MLPM: Defining a Semantics and Axiomatization for Specifying the Reasoning Process of Knowledge-based Systems

Dieter Fensel[1] and **Rix Groenboom**[2]

Abstract. We investigate the formal specification of the dynamic reasoning process of knowledge-based systems. The main contributions of the paper are: defining a formal framework for describing the dynamic reasoning behaviour of knowledge-based systems which unifies existing approaches; defining a semantics for the specification of the dynamic reasoning behaviour of a knowledge-based system within the states as algebra setting that overcomes several shortcomings of the existing approaches; and providing for the first time an axiomatization and proof theory of specification languages for knowledge-based systems. We achieve this by developing the logical language *MLPM* (*Modal Logic of Predicate Modification*).

1 INTRODUCTION

The *model of expertise* as developed in the KADS-I and CommonKADS projects [13] has become a widely used framework for developing and describing knowledge-based systems (KBS). Such a model of expertise can be used to describe the reasoning process and the knowledge required by this process in an implementation-independent manner. During the last years a couple of formal or executable specification languages have been developed for describing KBS and most of them are based on the KADS model of expertise. A recent survey on (most of) these languages can be found in [4].

Common for all formal specification approaches for KBS is that a formal semantics has to cover three aspects: the specification of static aspects of a KBS, the specification of the dynamics of a KBS (i.e., its reasoning), and the combination of both i.e. its overall semantics. For our study we restrict our attention to the second and third part as we think that the main improvement has to be made for the dynamic part. This part introduces also the main difference to specification languages from software engineering aiming on a pure functional description of a software system (cf. [4]). In general, most problems tackled with knowledge-based systems are inherently complex and require to specify a (heuristic) reasoning process and the required knowledge that enable feasible problem solving. An important part of the knowledge that must be specified is knowledge about the way to achieve a solution and not just declarative knowledge about what a solution should be.

During the paper, we will discuss a new semantical framework for specifying the reasoning process of a KBS. This framework integrates and improves existing approaches. In fact, we take an analysis of the two languages KARL [3] and $(ML)^2$ [9] as a starting point. As the technical core of the semantics of $K_{BS}SF$ is close to that of KARL, most of the results of the paper also apply to it.

A serious shortcoming of all of the specification languages for KBS lies in the fact that none of these languages provide a proof calculus that can be used to formally prove properties of specifications. Making progress in such a direction is precisely the aim of our work. Our starting point is the logic MLCM [6], a multi-modal logic for reasoning about state. MLCM has been developed to formalize the dynamic aspects of the wide-spectrum specification language COLD [2]. For the notion of state, COLD and the software specification approach Evolving Algebras [7] use, what we call the *states as algebra's* setting. In this paradigm, a state is modelled using a (many-sorted) algebra. State transitions modify this algebra.

The structure of the paper is as follows. First we briefly sketch the knowledge specification languages $(ML)^2$ and KARL focusing on their dynamics. Second, we derive requirements for an appropriate semantical framework for the specifications of the dynamics of KBS. Then we discuss the language MLCM stemming from software engineering. Actually we introduce a subset of the language (called MLCM-Light) that we require for our purpose. We have to generalize one of its mechanisms leading to the definition of *Modal Logic of Predicate Modification (MLPM)*. Finally, we provide a comparison with related work in software, knowledge engineering, and database update languages.

2 EXISTING APPROACHES

The languages KARL and $(ML)^2$ use variants of the KADS model of expertise as conceptual framework for specifying a KBS. We use a simple diagnostic task as an example to illustrate this model. The domain layer provides causal knowledge which can be used to relate findings to diagnoses and knowledge which can be used to assign preferences to possible diagnoses. The task of the KBS consists of finding the diagnosis with the highest probability for a given set of symptoms. The inference layer consists of two inference actions:

- *Generate*, creating possible hypotheses based on the given findings and the causal relationships at the domain layer.
- *Select*, assigning a preference to hypotheses and selecting the diagnosis with the highest preference.

A simple control flow is defined by first executing the inference *generate* and applying the inference *select* on its output. *Generate* derives all possible hypotheses which could explain some of the findings and *select* chooses the hypothesis with the highest preference (i.e., probability).

2.1. $(ML)^2$

The language $(ML)^2$ describes the reasoning behaviour by combining first-order (meta-) logic and quantified dynamic logic [8].

An inference action (called ***p***rimitive ***i***nference ***a***ction in $(ML)^2$) is described by a predicate and a logical theory. The inference action

[1] University of Amsterdam, Department SWI, 1018 WB Amsterdam, the Netherlands. E-mail: fensel@swi.psy.uva.nl.
[2] University of Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, the Netherlands. E-mail: rix@cs.rug.nl

ECAI 96. 12th European Conference on Artificial Intelligence, Budapest, 12.-16. August 1996. Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Lid.

select is modelled by a predicate $\text{pia}_{select}(hypothesis(X), preference(Z), diagnosis(Y))$. The definition of the inference action *select* is given by:

$$\text{pia}_{select}(hypothesis(X), Z, diagnosis(X)) \leftarrow \text{input}_{hypothesis}(hypothesis(X)) \wedge \text{input}_{preference}(Z) \wedge \neg(\exists Y\text{: input}_{hypothesis}(hypothesis(Y)) \wedge pref(hypothesis(Y), hypothesis(X)) \in Z)$$

The input predicate $\text{input}_{preference}$ used in the definition of the inference action pia_{select} is defined by reflection rules that connect truth in object- and meta-logic. As we abstract from this aspect of $(\text{ML})^2$ we will not go into any detail at this topic (cf. [9]). On the other hand, the knowledge role *hypothesis* does not provide domain knowledge for the inference actions. It collects the output of the inference action *generate* and provides it as an input to the inference action *select*. This dynamic character of *hypothesis* makes it necessary to define the input predicate $\text{input}_{hypothesis}$ at the task layer.

Quantified-dynamic logic is used to specify dynamic control at the task layer. Every predicate specifying an inference action at the inference layer together with the test operator *?* is regarded as an elementary program statement. For every such elementary program a history variable V_{pia} is defined which stores the input-output pairs for every execution step. The key idea is to non-deterministically choose a value binding of a logical variable by the test operator and store this value in a state variable.

Four types of task-layer operations are available for each inference action pia_i: checking whether an instantiation exists, checking whether an instantiation has already been computed, checking whether more instantiations exist, and actually computing and storing a new instantiation (*give-solution-*pia_i):

$$give\text{-}solution\text{-}pia_i\,(I,O) =_{\text{def}} (pia_i(I,O) \wedge \neg((I,O) \in V_{pia}))?;\ V_{pia} = \langle (I,O) \mid V_{pia} \rangle$$

These primitive programs and predicates can be combined using sequential composition, non-deterministic iteration and non-deterministic choice.

For our example, we have to define the input predicate $\text{input}_{hypothesis}$ and the control flow between the inference actions. The knowledge role *hypothesis* collects the output of the inference action *generate* and provides it as input to the inference action *select*. The following definition of the input predicate is the way in which $(\text{ML})^2$ can be used to define dataflow between inferences.

$$\text{input}_{hypothesis}(X) =_{\text{def}} \exists\, I_1, I_2 \text{ with } (I_1, I_2, X) \in V_{generate}$$

Dynamic Logic [8] uses *Kripke structures* to define a semantics for programs. A structure has the form $S = (D,F,P)$ consisting of a *domain* D, an interpretation F of the function symbols and an interpretation P of the predicate symbols. A *state over S* is a *function s* interpreting variables as elements of D. The interpretation of functions and predicates is fixed for all states. Programs p are interpreted by a binary relation on states. Formulas φ are interpreted by all states for which they are true.

The important construct in $(\text{ML})^2$ is *give-solution-pia* which gives one possible solution. The gist of the matter of this state transition is to apply the test operator *?* to the predicate *pia* which defines an inference action. $pia(I,O)?$ eliminates all states which interprets (i.e., substitutes) the variables I,O in a way which does not fulfil $pia(I,O)$. In the successor step, these variable substitutions are appended to the list of already computed input-output pairs.

2.2. KARL

The language KARL provides a formal and executable specification language for the KADS model of expertise by combining two types of logic. L-KARL, a variant of Frame Logic [10], is provided to specify domain and inference layers. It combines first-order logic with semantic data modelling primitives. A restricted version of dynamic logic is provided by P-KARL to specify a task layer.

L-KARL is provided for specifying inference actions and knowledge roles. The definitions of the inference *select* is given by:

$$x \in hypothesis \leftarrow y \in finding \wedge causality(cause{:}x, effect{:}y).$$

The logical language L-KARL used to describe the domain, the inference layer, and their connection has a Herbrand model semantics [11]. KARL allows stratified negation under the closed-world assumption using the minimal (i.e., perfect) Herbrand model as semantics.

KARL uses the logical language Procedural-KARL (P-KARL), a variant of dynamic logic, at the task-layer. The primitive programs correspond to *calling an inference action*, and atomic formulae indicate whether knowledge roles contain elements of a given class. Such primitive programs and atomic formulae can be arranged into sequences, loops, and alternatives. Programs may be combined to named subtasks, similar to procedures in programming languages. The task layer of our example looks like:

$$hypothesis := generate(finding);\ diagnosis := select(hypothesis)$$

Each inference action pia_i appears as a function symbol and each store *store* appear as a program variable *store* in the signature. The value assignments of the variables are used to represent the current state of the reasoning process. Such a state is characterized by the sets of true ground literals that are assigned to the program variables. Slightly simplified, each program variable is assigned to a set of true ground literals of a predicate symbol with the name of the variable.

The integration of the modal semantics of the task-layer and the Herbrand models of L-KARL is as follows: the models of L-KARL are used to define an interpretation for a P-KARL language, i.e., the perfect Herbrand model of the set of clauses which define an inference action *pia* is used to interpret a function symbol *pia* occurring in assignments in P-KARL. Each knowledge role is modelled by a (program) variable. The current state is represented by an assignment s of these variables. Notice, that a set of ground facts is assigned to each program variable. Slightly simplified, a transition is defined as:

$$I(output\text{-}role(X) := pia_i(input\text{-}role)) = \{(s_0,s_1) \mid s_1(output\text{-}role) = \text{perfect-Herbrand-model}(PIA \cup s_0(input\text{-}role))\} \text{ where } PIA \text{ is the set of clauses describing } pia.$$

3 REQUIREMENTS

$(\text{ML})^2$ uses the history variables in the definition of the input predicates and constructs as *give-solution-pia to combine the value assignments of logical and program variables*. V_{pia} collects the results of an inference action and can provide it to another inference action via the definition of an according input predicate. Often this is not very intuitive and introduces new modelling constructs that were not mentioned in the conceptual modelling context of the KADS model of expertise. We want to separate logical variables used in the definition of elementary transitions and program variables expressing the dynamic state of the reasoning process. Actually, the latter disappear in our framework. An algebra is used to express a state and state changes are expressed by changes of this algebra.

$(ML)^2$ provides the *local history* of each inference action *pia* by the list of values in V_{pia} but it does neither provide a description of the *global state* nor the *global history* as the states of the different inference actions are not uniquely related together. We will provide the *global* representation of the *state*. The *global* representation of the state seems to be quite natural for a sequential problem solver with a procedural control. Local representation of states would be more suitable for distributed problem-solving agent which cooperate during problem solving without a central control.

The restriction on states implies that we do not aim on specifying the history of the reasoning process. We can express control over the execution of inference actions as it is required by the task layer of a KADS model of expertise. Representing strategiec reasoning over the the sequence of reasoning states would require a different semantical setting. In dynamic logic two different paths of computing the same input-output tuple are not distinguished. The semantics of a program is the set of input-output tuples, i.e. a relation between input and output states. Intermediate states are not represented. Their representation would require a path semantics (cf. Transaction Logic [1], DESIRE [16]) where all intermediate states are captured by a sequence of models.

$(ML)^2$ changes a state by selecting precisely one new instantiation of a predicate (i.e. for the given state the predicate is true for one ground variable assignment and false for all others). KARL changes a state by setting all instantiations of a predicate to true which follow from the logical theory of an inference actions and its input. Both types of inferences appear in formalized KADS models. We call both update types *bulk* updates as they change the complete extension of a predicate in one step. Each ground literal of a predicate symbol becomes re-evaluated by such a state transition. A specification language for KBS must provide the possibility to specify such updates without requiring any commitments to how this is achieved by a specific algorithm.

4 MLCM-LIGHT

MLCM is an abbreviation of *Modal Logic of Creation and Modification* [6]. The logic is a modal logic in the tradition of Dynamic Logic [8]. In the traditional setting of dynamic logic, a state is represented as the assignment of variables. To modify a state the assignment of variables is changed. The same interpretation is given to the evaluation of the quantifiers. Take as example the following formula from dynamic logic; the evaluation of the existential quantification more or less `undoes' the states modification as $[x := 3]$ $(\exists\, x\, (x = 2))$ In MLCM, "programming variables" are represented as objects (i.e. function symbols). The variables in a quantification ranges over objects, and constants and functions refer to objects. So, in MLCM we write: $[c := 3]$ $(\exists\, x\, (x = 2))$

The signature to describe the states consists of constants, functions, and predicate symbols. One state is described by the set of defined constants and their value, by the values of the functions, and the truth values of the ground literals. MLCM provides three elementary transitions between states. Defining a new constant, changing the value of a function at one point, changing the truth value of predicates (i.e., of a ground literal). Control over these transitions can be defined by the usual constructs of dynamic logic (i.e., sequence, branch, loop). As we will only require one type of state transitions (changing the truth value of a predicate) we will skip the two other constructs and refer the reader to [6]. In MLCM-Light, we include only the modification of predicates.

4.1. Syntax

The logic MLCM is an extension of predicate logic with partial functions, obtained by adding formulas $[\alpha]A$, where α is a program expression. For the construction of complex program expressions the same operations are used as in dynamic logic. The syntax of MLCM-Light reads:

VAR x	(a countably infinite collection of variables)
FUNC f	(function symbols, with arity ≥ 0)
PRED p	(predicate symbols, with arity ≥ 0)
TERM t ::=	$x \mid c \mid f(t_1,..., t_n)$
PROG α ::=	$p(t_1,..., t_n) :\leftrightarrow A \mid A? \mid \alpha ; \alpha \mid \alpha \cup \alpha \mid \alpha^*$
FORM A ::=	$(t = t) \mid p(t_1,..., t_n) \mid A \wedge A \mid \neg A \mid \forall x\, A \mid [\alpha]A$

4.2. Semantics

The definition of the semantics of MLCM-Light has two steps. First we define the notion of *structure*, a kind of proto-model in which terms and formulae of MLCM-Light can be interpreted. Then we restrict this notion to *models* by imposing requirements on the accessibility relations corresponding to the program statements: the interpretation of formulas is used in the formulation of these requirements. A *structure* is a quadruple $M = \langle U,F,W,R \rangle$, where

- U is a set of objects;
- $F = \{f_M \mid f \in \text{FUNC}\}$ with $f_M : U^n \rightarrow U$ (n the arity of f);
- $W \neq \emptyset$ is the collection of *worlds*. A world w is a set $\{p_w \mid p \in \text{PRED}\}$ with $p_w \subseteq U^n$ (n the arity of p). Each of these worlds w represents a state by defining an interpretation of the predicate symbols.
- $R = \{R_{\alpha,a} \mid \alpha \in \text{PROG}, a \in \text{ASS}\} \subseteq W \times W$ is a collection of binary relations on W.
  Here, ASS is the set of assignments. Assignments $a \in \text{ASS} = \text{VAR} \rightarrow U$ and pointwise modification $a[x \rightarrow u]$ (where $x \in \text{VAR}$, $u \in U$) of an assignment are defined as usual.

For the definition of the interpretation we observe only unary function and predicate application. Let $[t]_a$ denote the interpretation of term t with assignment a. For $w,a \models A$, the interpretation of formula A in world w with assignment a, we have

- $w,a \models p(t)$ $\quad =_{\text{def}} \quad [t]_a \in p_w$
- $w,a \models (s = t)$ $\quad =_{\text{def}} \quad [s]_a = [t]_a$
- $w,a \models \neg A$ $\quad =_{\text{def}} \quad$ **not** $(w,a \models A)$
- $w,a \models A \wedge B$ $\quad =_{\text{def}} \quad w,a \models A$ **and** $w,a \models B$
- $w,a \models \forall x\, A$ $\quad =_{\text{def}} \quad$ **for all** $u \in U\, (w,a[x \rightarrow u] \models A)$
- $w,a \models [\alpha]A$ $\quad =_{\text{def}} \quad$ **for all** $w' \in W\, (wR_{\alpha,a}w' \Rightarrow w',a \models A)$

A structure is called a *model* if it satisfies a number of requirements for the relations $R_{\alpha,a} \in R$.

- $wR_{p(s) :\leftrightarrow A,a}w'$ iff w' is the unique world with: $([s]_a) \in p_{w'}$ **iff** $w,a \models A$ for all assignments a and $q_{w'} = q_w$ **for all** $q \in \text{PRED} \setminus \{p\}$.
- $R_{A?,a} = \{(w,w) \mid w,a \models A\}$
- $R_{\alpha;\beta,a} = R_{\alpha,a} \circ R_{\beta,a}$

- $R_{\alpha\cup\beta,a} = R_{\alpha,a} \cup R_{\beta,a}$
- $R_{\alpha^*,a} = R^*_{\alpha,a}$

4.3. Axiomatization

The axiomatization of MLCM is already proven complete for the fragment without the *-operator in [6] but it is claimed to be complete and a publication on this issue is being prepared. In the following we discuss the axioms of MLCM that are relevant in the context of MLCM-Light. MLCM-Light is axiomatised as follows ($[\alpha]\Gamma$ stands for the set $\{[\alpha]A \mid A \in \Gamma\}$). The logic has the following inference rules ($\Rightarrow$ denotes meta implication):

- **MP** $A, A \rightarrow B \Rightarrow B$
- **Subst** $A \Rightarrow [t/x]A$
- **N** $\Gamma \vdash A \Rightarrow [\alpha]\Gamma \vdash [\alpha]A$
- **Dec** $\Gamma, A \vdash B \Rightarrow \Gamma \vdash (A \rightarrow B)$

For the predicate logic part, i.e. the fragment without modal operators, MLCM-Light has the following axioms: all tautologies of propositional logic, the axioms for equality, the axioms for the $\exists$ and $\forall$ quantifiers. Derivability, $\Gamma \vdash A$ (A is derivable from Γ), is defined inductively as usual: all $A \in \Gamma$ and all instances of axioms are derivable from Γ, and if all premises of a rule are derivable from Γ then so is the conclusion. A collection of formulae Γ is called *consistent* iff $\Gamma \nvdash \bot$. The following axioms are provided for predicate modification:

- **PM1** $A \leftrightarrow [p(s) :\leftrightarrow C]A$ for all A not containing p
- **PM2** $s = x \rightarrow (C \leftrightarrow [p(s) :\leftrightarrow C]px)$
- **PM3** $s \neq x \rightarrow (px \leftrightarrow [p(s) :\leftrightarrow C]px)$
- **PM4** $\forall x[p(s) :\leftrightarrow C]A \leftrightarrow [p(s) :\leftrightarrow C](\forall xA)$
  for all C not containing p.

The following axioms are provided for the program constructs:

- **?AX** $[A?]B \leftrightarrow (A \rightarrow B)$
- **;AX** $[\alpha ; \beta]A \leftrightarrow [\alpha][\beta]A$
- **∪AX** $[\alpha \cup \beta]A \leftrightarrow ([\alpha]A \wedge [\beta]A)$
- ***AX** $[\alpha^*]A \leftrightarrow (A \wedge [\alpha][\alpha^*]A)$
- **INF** $\{A \rightarrow [\alpha^n]B \mid n \in \mathbb{N}\} \Rightarrow A \rightarrow [\alpha^*]B$

Here, $[\alpha^n]$ is recursively defined as: $[\alpha^0]$ = [true?] and $[\alpha^{n+1}] = [\alpha][\alpha^n]$.

5 MLPM

In MLCM, the elementary programs are assignments of a value to one constant or the modification of one function value or an assignment of a truth value to one ground literal. In KARL, the elementary program in dynamic logic is itself a complex transition. It can be viewed as a complex logical program describing declaratively an entire set of these updates which should be done in parallel. In the following, two new language constructs are introduced in MLCM to allow the evaluation of a logic formula as elementary transitions in dynamic logic. The ε-operator expresses non-deterministic selection of one ground literal. A formula can be used to restrict the set of possible ground literals from which one is chosen. All other ground literals of the predicate are set to false. The λ-operator allows updates all ground literals of a predicate according to the truth values of a formula. All ground literals are set to true for which the according variable assignments evaluate the formula to true. All other ground literals of the predicate are set to false. The elementary programs that we add to the syntax are $p :\leftrightarrow \varepsilon x.\ A$ and $p :\leftrightarrow \lambda x.\ A$. A is a first-order formula containing only bounded variables expect the free variable x. For these operations, we define the following semantics:

- $wR_{p(s) :\leftrightarrow \varepsilon x.A,a}w'$ iff w' is ***a*** world with $u \in p_{w'}$ **for an** $u \in U$ **with** $w,a[x \rightarrow u] \models A$ for an assignments a **and for all** $u' \in U \setminus \{u\}(u' \notin p_{w'})$) and $q_{w'} = q_w$ **for all** $q \in$ PRED $\setminus \{p\}$
- $wR_{p(s) :\leftrightarrow \lambda x.A,a}w'$ iff w' is ***the unique*** world with **for all** $u \in U$ ($u \in p_{w'} \Leftrightarrow w,a[x \rightarrow u] \models A$) for all assingments a and $q_{w'} = q_w$ **for all** $q \in$ PRED $\setminus \{p\}$

A program using the λ-operator also succeeds in the case where no ground instantiation of the predicate p is evaluated to truth (i.e., $\neg A$ is a tautology). A program using the ε-operator fails in that case as *no* instantiation of the predicate with truth value true can be produced. Note also that the program $p :\leftrightarrow \varepsilon x.A$ is non-deterministic.

(i)	**EM1**	$\forall y\ [p :\leftrightarrow \varepsilon x.\ A]B \leftrightarrow [p :\leftrightarrow \varepsilon x.A](\forall y\ B)$	
(ii)	**EM2**	$[p :\leftrightarrow \varepsilon x.A](\exists! x\ (px))$	
(iii)	**EM3**	$\exists x\ A \rightarrow \exists x\ (A \wedge [p :\leftrightarrow \varepsilon x.A]\ px)$	
(iv)	**EM4**	$B \leftrightarrow [p :\leftrightarrow \varepsilon x.A]B$	for all B not containing p
(v)	**LM1**	$\forall y\ [p :\leftrightarrow \lambda x.A]B \leftrightarrow [p :\leftrightarrow \lambda x.A](\forall y\ B)$	
(vi)	**LM2**	$\langle p :\leftrightarrow \lambda x.A\rangle B \leftrightarrow [p :\leftrightarrow \lambda x.A]B$	
(vii)	**LM3**	$\forall x\ (A \leftrightarrow [p :\leftrightarrow \lambda x.A](px)$	
(viii)	**LM4**	$B \leftrightarrow [p :\leftrightarrow \lambda x.A]B$	for all B not containing p

For both axiomatizations holds that the first axiom reflects that the universe does not change. The second and third state the change of the predicate p. The last axiom states that the other predicates do not change.

The λ-operator of MLPM can be used to model the bulk-update of KARL. The ε-operator of MLPM catches the conceptual and technical core of $(ML)^2$. One instantiation of the predicate is chosen non-deterministically. On the other hand, we have not yet decided to hardwire the mechanism *give-solution-pia* of $(ML)^2$ in MLPM as there are some problems related to this construct. For example, an inference action multiply fails if it gets the same input values a second time. The definition of the example can now be given as follows:

$$hypothesis :\leftrightarrow \lambda y.(\exists x(finding(x) \wedge causality(x,y)));$$
$$diagnosis :\leftrightarrow \varepsilon x.(hypothesis(x) \wedge \neg \exists y(hypothesis(y) \wedge (preference(x) < preference(y))))$$

6 RELATED WORK

Evolving Algebras (*ealgebras*) [7], a recent specification approach from software engineering, also use an algebra to express a *state*. *Transitions* between states can be expressed by *function updates*. This point-wise modification of a function corresponds to the type of transition in MLCM. Ealgebras provide also means to specify parallel algorithms that can be used to express our type of bulk-updates (i.e. the *choose* and the *range* constructs, cf. [7]). Otherwise, current operationalizations and axiomizations of ealgebras are restricted to the case of point-wise updates. A further difference to our approach is that MLPM provides a richer vocabulary to express control over the execution of transitions.

The language DESIRE uses the notion of meta-layered compositional architecture to specify a KBS. An important distinctions between DESIRE and the specification languages which were discussed above is that DESIRE uses its object/meta-level distinction *to specify and to reason about* flexible control of object-level inferences whereas languages as $(ML)^2$ or KARL define control of object-level inferences by a procedural language. From a semantical point of view a severe difference between DESIRE (see [16]) and $(ML)^2$ or KARL lies in the fact that the former uses linear temporal logics for specifying the dynamic reasoning process whereas the latter use dynamic logic. In dynamic logic, the semantics of the overall program is a binary relation between its input and output sets (M_i,M_o). Two different paths of computing the same input-output tuple are not distinguished. In DESIRE, the entire reasoning trace T which leads to the derived output is used as semantics $T = M_i,M_1,...,M_n,M_o$. DESIRE uses a sequence of models to define the semantics of a specification as it should be able to express strategic reasoning over the history of the derivation process.[1] This type of reasoning, that was allocated at the strategic layer in earlier versions of KADS, is beyond the scope of our approach.

PDDL [15] and DDL [14] are developed for the logical specification of database updates and are closely related to our efforts. *Propositional dynamic database logic* (PDDL) defines a variant of propositional dynamic logic by restricting elementary state transitions (i.e., elementary programs) to two pre-defined types. The (point-wise) update of the truth value of one proposition and the bulk update of the truth values of a set of propositions according to the minimal Herbrand model of a set of propositional Horn clauses. PDDL is close in spirit to KARL where the perfect Herbrand model of a set of clauses is used to define elementary updates. Dynamic database Logic (DDL) [14] syntactically extends the ideas of PDDL to the first-order case. Again two types of pre-defined updates (i.e., elementary programs) are provided: Updating the truth values of all ground literals over a predicate symbol according to the truth values of the according variable assignments of a first-order formula and the non-deterministic selection of one of the variable assignments that makes the formula true. Besides some details both types of updates are identical with the λ- and ε-operator of MLPM. As in MLPM, a state is described by an interpretation of the predicate symbols. [14] defines DDL without function symbols and provide a complete axiomatization under the domain closure and unique naming assumption (i.e., the expressive power of the language is restricted to the propositional case). Still, the close correspondence of formal specification languages for KBS based on the KADS model of expertise and languages for specifying database updates is one the significant and surprising (?) outcomes of our analysis. It looks very promising to increase the cross-fertilization of both research fields in the future.

7 CONCLUSION

The logic MLPM generalizes the gist of the matter of existing specification languages for KBS and provides an adequate mathematical framework for their uniform formalization. In a nutshell, our approach uses algebras to represent states of the reasoning process, bulk-updates that change the truth values of predicates to express state transitions and procedural constructs to define control over the execution of transitions. In the paper, we define a formal semantics and an axiomatic semantics for specification languages for KBS.

Automated support for proofs in MLPM could be provided by the Karlsruhe Interactive Verifier (KIV) [12]. It is based on dynamic logic and applied for program verification. [5] provide a first case study on using KIV for the verification of kbs.

1. Still, the current version of the language does not make use of this feature of its semantics.

Acknowledgment

We would like to thank Joeri Engelfriet, Pascal van Eick, Frank van Harmelen, G. R. Renardel de Lavalette, Arno Schönegge, Remco Straatman, Mark Willems and two anonymous reviewers for valuable comments on early drafts of the paper and inspiring discussions on topics related to the paper.

8 References

[1] A. J. Bonner and M. Kifer: Transaction Logic Programming. In *Proceedings of the 10th International Conference on Logic Programming (ICLP)*, Budapest, Hungary, June 21-24, 1993.

[2] L.M.G. Feijs and H.B.M. Jonkers: *Formal Specification and Design*, Cambridge Tracts in Theoretical Computer Science 35, 1992.

[3] D. Fensel: *The Knowledge Acquisition and Representation Language KARL*, Kluwer Academic Publ., Boston, 1995.

[4] D. Fensel: Formal Specification Languages in Knowledge and Software Engineering, *The Knowledge Engineering Review*, 10(4), 1995.

[5] D. Fensel, A. Schoenegge, R. Groenboom and B. Wielinga: Specification and Verification of Knowledge-Based Systems. Submitted to the ECAI'96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems.

[6] R. Groenboom and G.R. Renardel de Lavalette: Reasoning about Dynamic Features in Specification Languages. In D.J. Andrews et al. (eds.), *Proceedings of Workshop in Semantics of Specification Languages*, October 1993, Utrecht, Springer Verlag, Berlin, 1994.

[7] Y. Gurevich: Evolving Algebras 1993: Lipari Guide. In E.B. Börger (ed.), *Specification and Validation Methods*, Oxford University Press, 1994.

[8] D. Harel: Dynamic Logic. In D. Gabby et al. (eds.), *Handook of Philosophical Logic, vol. II, Extensions of Classical Logic*, D. Reidel Publishing Company, Dordrecht (NL), 1984.

[9] F. van Harmelen and J. Balder: $(ML)^2$: A Formal Language for KADS Conceptual Models, *Knowledge Acquisition*, 4(1), 1992.

[10] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of the ACM*, vol 42, 1995.

[11] J.W. Lloyd: *Foundations of Logic Programming, 2nd Editon*, Springer-Verlag, Berlin, 1987.

[12] W. Reif: The KIV Approach to Software Engineering. In M. Broy and S. Jähnichen (eds.): Methods, Languages, and Tools for the Construction of Correct Software, LNCS 1009, Springer-Verlag, Berlin, 1995.

[13] A.T. Schreiber, B. Wielinga, J. M. Akkermans, W. Van De Velde, and R. de Hoog: CommonKADS. A Comprehensive Methodology for KBS Development, *IEEE Expert*, 9(6):28—37, 1994.

[14] P. A. Spruit, R. Wieringa, and J.-J. Meyer: Dynamic Database Logic: The First Order Case. In V.W. Lipeck and B. Thalheim (eds.), *Fourth International Workshop on Foundations of Models and Languages for Data and Objects*, Workshop in Computing, Springer-Verlag, 1993.

[15] P. A. Spruit, R. Wieringa, and J.-J. Meyer: Axiomatization, Declarative Semantics and Operational Semantics of Passive and Active Updates in Logic Databases, *Journal of Logic Computation*, 5(1), 1995.

[16] J. Treur: Temporal Semantics of Meta-Level Architectures for Dynamic Control of Reasoning. In L. Fribourg et al. (eds.), *Logic Program Synthesis and Transformation - Meta Programming in Logic,* , Springer Verlag, LNCS 883, 1994.