal structure can be opened to show this structure.
- All functionality can be activated by means op pop-up menus attached to visible items. These pop-up menus are dynamic and context dependent, i.e. depending on the status of other parts of TheME.

A very natural way for humans to present structural information is to use pencil and paper to draw up a graph. Therefore, at the layer level, TheME offers an interface that has this pen-and-paper appearance to it. Much in the style of visual programming the user

2. It should be noted that we only talk about *mathematical* properties of the model here. It is very well possible to produce a well-formed model that it completely nonsensical from a *conceptual* point of view. Accessing the semantics of a model clearly is still only within the competence of the human knowledge engineer.

can add nodes and edges to a graph. The nodes represent the knowledge modules, i.e. theories or lift-definitions. The edges represent logical operations between modules defined in the logical framework, i.e. *import, use* or *from*. The user can manipulate the graph, while the system keeps track of the implications for the modules. That means, for instance, that, if an edge representing an *import* is added, the corresponding logical statements in the modules are generated. Of course this mechanism also works the other way around: modifications to the structure through the text of modules are reflected in the visual representation. All modules can be moved by dragging them around with the mouse. In this way working on a layer very much resembles the activity of sketching the layer hierarchy on a piece of paper.

4.1 Functionality at the model level

The top-level of TheME presents the user with a view on a knowledge model that brings about the KADS layer structure. This is shown in Figure 1. The label identifies the model under construction. The popup menu that belongs to this window shows that one can select each layer and open a browser on it.

The menu also provides a utility to make snapshots of the screen[3]. Also a facility to file in (out) a complete model is provided here, as well as a possibility to rename the model.

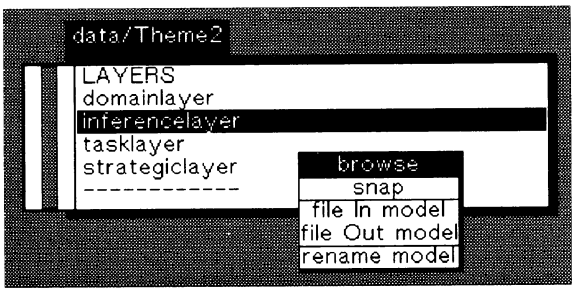

Figure 1: Topview of TheME.

4.2 Functionality at the layer level

Figure 2 shows the situation after a browser on the domain layer has been opened. Four subviews can be distinguished in this browser.

- The lower part shows a graphical representation of the modules and their import relations. This subview provides a visual-programming style interface. A menu attached to the subview enables:
  - adding theories;
  - several functions to lay-out the figure;
  - a function to detect unused signature elements in this layer.
- In the upper-left corner an alphabetic list of all

3. This facility was used to prepare the illustrations in this document

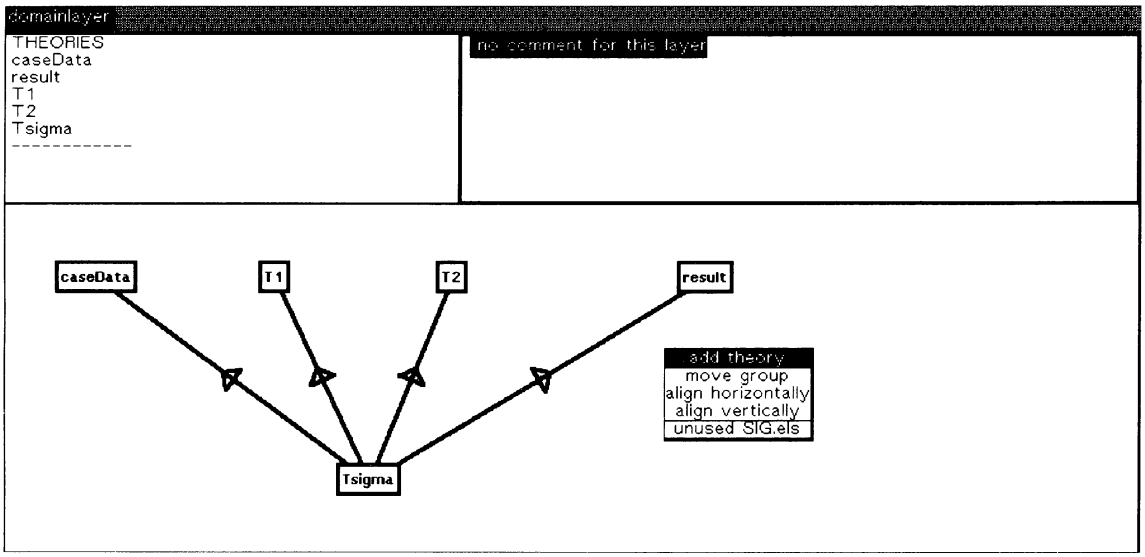

Figure 2: Browser on the domain layer of a knowledge model

theories is shown. This separate list is useful for finding a certain theory. Selecting a theory in this list highlights it in the graphical subview. The functionality here is adding theories or browsing a selected one.

- The upper-right corner contains a text subview showing comment for this layer. The popup menu contains functionality related to the layer such as, for instance:
  - filing in (out) a layer
  - printing the description of the layer[4]
  - producing LaTeX output for the layer.

  The rationale for dividing functionality between this subwindow and the graphical subwindow is: functionality that is internally directed, i.e. towards modifying the model, is located in the graphical subwindow. Externally directed functionality is located in this subwindow.

The last functionality to describe at the *KADS-layer*-level is that of theories and liftdefinitions in the graphical view (see Figure 3), i.e. modules as part of a layer. All modules provide functionality for:

- browsing, a browser can be opened on the module;
- connecting, the user can, by means of a rubber-band animation, connect modules. TheME generates the appropriate formal statements;
- removing, modules can be removed, but only in those cases where no other modules depend on them;
- renaming, if the name of a module is changed all references to it automatically show the new name.

4. This description has the usual syntax.

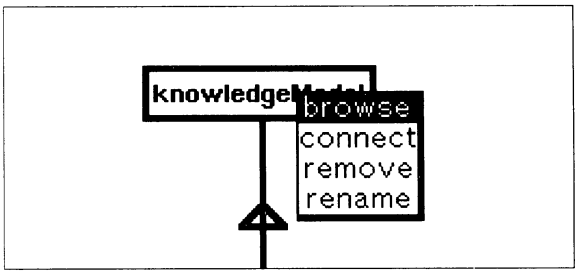

Figure 3: Menu for an item in a layer.

The browser described above views the domain layer as a set of knowledge modules. This view is available for all layers. For the inference layer, however, TheME offers also a KADS model point of view. I.e. a view in terms of dynamic knowledge roles and primitive inference actions. It should be noted that both views operate on the same internal model. Figure 4 shows an inference layer under construction. Ellipsis denote primitive inference actions and rectangles denote dynamic knowledge roles[5]. The menu

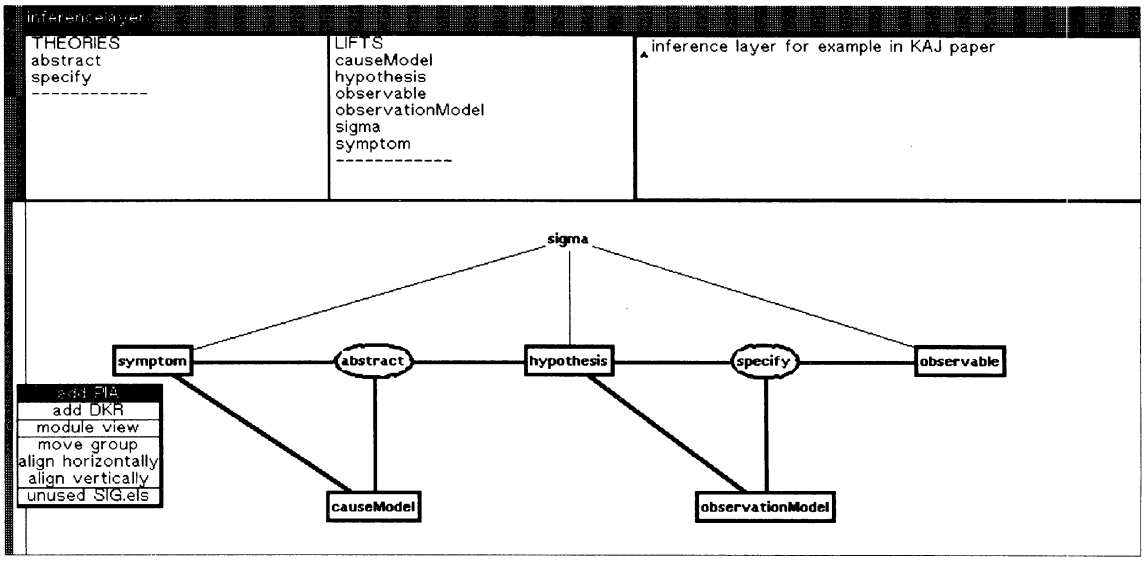

Figure 4: KADS view on an inference layer.

in the graphical view has been augmented with an option to switch between the *formal* and the *KADS* view. In the *KADS* mode addition of elements is in terms of dynamic knowledge roles and primitive inference actions. The normal KADS restrictions apply for connections, e.g. no connection is possible between two primitive inference actions.

4.3 Functionality at the module level

The modules in a layer contain the logical statements that make up the knowledge model. The internal structure of modules can be accessed by opening a browser upon them. An example of such a browser on a theory is shown in Figure 5. Notice that the layout (indentation, bold keywords) enhances the readability.

Here we see 10 subviews, the bottom one shows the textual representation of the theory. The upper ones show lists of selected parts of the signature of the theory, *e.g.* (from left to right) lists of the sorts, constants, variables, predicate and function names, imported theories, used lifts, theories that import this one and finally lifts that lift from this theory. The last two subviews are not part of the formal syntax but are provided because they are very useful to access the hierachical structure of the layers. The lower subview contains only local definitions. The upper subviews, on the other hand, are expanded with respect to the import and use relations. That means that they show also signature elements defined in the modules that this module imports or uses.

The upper subviews can provide still more information. The constant and variable subview depend on the selection in the sorts view. If no sort is selected, all constants c.q. variables are shown. If a sort is selected, only those constants c.q. variables are shown that are of the corresponding sort. The predicate and function name subviews depend in a similar way: they show all items, or only those items that contain the selected sort in their schema. In this way the use of sorts can easily be overseen. Menus in the upper subviews provide functionality for adding signature

5. Note that there is one module (*sigma*) that is neither dynamic knowledge role nor primitive inference action. This module is necessary because of properties of the formal framework, it has no KADS semantics, however.

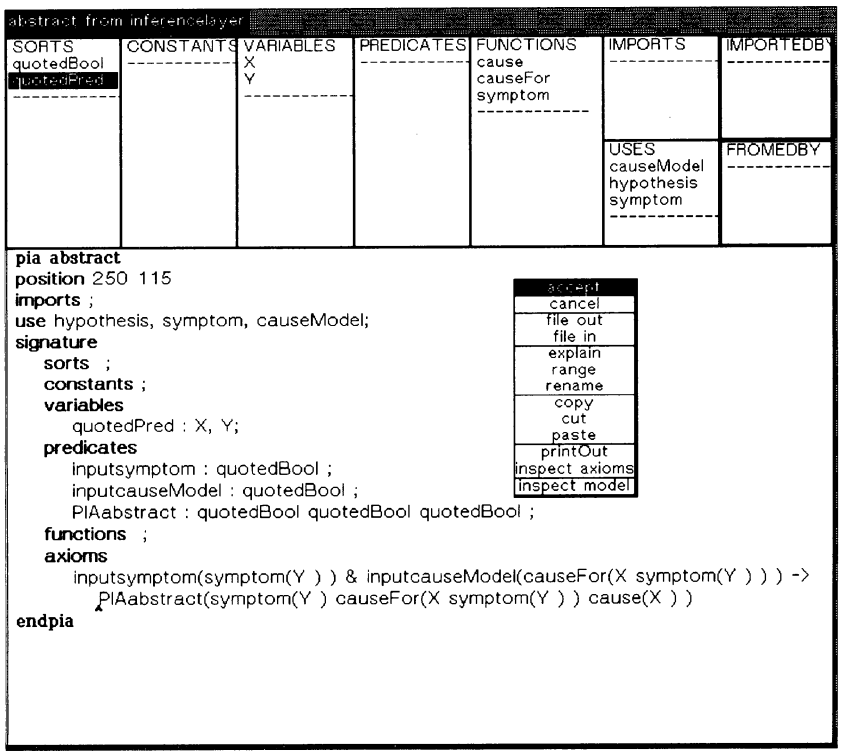

Figure 5: Browser on a logical theory.

elements. The imports subview for instance provides a menu with theories to import. This menu is created dynamically in such a way that circular imports are prevented. (This is an example where functionality is duplicated. The same could have been achieved by switching to the layer browser, selecting the module and connecting it it.) Also one can request information about signature elements. Two kinds of information are available: about the origin of a signature element, i.e. where it is defined, and about its range, i.e. where it is used.

The lower subview which contains the textual specification of the module offers full editor functionality to modify the specification. Besides that it has the following functionality (see menu in Figure 5):

- **accept** i.e. accept the modified text as part of the knowledge model, **cancel** restores the last accepted version of the module;
- **explain** displayes what a token is, e.g. keyword, sort, predicate etc.;
- **range** displays information about the use of a certain token the user has selected;
- **rename** allows the user to rename all items in a description (with the exception of keywords of course), all other occurrences of this item in the model will automatically show the new name;
- **printOut** makes a hardcopy of the module;
- **inspect axioms** displays a hierarchy of axioms, where the origin of axioms is still recognizable.

Liftdefinitions have a browser analogue to a theory browser. The content is somewhat different, however, because a liftdefinition has different elements in its definition.

Interactive Module Parser. TheME offers two ways to modify a knowledge model which we call *open* and *guided*. The *open* way for modification accepts a completely new description of a module. There are two ways to add new modules to a knowledge model: from file or from a module browser (i.e. by **accept**ing the modified text in the lower subview). Both ways are treated in the same way; the description is parsed and checked for well-formedness as described in section 3.2. If no inconsistency is detected the new module is accepted. If there is, the system restores the old model. The modification the user made is not lost, however, but is kept in the browser, together with an error notification. This enables the user to repair the error. Within this environment TheME provides the following support.

Positioned errors. In order to give maximal feedback it is helpful to display the error at the position where it occurs. Also, by providing such a functionality, the user can easily switch from observing (the error) to repairing.

Spelling correction. A common error is the misspelling of names in a description. TheME supports a prototype version of spelling correction, i.e. if a certain name cannot be evaluated in the context of a module, the name is matched against all names of the same type (i.e. constant, sort or predicate-name). The user then is presented with a menu of names that resemble the misspelled one, from which he can select the right one.

Automatic declarations. In case of forgotten declarations the system can often deduce this declaration from other information in the module. If this is the case, TheME offers the user the possibility to automatically generate the necessary text.

Figure 6 shows an example of a parsing error.

Sometimes when TheME signals an error it is evident how the error can be repaired. This is especially true in case of missing declarations of constants or variables. In this case TheME offers the possibility for automatic generation of the appropriate specification, as shown in figure 7.

```
use description model
    sorts ;
    constants ;
    variables
        item anItem <- separator : forgotten ;
        description : aText itemDescription ;
    predicates
```

Figure 6: Parser error notification.

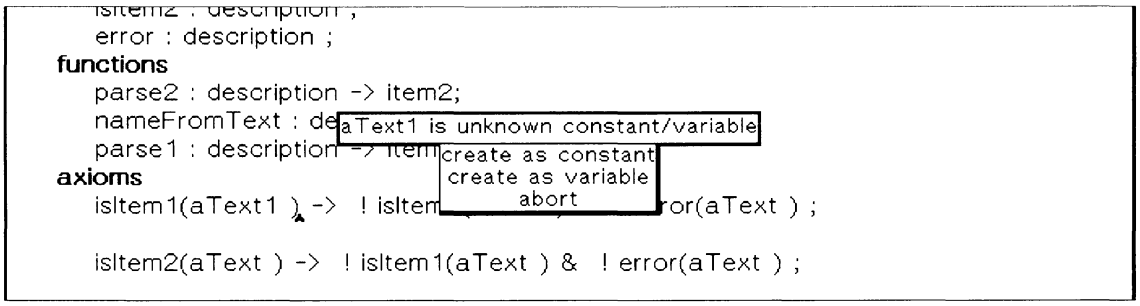

Figure 7: Suggestion for repair of a parsing error.

If only a small part of a description is modified, e.g. when only adding a sort, it is not necessary to parse the whole module. Therefore TheME offers also *guided* ways for modification which are more restricted and which thus require more simple checking. These modifications can be done in the upper subviews that show selected parts of a module. Interaction is predefined and offered to the user by means of menus.

5 Discussion and Conclusion

The present implementation of TheME has been tested on several real-life applications, see for instance (Balder 1991, Balder et~al., 1990; Reinders, 1990). Concerning the functionality the present implementation seems rather complete. The syntax that can be handled is almost the complete syntax defined for $(ML)^2$. Our practical experience with using $(ML)^2$ on real-life models of expertise is encouraging. We gain the expected advantages of formal representations over informal ones (such as removal of ambiguity, inspectability). On the other hand, constructing a formal model is quite a complex activity. It is here that TheME as an engineering tool has demonstrated to be not only useful, but even indispensable. First, TheME relieves the knowledge engineer of a major administrative burden, by propagating changes throughout the model, checking their consistency and keeping track of the many items in the knowledge model. It is the experience that doing this by hand is virtually impossible for all but the most simple models of expertise. Second, TheME enforces precision in modelling since it does not accept any change that is incompatible with the modelling language. So, sloppy modelling is impossible with TheME. One experiences that this is an important feature upon feeding (as has been done by us) models into TheME that were previously developed only manually.

Another interesting experience from working with TheME concerns guidelines for formal knowledge modelling. How does one make a formal model of a given KBS application domain? This question regarding the "process model" behind $(ML)^2$ formalisation can be answered on the basis of our current application experiences with TheME. The starting point for developing a formal model is the existence of a decent informal COMMONKADS conceptual model (and of course a satisfactory knowledge of $(ML)^2$ by the knowledge engineer). The next step is to start with the inference layer, since the inference diagram structure of the conceptual model immediately maps onto the $(ML)^2$ module structure at the inference layer of the formal model. Next, the lift definitions (i.e. the links between the domain and inference layers) have to be worked out. Making the lift definitions includes the formalisation of the dynamic knowledge roles (called metaclasses in the older KADS framework). The main new modelling decisions here concern the (re-)structuring of the domain knowledge so as to obtain a natural connection between the layers. This is the hard part: the formalisation of the domain and lift knowledge itself appears to be reasonably straightforward and is done in parallel. Subsequently, the primitive inference actions are formally defined (called knowledge sources in the older KADS framework). The last step is to add the control knowledge at the task layer. The handles for this are the conceptual model task layer plus the formats and restrictions provided by $(ML)^2$.

In general, formal modelling is done *iteratively* in interaction with conceptual modelling. Sometimes it appears that the initial conceptual model is incomplete or inconsistent. Even if there is no problem in this respect, formalisation of the primitive inference actions in particular appears to necessitate additional

refinements of the available knowledge. Hence, formal knowledge modelling helps to point out where additional knowledge acquisition is necessary.

We note that there exist several other attempts to pin down more clearly the KADS conceptual models of expertise. Roughly speaking, this can be done either by formalising them based on mathematics and logic (which is the focus of the present work) or by operationalising (mechanise) them in terms of a KADS-specific computer language. For instance, the MODEL-K approach from (Karbach et~al., 1991) aims at operationalising a model of expertise, but it does not provide a declarative representation. So, that work focusses on mechanisation rather than formalisation. Mixed approaches are found in (Wetter, 1990 and KARL (Angele et~al., 1991; Fensel et~al., 1991). We would argue however that the formalism presented in (Wetter, 1990) contains a number of constructions that can only be understood through a procedural (i.e. non-declarative) semantics. The KARL language more closely resembles $(ML)^2$, but is much more restricted and less expressive. For a further discussion see (van Harmelen and Balder, 1992). One may say that there is a tension between formalisation and (efficient) mechanisation, and that the various approaches provide a different trade-off between the two.

It is however our intention to couple TheME, as an environment to prepare formal knowledge models, with an interpreter for $(ML)^2$. In this way formal $(ML)^2$ models also become mechanisable, and it becomes possible to simulate the problem-solving *behaviour* of a model of expertise. We point out however that the combination of a *formal model + interpreter* is to be seen as a simulation environment but it does not equal a real implementation. This type of mechanisation will enable to test the adequacy of the conceptual model by simulation of its behaviour (a modern design technique in itself in many branches of engineering), but design issues like computational efficiency are not considered in this stage. On the other hand, it is expected that a model of expertise, when it has proved to be adequate via formalisation and simulation, can be turned into an operational KBS through structure-preserving or transformational design techniques. The intended coupling of the current implementation of TheME with an interpreter called $Si(ML)^2$ has recently started and preliminary experiments have already been carried out (ten Teije et~al., 1991).

In sum, the COMMONKADS approach to knowledge engineering conforms, as it in our opinion should, to recent trends in engineering in general: it attempts to turn KBS development into a branch of scientific engineering. The formal methods enterprise is a necessary and normal part of this, as it is in all branches of advanced engineering. The experiences with TheME shed some light on various issues related to future computer-aided knowledge engineering, with modelling and simulation in a pivotal role.

Acknowledgement. We are indebted to Frank van Harmelen (University of Amsterdam) who has been central in the development of $(ML)^2$ and also greatly contributed to the modelling experiences on which TheME is based.

References

Akkermans, H., van Harmelen, F., Schreiber, G., and Wielinga, B. (1992). A formalisation of knowledge-level models for knowledge acquistion. *International Journal of Intelligent Systems*. forthcoming.

Angele, J., Fensel, D., Landes, D., and Studer, R. (1991). Sisyphus - no problem with KARL. In *Proceedings of the European Knowledge Acquisition Workshop EKAW'91, (Sisyphus Working Papers Part 2)*, Strathclyde.

Balder, J. (1991). A formalized kads model of TheME. ESPRIT Project P5248 KADS-II, KADS-II/T1.2/TR/ECN/003, Netherlands Energy Research Foundation ECN. in preparation.

Balder, J. and Akkermans, J. (1990). StrucTool: Supporting formal specifications of knowledge-level models. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., editors, *Current trends in knowledge acquisition*, pages 60-77, Amsterdam. IOS Press.

Balder, J., van Harmelen, F., and Schreiber, G. (1990). Example formal representations in the synthesis domain. ESPRIT Basic Research Action P3178 REFLECT, Intermediate Report IR.1.2 RFL/UvA/I.1/2, University of Amsterdam.

Bergstra, J., Heering, J., and Klint, P. (1990). Module algebra. *Journal of the ACM*, 37(2):335-372.

Breuker, J. A. and Wielinga, B. J. (1989). Model Driven Knowledge Acquisition. In Guida, P. and Tasso, G., editors, {*Topics in the Design of Expert Systems*, pages 265-296, Amsterdam. North Holland.

Fensel, D., Angele, J., and Landes, D. (1991). Knowledge representation and acquisition language ({KARL). In *Proceedings 11th International workshop on expert systems and their applications (Volume: Tools and Techniques)*, Avignon.

Jackson, P., Reichgelt, H., and van Harmelen, F. (1989). *Logic-Based Knowledge Representation.* The MIT Press, Cambridge, MA.

Karbach, W., Voss, A., Schukey, R., and Drouwen, U. (1991). Model-K: Prototyping at the knowledge level. In *Proceedings Expert Systems-91,* Avignon, France, pages 501-512.

Lavrac, N. and Vassilev, H. (1989). Meta-level architecture of a second-generation knowledge acquisition system. In Morik, K., editor, *Proceedings EWSL-89,* pages 99-109, London. Pitman.

Reinders, M. (1990). On the correspondence between abductive diagnosis and qualitative reasoning. ESPRIT Basic Research Action P3178 REFLECT, Technical Report RFL/ECN/II.1/1, Netherlands Energy Research Foundation ECN.

Schreiber, G., Akkermans, H., and Wielinga, B. (1991). On problems with the knowledge level perspective. In Steels, L. and Smith, B., editors, *AISB-91: Artificial Intelligence and Simulation of behaviour,* pages 208-221, London. Springer Verlag. Also in: *Proceedings Banff-90 Knowledge Acquisition Workshop,* J.H. Boose and B.R. Gaines (editors), SRDG Publications, University of Calgary, pages 30-1 - 30-14.

ten Teije, A., van Harmelen, F., and Reinders, M. (1991). Si$(ML)^2$: a prototype interpreter for a subset of $(ML)^2$. ESPRIT Project P5248 KADS-II, KADS-II/T1.2/TR/UvA/005, University of Amsterdam, Netherlands Energy Research Foundation ECN.

van Harmelen, F., Akkermans, H., Balder, J., Schreiber, G., and Wielinga, B. (1990). Formal specifications of knowledge models. ESPRIT Basic Research Action P3178 REFLECT, Technical Report RFL/ECN/I.4/1, Netherlands Energy Research Foundation ECN.

van Harmelen, F. and Balder, J. (1992). $(ML)^2$: a formal language for KADS conceptual models. *Knowledge Acquisition.* to appear.

van Harmelen, F., Balder, J., Aben, M., and Akkermans, J. (1991). $(ML)^2$: A formal language for KADS models of expertise. ESPRIT Project P5248 KADS-II, Kads-II/T1.2/TR/ECN/006/1.1, Netherlands Energy Research Foundation ECN,. deliverable D1.2.1.

Wetter, T. (1990). First-order logic foundation of the KADS conceptual model. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., editors, *Current trends in knowledge acquisition,* pages 356-375, Amsterdam. IOS Press.

Weyhrauch, R. (1980). Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence,* 13. Also in: *Readings in Artificial Intelligence,* Webber, B.L. and Nilsson, N.J. (eds.), Tioga publishing, Palo Alto, CA, 1981, pp. 173-191. Also in: *Readings in Knowledge Representation,* Brachman, R.J. and Levesque, H.J. (eds.), Morgan Kaufman, California, 1985, pp. 309-328.

Wielinga, B., Akkermans, H., Schreiber, G., and Balder, J. (1989). A knowledge acquisition perspective on knowledge-level models. In Boose, J. H. and Gaines, B. R., editors, *Proceedings Knowledge Acquisition Workshop KAW-89, Banff,* pages 36-1 -- 36-22, University of Calgary. SRDG Publications.

Wielinga, B. and Breuker, J. (1986). Models of expertise. In *Proceedings ECAI-86,* pages 306--318.

Wielinga, B., van de Velde, W., Schreiber, G., and Akkermans, H. (1992). Towards a Unification of Knowledge Modelling Approaches. ESPRIT Project P5248 KADS-II, KADS-II/T1.1/UvA/RR/004/3.0, University of Amsterdam & Free University of Brussels & Netherlands Energy Research Foundation ECN. Draft Deliverable D1.1, version 3.

The EUROLANG Award

The aim of the EUROLANG project (European Languages) is to develop a second-generation machine-aided Translation System (MT). The SITE (France) and SIEMENS-NIXDORF (Germany) groups and several subcontractors among which Krupp, Matra Marconi Space, Cap Gemini and Rank Xerox, will pool their resources in this project. Academic subcontractors are the 5 Euretra Groups, LADI (Paris) and Universita Autonoma di Barcelona.

To express the confidence of SITE and SIEMENS-NIXDORF in the future of the European Research, they have funded the "EUROLANG Award" for the best paper presented at the 12th Avignon conference.

This award (FRF 20.000) was presented to John Balder and Hans Akkermans for the paper presented above. For more information please address: SITE, Dominique Maret, 12 rue de Reims, 94701 Maisons-Alfort Cedex, France.
Tel: +33 1 45 13 05 00, fax: ... 05 59