Comparing Formal Specification Languages for Complex Reasoning Systems

Frank van Harmelen*
University of Amsterdam
Department of Social Science Informatics
Roetersstraat 15, NL–1018 WB Amsterdam, The Netherlands
frankh@swi.psy.uva.nl

Ramon López de Mántaras†
IIIA, CEAB–CSIC
Camí de Santa Bàrbara
17300 Blanes, Girona, Spain
mantaras@ceab.es

Jacek Malec‡
Department of Computer and Information Sciences
Linköping University
S–581 83 Linköping, Sweden
jam@ida.liu.se

Jan Treur
Free University Amsterdam
Department of Mathematics and Computer Science
Artificial Intelligence Group
De Boelelaan 1081a, NL–1081 HV Amsterdam, The Netherlands
treur@cs.vu.nl

*Partially funded by the ESPRIT Programme of the of the European Communities as project number 5248 (KADS-II).
†Partially supported by the ESPRIT-BRA DRUMS-II (6156).
‡Supported by the Center for Industrial Information Technology (CENIIT).

Abstract

This paper presents a comparison between eight specification languages discussed during the Workshop on Formal Specification Techniques for Complex Reasoning Systems held in Vienna during the ECAI'92 conference. The languages as discussed here possess many important common characteristics, but also differ substantially. The analysis discussed here departs from looking at the purposes of the presented languages (Section 1). The comparison in Section 2 focuses on the way of dealing with heuristic knowledge in the specification of the common example task. In Section 3 some differences between the languages are discussed:

- expressive power;
- the way of specifying control knowledge;
- layering of the system architecture.

In Section 4 we identify where already a certain consensus can be found; points are discussed that are in common for most of the languages:

- modularity;
- local declarativeness;
- multi-level view on a specification;
- distinct specification of static and dynamic aspects;
- separation of generic and domain-specific parts of a specification;
- object-meta distinctions in specifications;
- the level of specification.

Moreover, in Section 4.2 the major open problems are presented.

Introduction

This paper presents a comparison between eight specification languages discussed during the Workshop on Formal Specification Techniques for Complex Reasoning Systems held in Vienna during the ECAI'92 conference (cf. this volume [3, 4, 6, 7, 9, 10, 11, 14]). The languages as discussed here possess many important common characteristics, but also differ substantially. The analysis discussed here departs from looking at the purposes of the presented languages (Section 1). The comparison in Section 2 focuses on the way of dealing with heuristic knowledge in the specification of the common example task (cf. [8]). In Section 3 some differences between the languages are discussed:

- expressive power;
- the way of specifying control knowledge;
- layering of the system architecture.

In Section 4 we identify where already a certain consensus can be found; points are discussed that are in common for most of the languages:

- modularity;
- local declarativeness;
- multi-level view on a specification;
- distinct specification of static and dynamic aspects;
- separation of generic and domain-specific parts of a specification;
- object-meta distinctions in specifications;
- the level of specification.

Moreover, in Section 4.2 the major open problems that are felt are presented.

In this Introduction we give a summary of the list of criteria for comparison that were collected before the workshop. Table 1 is based on the author's own opinions about their languages with respect to these criteria. The list consisted of the following items:

1. Expressive power;
2. Transparency;
3. Knowledge level specification or not;
4. Declarativeness;

5. Adequacy to specify dynamic aspects of reasoning patterns;
6. Possibility to specify multi-level (e.g., global vs. local) architectures adequately;
7. Possibility to distinguish between generic and non-generic parts of a specification adequately;
8. Possibility to specify reflective reasoning patterns adequately;
9. Adequacy to specify non-classical reasoning patterns (reasoning with uncertainty, non-monotonic reasoning, etc.);
10. Possibility to specify integrated systems (with conventional components) adequately;
11. Executability; Possibility to generate prototype implementations automatically;
12. Availability of a clear semantics;
13. Availability of a solid formal semantics.

An important issue related to this list is the interpretation of "a language offers a possibility of doing something" (notice that the word "possibility" is used extensively in the list). The weakest interpretation is that the language does not exclude doing something. But of course a stronger interpretation is more interesting: does the language actually offer constructs that are easy to use and does it encourage (or force) the user to exploit these constructs. During the workshop one more criterion has been added to the list, namely the question of support for incrementality, both during the formalization (conceptualization) stage, and during the implementation phase.

One more issue related to this list is its vagueness. In other words, its positions can be (and actually have been) understood differently by different authors. Already the first position can be understood either as a formal expressive power (studied along e.g. [2]), or just as richness of the offered vocabulary. This particular ambiguity will be addressed later in the paper, but some other ones (e.g. the meaning of the word "transparency" or the understanding of what constitutes a "reflective" reasoning pattern) have not been discussed here. Therefore, (although we tried to avoid this) a possibility exists that the answers listed in the same row of the table are actually answering different interpretations of the questions.

The results are presented in Table 1.

Property	ML^2	MC	AIDE	KARL
1	FOL, META-LOGIC, DYNAMIC LOGIC	FOL META-LOGIC	RESTRICTED FOL	RESTRICTED FOL ALSO PROCEDURAL
2	YES		YES	YES
3	YES	YES	YES	YES
4	YES	YES	ONLY AT DOMAIN LEVEL	LOCALLY
5	YES	BY THE USER	YES	YES
6	YES	YES	YES	YES (1)
7	YES	YES	NO	YES
8	YES	YES	NO	NO
9	NO	YES	NO	NO
10	LIMITED	YES	LIMITED	YES
11	ONLY SIMULATION	BY THE USER	YES	YES
12	YES	UNDER DEVELOPMENT	PARTIAL	YES
13	COMPONENT-WISE	COMPONENT-WISE	NO	COMPONENT-WISE

Property	DESIRE	OBJ3	MILORD II	$K_{bs}SF$
1	3-VALUED FOL, META-LOGIC, TEMPORAL LOGIC ELEMENTS	ALGEBRAIC SPECIFICATION LANGUAGE	MULTI-VALUED LOGIC	ORDER-SORTED LOGIC
2	YES	YES	YES	
3	YES	YES	PARTIAL	
4	YES (2)	YES	YES (4)	Dmodules only
5	YES	NO (3)	YES	YES
6	YES	YES	YES	YES
7	YES	YES	YES	YES
8	YES	NO	YES	HARD
9	YES	NO	YES	VIA PARAMETERIZED DEDUCTION
10	YES	YES	POOR	YES
11	YES	YES	YES	NO
12	YES	YES	YES	YES
13	COMPONENT-WISE	YES	PARTIAL	COMPONENT-WISE

Table 1: Table based on the author's opinions about properties of the languages.

Notes:

1. Besides the three global layers of KADS, there is also a possibility of introducing a hierarchy of inference actions within the second (inference) layer and a corresponding (induced) hierarchy in the task layer;

2. Component-wise (within each module and within the supervisor) declarativeness in classical sense; compositionality (combining the component-wise semantics by the interactions): according to a temporal logic interpretation;

3. Control is imposed by appropriate ordering of specification with respect to the rewrite system used as the interpreter;

4. Component-wise (within each module) declarativeness according to multi-valued logic.

1 A purpose-driven comparison (Ramon López de Mántaras)

One of the key issues in trying to compare the different languages described during the Workshop is the intended purpose of these languages. It is clear that there are two categories: those languages aiming at delivering an executable system for some concrete applications (diagnosis of a particular disease, design of a particular device, etc.) and those aiming at obtaining a formal specification of general tasks, problem solving methods, domain models, etc. These two different purposes imply very different approaches during the definition and construction of the language (i.e. its constructs, expressive power, reasoning capabilities, control aspects, declarativeness, etc.). To be more precise, MILORD II is an example of a language aiming at delivering expert systems for complex diagnostic reasoning applications and therefore it has been designed following a bottom-up process. That is, the characteristics of MILORD II have been driven by the needs to solve the problems arising from the specific difficulties encountered in medical diagnostic reasoning tasks (definition of modules, rule-based language within the modules, reasoning with uncertainty, declarative control by means of meta-rules, etc.).

AIDE is also a language whose purpose is an executable system, although the emphasis is in the particular aspect of improving the explanations given by the final systems. In this case, this was the main reason to do the specification (high level model of expertise or conceptual model in KADS terminology) of the system.

At the other extreme we have those languages whose intended purpose is not to deliver a running system but a specification. As such, their conceptors have been able to follow a top-down approach during their design i.e. they are intended to be able to specify generic task models or KADS models ($\mathrm{K_{bs}SF}$, OBJ3, DESIRE, $(\mathrm{ML})^2$, KARL). Among them DESIRE, KARL and OBJ3 have some prototyping capabilities. DESIRE can automatically generate prototype implementations from a formal specification into either NEXPERT OBJECT code or PROLOG code. KARL offers the possibility of automatically mapping the obtained specification into an operational one that allows to evaluate the specified model of expertise by means of prototyping. The OBJ3 specification is executable as a term rewriting system and therefore it allows to generate and evaluate prototype specifications as in the case of KARL and DESIRE.

We may ask ourselves the trivial question (but not at all trivial answer) of how to make this approaches to converge to some middle ground. To do this, the bottom-up approach should include in its aims the genericity aspect paying the price of losing a computational efficiency (mainly in terms of speed) that in some applications could be unacceptable. On the other hand, the top-down approach could aim also to deliver efficiently executable systems. This of course would imply a loss of genericity power as well as a loss of solidity of the formal semantics of the language because in this case the language would be more adhoc. As an example of this let us consider the problem of deciding a therapy in a medical decision making task. In general, it is necessary to administrate several drugs simultaneously and this implies computing the different doses to be administrated. Such computations are to be performed by several operations. In a bottom-up approach we would introduce in the language particular operators that cope with the particular semantics of each operation, this lack of genericity results in a more difficult way of extending the set of operators when needed. On the other hand, in a top-down approach we would provide the language with the capability of defining any operator whose properties have to be syntactically defined in the language what would imply an extra effort to check them when introducing the knowledge and would also imply writing more code.

Another way to bridge the gap between the two approaches is suggested by observing the common need of local reasoning (modularization) in all the languages. As an example let us discuss the particularly interesting idea present in the MC language of defining a module (context in MC) as an axiomatic formal system. This allows to structure the specification in terms of local reasoning tasks performed within the different modules. It is indeed quite remarkable that all these languages independently of the approach (bottom-up or top-down) share such a common middle ground. This commonality suggests that perhaps the formal specification effort should be spent at the module level, i.e. inside the modules; leaving the issues of control (module or context switching, backtracking, etc.) not formally specified but tailored according to the different specific application problems being solved. The diversity of solutions provided by the different languages concerning the control aspects in the example task proposed in the Workshop is a clear evidence of the great difficulty to formally specify control. As an example of this diversity, in MILORD II the control is done declaratively through a reflection mechanism, in MC is not automatic but guided by the user, in AIDE, $(ML)^2$, KARL, and K_{bs}SF is procedural, in OBJ3 is functional (with recursive constructs) and in DESIRE is specified declaratively and realized procedurally by reflection mechanisms like in MILORD II. We think then that it is worth pursuing research efforts along this module level specification. After all why should we specify everything?

2 The treatment of heuristic knowledge (Frank van Harmelen)

The purpose of this section is to compare the various languages on how they deal with some specific aspects of the example task. In particular, we compare how the various languages have dealt with specifying the heuristic knowledge that is used in the scheduling task.

2.1 The heuristic knowledge used in the example task

Heuristic knowledge is used in the scheduling task in three places:

(H1) **Assumption generation:**

This heuristic is used in generating candidates for scheduling. The prescribed version of the heuristic says that any not currently scheduled activity can be scheduled in any time period which has not turned out to be inadequate for that time activity in the context of the current assumptions.

(H2) **Assumption selection:**

This heuristic is used in selecting an assumption from the candidates. The selected assumption is the one that will be added to the schedule. The prescribed version of the heuristic says that we should select an assumption that schedules an activity in a time period as early as possible.

(H3) **Assumption retraction:**

This heuristic is used in choosing which assumption to withdraw if a partial schedule turns out to violate any of the requirements. The prescribed version of the heuristic says that the most recently made assumption will be retracted.

Notice that H1 is not really a heuristic: heuristics are usually seen as rules that limit the search space in a way that may not be strictly correct all of the time. H1 neither limits the search space (since it prescribes all logically possible candidates as actual candidates) and is always correct (no incorrect decision will ever be made because of H1). H2 and H3 both are real heuristics in the sense that they both limit the search space and do not guarantee optimal (or even correct) solutions.

In general, heuristic knowledge (as opposed to factual knowledge) is an interesting focal point for comparing languages. We can compare the ways in which the languages deal with heuristic knowledge on the following grounds:

- does the encoding differentiate between the *role* of the heuristic in the reasoning process and the actual *contents* of the heuristic? For instance, a language may simply encode the selection heuristic H2, or it may explicitly specify the role of H2 as a selection heuristic, and subsequently state the specific version of the heuristic as given above.

- Furthermore, languages differ on their ability to specify a heuristic independently from its procedural use. Ideally, we want to separate the declarative contents of a heuristic from the procedural way it is used in the reasoning process.

- Often, heuristics only partially specify behaviour, and leave behaviour in the remaining cases unspecified. For instance, H2 does not specify which assumption should be selected if two candidates schedule activities at the same earliest time period. How do the languages deal with this indeterminacy in a specification?

Beside these general points, the heuristics H1, H2 and H3 are particularly interesting focal points for analyzing the various languages for the following reasons:

- H1 requires reference to the current process state ("any not currently scheduled activity");

- H2 is a control heuristic that requires reference to domain knowledge ("a time period as early possible");

- H3 requires reference to the process history ("the most recently made assumption").

2.2 How the languages specify the heuristics

Below we briefly characterize for each of the languages how they encode heuristics H1, H2 and H3, and discuss the languages on the basis of the criteria given above.

(ML)^2 In general, it is possible in (ML)^2 to distinguish between the role of a heuristic and the actual contents of it. H2 for instance is referenced in the axiom for assumption selection ([3], p. 36, lower part), but the actual contents of H2 (an ordering relation among candidates based on time-slots) is defined as a separate formula (in the lift-rule on p. 36, upper part). However, this clean separation for H2 has not been applied to the other two heuristics.

Similarly, the declarative contents of H2 (the definition of the ordering in the lift rule) has been separated from its procedural use (select a top element of the ordering). The price that is paid for this separation is that the declarative specification of H2 does not immediately suggest an efficient algorithm for actually selecting a top element of the ordering: a direct implementation of H2 as specified in (ML)^2 would result in a quadratic algorithm, while simple optimizations would lead to a linear computation.

$(ML)^2$ deals with the indeterminacy of H2 by not specifying the behaviour of the system in such cases. The reason given for this is that apparently the behaviour of the system is not important in such cases, and should be left as a decision for the implementor of the specification (and not arbitrarily enforced by the specification).

Reference to the process state (for H1, p. 35) is possible in $(ML)^2$ by referring to special variables that encode this state. Similarly, reference to the process history (for H3, p. 39) is possible in $(ML)^2$ through the use of special variables that keep track of the computational history, although the crucial exploitation of encoding the ObjectDescription as a list to maintain the ordering of the assignments is rather implicit in the specification. The access to domain knowledge by H2 is achieved through the lift-rules that instantiate the generic form of H2 (a preference relation) with a specific domain relation (the ordering among time-slots).

MC The MC formalization only specifies the required dependencies between input and output of the scheduling task, and does not specify how the search during the computation should proceed. Thus, neither H1 nor H2 are encoded in the MC specification. As a result, it is not possible to discuss how MC would deal with heuristic knowledge on the basis of the given specification.

Heuristic H3 is enforced in the MC specification by means of axiom PSC6 (p. 58), which exploits the fact that problem solving contexts are essentially represented by lists of assumptions whose order is maintained. This makes it possible to recover an old context by retracting the most recently made assumption.

AIDE In AIDE, all three heuristics are represented in the body of inference steps (tasks, p. 96–98), as in e.g. $(ML)^2$ as discussed above. No separation is made between the role of a heuristic and its actual contents (although this could have been done via the introduction of additional tasks). The heuristics are all specified in procedural form, and no separation is made between their declarative contents and their procedural use.

The heuristics have direct access to domain knowledge (enabling the formulation of H2). Reference to the process state (required for H1) is made through the predicates that range over domain objects ("is-scheduled-in") and over relations ("is-inadequate-for"). The process history (required for H3) is explicitly encoded via the "last" predicate ranging over domain relations.

The indeterminacy in H2 is resolved in H2 by marking all possible selections as "selected". Thus, the specification is left genuinely non-deterministic, and the AIDE interpreter makes a specific choice.

KARL Of all languages, KARL has most consistently separated the role of the heuristics from their actual contents, and their declarative contents from their procedural use. This is true of both H2 (p. 124–125) and H3 (p. 134–137).

The reference to domain knowledge for the purposes of H2 is resolved in way similar to $(\mathrm{ML})^2$, by means of an explicitly encoded relation between a generic preference criterion and a specific domain relation (p. 137).

Reference to the process state and process history is achieved by explicitly encoding the history through keeping track of the partial models that have been computed before, in the knowledge role design-states. This allows computation of all currently unscheduled activities (for the purpose of H1). A previous solution in KARL used the knowledge role design-objects for this purpose. Since this knowledge role contained all currently unscheduled activities, it was a sufficient encoding of the process state for the purposes of H1. Filtering out of previously known violations as part of H1 is not realized in the KARL specification.

Although the KARL specification does not follow the task description for the retraction heuristic, and instead specifies a rather more elaborate revision step, the encoding of the process history through the knowledge role design-states is clearly sufficient to specify chronological backtracking. The use of an explicit history is an improvement over the previous KARL specification which only represented the most recent assignment that was made, thus capturing the computational history only one level deep.

The situation concerning non-determinism in KARL is rather curious. The prescribed task description is underspecified in the selection step (since multiple candidates may remain after the selection step). Various languages (e.g. DESIRE or $(\mathrm{ML})^2$), choose to preserve this non-determinism in their specification, and leave it to the execution environment or to the implementor to make whatever choice is most convenient. KARL on the other hand removes the non-determinism from the specification altogether by specifying that remaining candidates are selected on the basis of the alphabetic ordering of their identifiers. The discussion section in the KARL paper indicates that this rather arbitrary choice could have been avoided by pursuing all alternative candidates simultaneously, but this would again have lead to a deterministic specification (albeit a different one). It is unclear why a genuine non-determinism (through underspecification) would be impossible in KARL.

DESIRE As before in $(\mathrm{ML})^2$, AIDE and KARL, DESIRE specifies the heuristic knowledge in the body of the inference steps (p. 171). Although no separation is made between the role of the heuristic knowledge and its contents, the declarative contents and procedural use of this knowledge are separated. A good example of this is H2, which declaratively specifies when an assumption is selected (p. 171), and only later specifies how this knowledge should be used procedurally: DESIRE first computes all eliminated assumptions and then applies a closed world assumption to compute the selected assumptions. The same declarative knowledge could have been used in entirely different control-regimes.

Concerning the non-determinacy of H2, the same phenomenon happens in DESIRE as in AIDE: the specification seems to be genuinely non-deterministic, and the interpreter for DESIRE makes a specific choice.

Reference to the state of the computation (for the benefit of H1) is possible in DESIRE by keeping track of (and being able to refer to) the partial models that have been computed for other theories during the computation. Reference to the process history (as required by H3) seems to be explicitly encoded in the DESIRE specification by the module "store-process".

OBJ3 The OBJ3 specification of the scheduling task strongly separates the role of a heuristic from its contents (p. 191–193). Since OBJ3 specifies no control regime whatsoever, the issue of separating declarative contents from procedural use does not arise.

Since no separation between domain and task knowledge is made in OBJ3, no problem arises in the reference of domain knowledge for the purpose of H2.

OBJ3 is one of the most extreme examples of how a language without facilities for referencing process history or state can be made to do so by explicitly encoding the computational trace of the system. In the OBJ3 example this is done through the data-type DESIGN-TASK on p. 183.

MILORD II The analysis of MILORD II in this context is somewhat problematic, since the MILORD II specification is based on a rather different algorithm from the one used in the other specifications. Whereas the other specifications use an algorithm that iteratively assigns an additional activity to a time period, the MILORD II algorithm simultaneously assigns multiple time intervals to all activities and proceeds by trying to narrow these assignments to those intervals that satisfy the requirements. Thus, the MILORD II specification is based on simultaneous assignment, whereas H1, H2 and H3 are phrased in terms of iterative assignment.

As a result of this, no explicit candidate generation and selection occurs in MILORD II. The closest analogous construction is the initial assignment of all time-periods to all assignments (rules R001-R004, p. 227). The analogue to H2 (candidate selection) is the strategy that in a conflicting schedule lower bounds of the possible periods for an activity are increased, instead of the upper bounds decreased (rules M001-M005, p. 227–228).

No separation is made between the role of these heuristics and their actual contents, and between contents and use.

The process state is available in the MILORD II specification because truth-assignments are used to capture the assignments of time-slots to activities and MILORD II can explicitly reason about truth-assignments. The predicate $K(a, [t1, t2])$ represents the fact that the time-slots in the interval $[t1, t2]$ are still considered possible for activity a, and because only lower-bounds are adapted, and are only increased, this encodes the fact that in the process history all time-intervals before $t1$ have already been considered and found inadequate for a. The "most recent assumption withdrawal" heuristic is irrelevant because of MILORD's simultaneous assignment strategy. The order in which different alternatives are being explored is declaratively unspecified, and is determined by the procedural implementation of MILORD's "res" operator.

K_{bs}SF K_{bs}SF successfully exploits its strong parameterization mechanism to achieve a high degree of separation between the generic role of the various heuristics (p. 242) and their specific contents (p. 247–248), and between their declarative contents and their procedural use.

The non-determinacy of H2 is dealt with by the built-in "one-of" operator which specifies an unspecified element from a set. Thus, as with other languages above, K_{bs}SF leaves the behaviour partially unspecified where this behaviour is not important for the desired behaviour of the system.

K_{bs}SF contains a notion of variables and assignment. These variables can capture the process state to a degree sufficient for expressing H1. Reference to the process history (required by H3) is more problematic, and is implicit in the execution of the design task. As pointed out by the authors, explication of such knowledge would require the full encoding of process states (in the same vein as in OBJ3 above).

2.3 Conclusions of the comparison on modeling the heuristics

From the above, we can distill the following general points:

- All languages allow a separation of existence of heuristic from actual contents, although not all example specifications actually exploit this to the fullest potential. Although such separation is strictly speaking possible in all languages, some languages seem to encourage this more than others. This might be explained by the different conceptual models that underlie some of the languages, because some of these conceptual models strongly advocate such separations. Thus, the underlying conceptual models rather than the syntactic constructions of the languages themselves can account for this difference.

- Not all languages allow separation of declarative contents from procedural use.

- All languages allow reference from control heuristics to domain knowledge, but in varying degrees of indirectness/genericity.

- Some languages have built-in facilities for reference to process state and/or reference to process history, other languages require this to be encoded by the specifier of a particular task.

- Most but not all languages seem to be able to deal with underspecified (non-deterministic) behaviour in the specification, and leave it either to the implementor of the specification or to the interpreter for the language to resolve the non-determinacy one way or the other.

3 Differences between the languages (Jacek Malec)

In this section we will focus on differences between the languages with respect to the following comparison criteria:

- Expressive power;

- Adequacy to specify the dynamic aspects of the reasoning pattern (abbreviated as "specification of control");

- Possibility to specify multi-level architectures adequately.

3.1 Expressive power

The task of judging the expressive power of a given language has to be done relative to some reference language, as there exists no objective measure of expressiveness of a language. In the context of specifying reasoning patterns one has to remember that the main issue is to analyze the language's ability to express some reasoning pattern rather than the domain knowledge this reasoning pattern is used for. On the other hand, if a specification formalism were able to express complex reasoning tasks but only for conceptually poor domains, then probably it would not be judged as useful nor even interesting.

So the analysis of the systems presented at the workshop will be done both in terms of the expressive power of their domain-knowledge language and the ability of specifying (complex) reasoning tasks consisting of appropriate manipulations (either syntactical or semantical) on the domain-language items. The obvious reference language immediately coming to mind is the language of the first order predicate calculus (FOL). It is usually seen as the standard language for expressing knowledge about some domain. It is also the standard language of meta-mathematics, in this case able to express (on the meta-level) some piece of reasoning performed on the object level.

Another question that has to be answered after choosing the reference language, but before making the comparison itself is: What is the procedure of comparing the expressive power of two given languages? At least two distinct forms of comparison are conceivable. On the first level one compares the expressiveness by checking whether all the theories expressible in one language can be expressed in the second. This corresponds to the approach advocated by [2]. In the case when two languages have the same expressive power in the abovementioned sense one can ask the question of economy or efficiency of representation, i.e. in which of the languages the description of a given theory is shorter (more concise, modular, more readable, etc.)? To be able to answer this question, one obviously has to provide some measure of this economy of representation. However, such concepts as "readability" or "modularity" are quite hard to be stated in a quantitative way.

Throughout the following section we have tried to adhere to the first understanding of the comparison, i.e. we are interested whether the sets of theories and reasoning patterns expressible in the compared languages are equal or not. However, this comparison has not been done in any formal way — such formal analysis is still a challenging open problem. The following should rather be seen as an initial effort towards providing guidelines for such formal comparison. In other words, the conclusions drawn have the status of disprovable conjectures rather than theorems, and reflect the intuitions of the author of this section.

Assuming that the scale from FOL to its standard extensions (modal, temporal, etc.) is taken as reference, we can partially order (with respect to their expressive power) the languages presented during the workshop. Such ordering should be done taking into account different aspects of reasoning process, as illustrated by the three bottom layers of the KADS architecture, i.e. the domain layer (for coding domain knowledge), the inference layer (for expressing inference rules) and the task layer (for specifying the reasoning pattern itself). There is a risk that such strict layering of comparison might appear incompatible with some intended way of using a specification language (this is actually the case for OBJ3 on one hand, where there is no discrimination of layers of specification at all, and for DESIRE, MC, and MILORD II on the other hand, where layering, or rather interconnection between modules, can be much more complex and moreover specified by the user). For the purpose of comparing expressive power of the presented systems we will analyze the domain level language used in each of them, flexibility of specification of reasoning patterns, and finally the control (or task) knowledge expressiveness. The discussion of layering aspects is deferred to the later section. Moreover, the reader is asked to bear in mind that the conclusions drawn here are "partial views" of the problem, strongly depending on the assumed criteria.

The relative ordering of the eight specification systems is presented in the following three diagrams.

Diagram 1: Domain knowledge expressiveness

```
FOL
```

```
----------------------------------
                      (ML)^2
                        MC
             AIDE
                   KARL
                      DESIRE
- - - - - - - - - - - - - - - - - - -
                        K_{bs}SF
                     OBJ3
- - - - - - - - - - - - - - - - - - -
                     MILORD II
```

Diagram 1 presents a relative ordering (or rather positioning) of the systems with respect to the expressive power of domain languages. Beginning with the commonalities, the first observation is that the presented languages either are equally or more expressive than FOL, with the exception of AIDE and KARL[1], so this property is not very distinctive in this context. When analyzing the economy of representation one can notice that all the systems except MILORD II use sorted languages. Another observation is that some systems explicitly support some form of modularization of theories and introduce operations on theories into their repertoire of accessible tools. As this topic is discussed later, we present here only a short list of features: $(\mathrm{ML})^2$ offers union of theories, $\mathrm{K_{bs}SF}$ offers a more extensive set of operations on theories, MILORD II and DESIRE introduce connectable modules corresponding to separate theories, and MC offers the ability to connect theories themselves (via bridge rules).

Although AIDE is classified as less expressive than other systems (because only a subset of FOL is supported), it offers the advantage of using semantic networks as a knowledge representation tool. The syntactic form of KARL's domain representation language is also a restricted form of FOL, but the authors claim that KARL's expressive power is increased due to the constructs introduced into the language.

$(\mathrm{MC})^2$ offers full first order logic. The possibility of extensions towards intensional (modal, temporal, etc.) logics is only mentioned. DESIRE on the other hand offers the three-valued first order logic as the domain language. However, comparing the "theory expressiveness", three-valued FOL is equal to standard FOL because each theory in the former language can be syntactically transformed into an appropriate theory in the latter. So the advantage of using DESIRE is noticeable on the "economy of representation" level.

A couple of languages (OBJ3 and $\mathrm{K_{bs}SF}$) incorporate algebraic specification techniques and therefore is separated in the diagram. OBJ3 has been put in

[1]The recent improvements of KARL have changed its properties, expressiveness included. However, the comparison made here refers to the systems as presented during the workshop.

the middle of the spectrum, because a direct comparison of expressive power of FOL and of this algebraic specification language is impossible. Some constructs easily representable in OBJ3 have to be laboriously written down in FOL (say, arithmetics or elementary set theory). On the other hand the algebraic form of the specification excludes e.g. advantages of unrestricted use of negation. As $K_{bs}SF$ offers both FOL and the tools of algebraic specification, it has been put towards the right of the diagram.

MILORD II is using multi-valued propositional logic for expressing the domain knowledge. It has been positioned at the level of FOL to express an (unproven) conjecture that the lack of first order constructs is compensated by multi-valuedness of the logic. However the ultimate ranking of MILORD II's object language may depend on the kind of domain knowledge one wants to deal with. Both kinds of domains (i.e. those requiring a first order language, and those requiring a truth scale more fine-grained than just 0 and 1) are easily imaginable, so we are not able to give any final conclusion here and hence MILORD II forms a separate entry in the diagram.

We can conclude this part by observing that expressiveness of the domain language is a non-discriminating criterion of comparison (cf. Diagram 1) as expressiveness of all the presented formalisms lays close to standard FOL. The real differences between the languages begin to appear in the inference-related parts of specification, i.e. in the form the meta- (or inference) knowledge is represented and used.

Diagram 2: Flexibility of specification of the meta-level reasoning

```
fixed structure                                    flexible
-----------------------------------------------------------
                      (ML)^2
                                                      MC
AIDE
                       KARL
                                                     DESIRE
                                 MILORD II
                                 K_{bs}SF
```

It can be argued that meta-level constructs increase the expressiveness of the domain language. On one hand this is the case, because the reasoning at the meta-level can extend the user's possibilities of asserting facts (e.g. by introducing assumptions into object-level theory). On the other hand, *from the domain knowledge level point of view* those meta-level-introduced facts have the same status as others - they are distinguishable only from the point of view of meta-level, where some contexts (or modules, or theories) have the status of hypotheses, and others of solid-fact-based theories. Therefore the analysis of expressiveness of the specification of reasoning pattern has to be separated from the issues related to the domain level language.

On one end of the spectrum we have AIDE with fixed, global structure of specification. The next group consists of $(ML)^2$ and KARL, where there also exist only three (domain, inference, and task) layers of description of reasoning pattern. In contrast to AIDE, however, those formalisms provide some tools for modularization and hierarchization, so that applicability of inferences can be limited to specific parts (subtheories) of the domain knowledge.

$K_{bs}SF$ is a two-level system, but due to the flexibility of configurations of its BModules it is a bit closer to the flexible side of the diagram. The same observation can be made about MILORD II. Although the set of allowed interactions in MILORD II is limited to object-object type only, and the meta-knowledge is local within each module, the user is still able to express more complex dependencies by imposing appropriate hierarchy of modules importing facts from other modules.

On the other end of the spectrum we have formalisms allowing arbitrary interconnections between reasoning modules, therefore offering the greatest degree of flexibility in specification. Both languages provide tools for interconnecting reasoning modules (bridge rules in MC, named forms of interactions in DESIRE). The basic advantage of this group of formalisms is that one can specify not only the single meta-level with the set of all possible inferences, but arbitrarily many of them, what allows for easy modeling of complex reasoning patterns, with dependencies spanning across a couple of meta-object relationships.

The issues related to the object-meta dependencies and to the degree of configurability of the specification modules are discussed in more detail in sections 3.2 and 3.3.

Diagram 3: Control (task) knowledge expressiveness

```
none                        partial                    total
------------------------------------------------------------
                                                  (ML)^2
               MC
                              AIDE
                                                  KARL
                                     DESIRE
               OBJ3
                                MILORD II
                                               K_{bs}SF
```

The biggest differences between languages are noticeable on the level of actual reasoning specification (understood as imposing some form of control over usage of existing inference rules/modules). The spectrum begins with MC system, where no information can be provided about the intended sequencing of usage of inference rules except via the meta-level specification. On one hand this gives the user the total freedom in specifying arbitrary problems, moreover in purely declarative way. On the other hand, the system's ability to solve a problem depends ultimately on user's skills in using the system and its supporting tools (such as `GETFOL`).

Another language offering no explicit way of affecting reasoning is OBJ3. Similarly to MC, the user is expected to provide a declarative specification of the reasoning patterns and the rest is left to the rewriting system interpreting the specification. Of course, as it is often the case with symbol manipulating programs, there exists a way of influencing the behavior of the system, namely by appropriate sequencing the formulae in the specification itself. However it can hardly be seen as a proper way of specifying the functioning of some reasoning pattern.

The next system in the diagram is MILORD II. In this case the specification of reasoning is done locally in each module by providing appropriate Horn-like rules at the meta-level of the module. The ordering of these rules is important, i.e. it influences the ordering of inferences performed at the object level. The actual specification of the reasoning pattern is done by imposing a hierarchy of dependencies between the modules. This hierarchy is built by using generic modules facility and module combination and refinement.

Another system providing declarative control locally for each module is DESIRE. Within a module, the situation is very similar to the previous one: the sequence of inferences can be controlled by providing a declarative representation in the form of meta-rules. Then the process of downward reflection provides the control for object-level inferences. What make DESIRE different form MILORD II are the explicit supervision rules declaratively specifying the flow of control between different modules. Another difference is that in DESIRE the meta-level knowledge resides in another (meta-)module, whereas in MILORD II it is given in the same module.

AIDE is equipped with a rule-based language (resembling PROLOG) specifying the control flow between different inference actions. Sequencing of rules is an important factor influencing the way inferences are performed.

$K_{bs}SF$ is a system equipped with a procedural language for specifying ordering of application of inference actions. This language includes such constructs as assignment, iteration, call and non-deterministic choice operator. So the user is able to control the behavior of the system in an explicit way, but the language itself, due to its procedural character, lacks a declarative semantics and therefore cannot be treated as a specification per se.

KARL is another system which provides the user with a procedural language for specification of control flow. Its set of constructs resembles the one of $K_{bs}SF$: it contains loops, branches, sequence operator and subtasking facility. Recently, after the workshop, the language has been assigned a declarative semantics based on dynamic logic.

$(ML)^2$ is the system offering the facilities of a procedural language for control flow specification (assignment, sequence and choice operators, non-deterministic iteration and test). Although simple, it appears sufficient to specify complex combinations of inferences, and on the other hand possesses a fully declarative semantics (as in the case of KARL, dynamic logic is used for this purpose).

3.2 Global vs. local layering of specification

One very important characteristic of the discussed languages is their support for introducing layering into a specification. Comparing them with respect to this aspect one can distinguish two groups of languages. The first one, influenced or motivated by the KADS methodology, supports or even forces global layering of a specification. This group consists of $(ML)^2$, AIDE, KARL and $K_{bs}SF$. The other group, consisting of DESIRE, MILORD II and MC, supports specifications in the form of a set of interconnected reasoning modules, where each module is treated as an independent unit. In case of MILORD II each module has its own object and meta-level, in case of DESIRE and MC the meta-level for some object module has to be specified in another module. This second approach obviously entails the first one because one can impose a rigid global layering when using languages belonging to this group. OBJ3 is somewhat outside this classification as it imposes no assumptions at all about the architecture of a specification.

The languages assuming global layering of a knowledge based system allow for further introduction of some hierarchy within the system, but it can be done only within a global layer. For example, in KARL we have the possibility of composing simple inferences into more complex ones, which are treated further on in the same way as primitive ones. There is also a mechanism (knowledge roles) allowing to restrict the applicability of an inference rule to some (sub)theory (some part of the domain knowledge base), thus introducing a kind of localization of reasoning. However, this limitation of the scope of reasoning patterns is secondary: the user is allowed to exploit it but not forced to do it. K_{bs}SF also supports the global stratification by supporting two kinds of modules corresponding directly to the task and inference (BModules), and domain (DModules) layers of KADS.

The systems assuming the structure consisting of a set of interacting but independent modules differ in their mechanisms for supporting global specification of control. In case of MC the user is free to do anything including introduction of a global context governing the reasoning performed within local contexts (this has been the case in the example task). DESIRE names the possible interactions between modules, thus supporting some kind of global division into domain and meta layers. However, the user is free to introduce any kind of possible interaction so the resulting structure may appear to be totally flat (all modules on the same level, with no interactions of the type meta-object or object-meta) or purely hierarchical (where each interaction is of the type object-meta). MILORD II limits interactions between modules to object-object only, thus forcing purely local application of the meta-knowledge.

3.3 Interactions between modules/theories

Another aspect differentiating the languages is their support for modularization of knowledge. As before, one can distinguish two major classes of systems: first strongly encouraging the user to encapsulate the "local" knowledge in modules and to explicitly state the interactions between the modules, and second limiting themselves to providing constructs for such modularization, but not forcing the user to exploit them.

The first group consists of MC, DESIRE, MILORD II, KARL, $(ML)^2$ and K_{bs}SF. Using MC system one is forced to explicitly state all interactions between contexts in the form of bridge rules. In the case of DESIRE the user has to provide explicit specification of input/output interface between modules (both at the same level or at different levels of reasoning). MILORD II and K_{bs}SF also require such import/export interface specification, although in case of MILORD II it is limited to the interaction between object levels (this is due to the overall control mechanism in MILORD II, where passing control to an from a module has to be done via import/export interface of the object level). In KARL modules are not transparent (i.e. accessible from outside) unless when explicitly stated. One can obviously use in each case one big module for specifying a problem, but it would be unreasonable provided the tools each system is equipped with. In $(ML)^2$ there exists a tool for (domain knowledge) modularization, namely the concept of union of sub-theories, but it is not a must, rather an add-on. However, on the higher levels $(ML)^2$ is also enforcing the user to think in terms of module-like entities.

The second group of systems consists of AIDE and OBJ3. Neither AIDE nor OBJ3 provide any explicit modularization tools. Everything is in the hands of a user who might or might not be able to introduce a reasonable modularization of a problem. One might object that neither the first class of system forces the user to extensively exploit modularization, and s/he would obviously be right. On the other hand, the manner of presentation of those systems stresses the idea of interacting modules whereas in the case of the latter group such possibility exists only potentially.

4 Commonalities and open problems (Jan Treur)

In this section we identify what characteristics are in common for a majority of the current languages and what open problems are felt that should be solved in the (near) future.

4.1 Commonalities between the languages

In the previous section it was pointed out on what aspects the eight specification languages differ. Stressing these differences, it may seem that currently there is no common view on the characteristics of a formal specification language for complex reasoning systems. However, during the workshop it turned out that there is a number of important common characteristics the majority of the designers of these languages aim at. The following list of required characteristics was created.

1. Modeling a complex reasoning system according to a composed structure;
2. Local declarativeness;
3. Multi-level view on the specification;
4. Distinct specification of static aspects and dynamic aspects;
5. Distinction between generic and domain-specific parts of the specification;
6. The use of object-meta distinctions in the specifications;
7. High level specification.

In this section we will discuss these characteristics.

1. Modeling a complex reasoning system according to a composed structure

A very general principle in (conventional) specification languages is that the overall specification is composed of more basic building blocks (modules). In our more specific case of specification of (complex) reasoning systems this can be worked out as a modularization of both the knowledge and of the reasoning pattern in terms of interactions between more local reasoning processes within each of the modules. Usually this modularization can be (and is) used to describe the task decomposition of a reasoning task. Points of attention are: to what extent the knowledge in the building blocks is specified in a declarative manner, how process aspects are specified, and what types of relations are possible between the building blocks.

AIDE Here the strategic knowledge is structured according to basic building blocks called tasks, adopted (and generalized) from Clancey's NEOMYCIN approach. They are specified by a (local) knowledge base mainly in a procedural form and they have mutual relations that are defined in a procedural manner. Process aspects are specified in the building blocks.

DESIRE Here basic building blocks are reasoning modules. They are specified by a declarative knowledge base. A number of relations between them are possible: union-like relations as well as relations of object-meta type. Some of the process aspects are specified in the building blocks themselves: (meta-)information on what target sets and targets can be used, and what inputs are requestable. For other process aspects (and specific control knowledge about them) separate modules can be defined that have an object-meta relation with the module(s) they are about.

KARL In KARL basic building blocks are inference actions (adopted from KADS). The task decomposition can be modeled hierarchically, according to a number of levels of abstraction. They are specified by a declarative knowledge base and can be related in a union-like manner. Process aspects are not specified in the building blocks themselves (global process aspects are specified in the task layer).

$\mathbf{K_{bs}SF}$ Basic building blocks are of two types, distinguished according to whether they specify data and knowledge (DModules) or behaviour (BModules, using the knowledge from the DModules). The DModules are specified by a declarative knowledge base and in BModules related by a number of union-like operations; also more procedural and inference-based relations are possible.

MC Basic building blocks are contexts that are defined by a (sub)language and a declarative knowledge base (called theory) containing knowledge relevant to the context. Contexts can be related according to bridge rules, a very general notion that can be used to specify in a declarative-like style inference rules involving more than one context. As examples of the use of bridging rules one can specify a union-like combination of contexts, or a specific object-meta relation between contexts. Process aspects are not specified.

(ML)^2 Basic building blocks in the first place are primitive inference actions known from KADS (formerly called knowledge sources); in (ML)^2 they are specified by declarative knowledge bases (called theories), that may be specified according to a further modularization. They can be combined according to a union operator. Process aspects are not specified in the building blocks themselves (global process aspects are specified in the task layer).

MILORD II Basic building blocks are modules that are specified by both (declarative) object-level knowledge and control knowledge (Modeling process aspects). They are related by union-like operations (via import and export). Also hierarchical decomposition of modules can be described.

OBJ3 Here basic building blocks are inherited from the notion of module in algebraic specification languages. Basically modules are defined by sets of equations expressing the relevant knowledge in a declarative, algebraic manner, and modules are related by a union-like operation. Process aspects are encoded in the equations as well.

Notice that the model of the reasoning task presented in the Example Reasoning Task Description [8] was based on the technique of task decomposition, a technique that is very familiar in knowledge analysis (for instance, one of the corner stones of KADS). This enabled the authors to use the possibilities for modularization provided by their languages.

2. Local declarativeness

From the previous point it has become clear that in almost all cases locally (in a basic building block) the knowledge can be described in a declarative manner. The view that a complex reasoning system should be specified according to local declarative knowledge bases is shared by all authors. Most languages already adequately support this view (see point 1.). In all these cases the system can be viewed as a collection of local theories that are turned into action in alternation.

However, if it comes to the point of how these theories interact with (relate to) each other, the approaches differ. In fact to give an overall semantics relating them is felt as one of the major open problems (see Section 4.2). Some approaches only allow a union-like relation between the theories, whereas others (DESIRE, $(ML)^2$, MC, MILORD II) allow object-meta relations as well. Other differences occur at the point of specifying control knowledge locally. In AIDE, DESIRE and MILORD II this is possible: in AIDE in a procedural form; in DESIRE and MILORD II in a declarative form (here downward reflection is used to transform declarative conclusions into the intended actions).

A discussion on declarative semantics for control knowledge can be found in 4. below.

3. Multi-level view on the specification

The languages DESIRE, $(ML)^2$, KARL, $K_{bs}SF$ are very similar in the manner in which they provide explicit language constructs to distinguish a global level description from the local level descriptions. By only looking at these global level specifications enable the specifier can get an overall view of the system. This distinction is similar to the distinction between domain layer and the higher layers (inference, task and strategic layer) in KADS. Using global level language constructs the basic building blocks can be used (by name) as objects, and (global) relations between them can be specified; this specifies the possibilities for global data flow between the components. Using the global level language the strategic or task knowledge (describing the global control flow) can also be specified (to be discussed in point 4.).

In MC language elements can be used to refer to (global) basic building blocks (contexts), but here the levels are not separated. They occur in an integrated manner: the global language elements occur (integrated with local level elements) in the bridging rules. Also in AIDE, MILORD II and OBJ3, local and global elements occur in an integrated manner.

4. Distinct specification of static aspects and dynamic aspects

In almost all languages it is possible to specify knowledge about (control of) the reasoning process state and behaviour in some explicit, separate form. Exceptions are MC (where control comes from the user) and OBJ3 (where control is encoded in an integrated manner in the equational knowledge).

In AIDE a control layer is specified that integrates both local and global process elements. The language has a procedural character. In DESIRE, $(ML)^2$, KARL, MILORD II and $K_{bs}SF$ a global task structure can be specified explicitly (at the global level as discussed in point 3.). The languages that are used differ. For $(ML)^2$ dynamic logic is used, for KARL, $K_{bs}SF$ and MILORD II a procedural language is used for the control at the global level. In DESIRE a declarative if-then format is used, with an additional interpretation of the conclusions by means of downward reflection. Knowledge on local process aspects can be locally specified in an explicit declarative manner in MILORD II and

DESIRE, with a similar additional interpretation of the conclusions by means of downward reflection.

As already mentioned in point 2 above, it is possible to specify control knowledge in a declarative form (see some of the approaches; e.g., DESIRE and MILORD II). However, the intended effect of conclusions about control is that some (procedural) actions should be undertaken (turning the words into the actions). These effectuations fall beyond the immediate declarativeness, one could say. On the other hand, meta-level architectures provide a standard possibility for such effects: there the notion of downward reflection is used to transform declarative conclusions into the intended actions. This can be given a formal declarative semantics, if temporal aspects are taken into account explicitly (see [12]). Another approach to obtain declarativeness for the control knowledge is based on dynamic logic. The paper on $(ML)^2$ shows this approach; recently also for KARL this dynamic logic approach was adopted.

5. Distinction between generic and domain-specific parts of the specification

This point concerns the separation of a generic part of the specification (generic task model or task-specific architecture) that can be reused in other domains. Many of the languages show possibilities for this option: DESIRE, KARL, $K_{bs}SF$, MILORD II, $(ML)^2$, OBJ3. In KADS-related languages such as $(ML)^2$ and KARL, the separation between specific and generic elements in a specification coincides with the distinction between domain layer and inference (and task) layer. As generic task models (or interpretation models, in terms of KADS), and libraries of them play a prominent role in knowledge analysis, there is much agreement that a specification language should support this notion.

6. The use of object-meta distinctions in the specifications

At least four of the specification languages use some form of distinction between object-level and meta-level information and knowledge. So, viewed globally this is a common point. However, the manner in which this distinction is applied differs significantly. In MILORD II in a module both an object-level and meta-level knowledge base can be included. Reflection occurs within the module, to control its reasoning. In MC and DESIRE there can be defined object-meta relations between components of the reasoning system. Reflection occurs as a specific type of interaction between components. In DESIRE and $(ML)^2$ (and, to a certain extent KARL) the global level is defined by means of a (global) meta-language with respect to the languages of the object-level. In DESIRE reflection is used to link global control with local inferences. In $(ML)^2$ the inference and task layer serve as a kind of meta-interpreter for the domain layer.

7. High level specification

Many of the languages aim at a specification that gives a high-level description of the reasoning system and its behaviour, not messed up with implementation details. It is, however, hard to give a clear definition of the term "high-level". Some authors claim their language is able to specify "knowledge level models", but the precise meaning of the term "knowledge level" is unclear. We will not try to give a decisive contribution to the discussion about the meaning of these terms. A related point of discussion is whether a distinction should be made between a specification of a knowledge model and a specification of the system (including all behavioural aspects as well).

For practical use of the specification languages, these points of discussion have some relevance. For instance, is it useful to have separate documents with the outcomes of knowledge analysis and of design, respectively? Can both be formally specified adequately? How do these documents relate? Is it possible to have one document specifying in detail a design for a reasoning system (including its behaviour) that can be an adequate basis for implementation in different available implementation environments (e.g., C++, PROLOG, LISP/SCHEME, a given Expert System Shell)?

4.2 Major open problems

The following important open problems can be recognized. They can be considered a research agenda for the area of formal specification of complex reasoning systems.

1. The lack of overall semantics of a system composed of components that each have their own (component-wise) semantics:

   - types of interactions between components;
   - declarative semantics and procedural semantics;
   - the use of a distinguished component for global control (supervisor, task layer).

   Recent approaches to this open problem are based on dynamic logic ($(ML)^2$; currently also KARL) or on a combination of dynamic and temporal semantics (DESIRE).

2. How to describe dynamic reasoning patterns in an adequate manner:

   - dynamic introduction and retraction of assumptions;
   - standard built-in possibilities for updating and revision of information;

- identification of types of strategic knowledge to control more sophisticated updating and revision processes (in relation to overall search strategies used).

3. How to integrate formal specification of reasoning systems in a conventional software engineering environment.

   In KARL components specified using Structured Analysis techniques can be integrated. In $\mathrm{K_{bs}SF}$ and OBJ3 conventional algebraic specifications can be integrated easily. In DESIRE interfaces to conventional specifications can be specified in what are called conventional modules.

4. Formal mappings between the different specification languages in order to make a more formal comparison.

   In a research project that will start in 1993 such a mapping will be investigated for $(\mathrm{ML})^2$ and DESIRE.

5. How to deal with a dynamic (external) environment where actions can be carried out, and where unexpected events can occur.

   Here the notion of state and (non-conservative) state transition may become important. DESIRE uses such a notion in an integrated manner. Also in $(\mathrm{ML})^2$ a notion of state has been introduced.

5 Conclusions

In this paper the state of the art of the area of formal specification languages for complex reasoning systems has been discussed and compared. Central issues in this field have been identified. It turns out that besides a number of differences between the current languages also there is consensus among a majority of the researchers about the following:

- Modeling a complex reasoning system according to a composed structure;
- Local declarativeness;
- Multi-level view on the specification;
- Distinct specification of static aspects and dynamic aspects;
- Distinction between generic and domain-specific parts of the specification;
- The use of object-meta distinctions in the specifications;
- High level specification.

Among the research issues for the near future is the question how to obtain an adequate overall formal semantics for these languages where both the static (data and knowledge) and dynamic (behaviour) aspects are covered in an integrated manner. Currently elements from dynamic logic, dynamic semantics and temporal logic are investigated to obtain such semantics.

References

[1] J. Angele, D. Fensel, and R. Studer. Formalizing and operationalizing models of expertize with KARL. Technical report, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, 1993.

[2] F. Baader. A formal definition of the expressive power of knowledge representation languages. Research Report RR-90-05, DFKI, 1990. Short version in Proc. ECAI-90, Stockholm.

[3] J. R. Balder, F. van Harmelen, and M. Aben. A KADS/$(ML)^2$ model of a scheduling task. In Treur and Wetter [13]. (This volume).

[4] E. Giunchiglia, P. Traverso, and F. Giunchiglia. Multi-context systems as a specification framework for complex reasoning systems. In Treur and Wetter [13]. (This volume).

[5] F. van Harmelen and J. R. Balder. $(ML)^2$: A formal language for KADS model of expertise. *Knowledge Acquisition Journal*, 4(1), 1992.

[6] G. Kassel and C. Gréboval. How AIDE succeeds in an example design task. In Treur and Wetter [13]. (This volume).

[7] D. Landes, D. Fensel, and J. Angele. Formalizing and operationalizing a design task with KARL. In Treur and Wetter [13]. (This volume).

[8] I. A. van Langevelde, A. W. Philipsen, and J. Treur. An example reasoning task description. In Treur and Wetter [13]. (This volume).

[9] I. A. van Langevelde, A. W. Philipsen, and J. Treur. A compositional architecture for simple design formally specified in DESIRE. In Treur and Wetter [13]. (This volume).

[10] A. T. Nakagawa, T. Sakakihara, and K. Futatsugi. Algebraic specification of reasoning systems. In Treur and Wetter [13]. (This volume).

[11] C. Sierra and L. Godo. Specifying simple scheduling tasks in a reflective and modular architecture. In Treur and Wetter [13]. (This volume).

[12] J. Treur. Towards dynamic and temporal semantics for meta-level architectures. Technical Report IR-321, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, 1992.

[13] J. Treur and Th. Wetter, editors. *Formal Specification of Complex Reasoning Systems.* Ellis Horwood, 1993. (This volume).

[14] L. in't Veld, W. Jonker, and J. W. Spee. Specification of complex reasoning tasks in K_{BS}SF. In Treur and Wetter [13]. (This volume).