A LOGIC FOR THE SPECIFICATION OF MULTI-OBJECT SYSTEMS

Jan Broersen Roel Wieringa

Faculty of Mathematics and Computer Science, Vrije Universiteit
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
broersen@cs.vu.nl

Abstract: We present Multi-Object Dynamic Logic (MODL), a generalization of Dynamic Logic of which the intended use is the declarative specification of systems that are conceptually described by a multitude of objects. In an example specification of the controls of a railroad crossing we demonstrate how MODL can be used to give semantics and reasoning capacity to graphical languages for communicating multi-object systems. Finally we study to what extend temporal and mixed dynamic/temporal properties can be expressed in MODL.

1 INTRODUCTION

There are a number of languages for the specification of systems consisting of sets of communicating objects. Graphical languages for object-oriented system specification often use one of the many dialects of statecharts [15] [17] [1] [10]. Like all diagram-based languages, statecharts need to be supplemented with a mechanism to reason about properties of the specified system. Logical languages like temporal logic allow reasoning about safety and liveness properties [20] [6] [5], but are traditionally less strong in reasoning about properties of named actions. Reasoning about actions is the province of Modal Action Logic and its generalization concerning programs, Dynamic Logic [9] [11] [18]. In earlier papers we showed how to use dynamic logic to specify the behavior of objects [23] [25]. Here, we show how to extend Dynamic Logic (DL) with **steps** of concurrent actions to something we call Multi-Object Dynamic Logic (MODL), which can be used to reason about systems of communicating objects. In a companion paper, we show how steps can be used to give a high-level semantics to UML statecharts [24].

We start this paper in sections 2 and 3 by discussing some properties that we argue any logic based declarative specification language for multi-object systems should satisfy. In section 4 we define Multi-Object Dynamic Logic (MODL), which satisfies

P. Ciancarini et al. (eds.), *Formal Methods for Open Object-Based Distributed Systems*

these properties. Section 5 presents the example of a railroad crossing control system specified in a graphical specification language for multi-object systems. We show how to translate this specification into MODL-formulas, which makes it possible to prove liveness and safety properties. In section 6 we investigate the suitability of MODL to express temporal and mixed dynamic/temporal properties. Section 7 concludes with the conclusions.

2 DESIRED ENTAILMENT PROPERTIES

We want to specify actions and steps declaratively by stating conditions on what holds before and after they are performed. These conditions are usually referred to as pre-end postconditions. If in a **specification** we state that action σ has postcondition ϕ, we take the strong view that ϕ is caused by α. With this in mind we can formulate the first desired entailment property:

> **P1: Simultaneous actions accumulate the effects of their constituent parts.** This means that if ϕ is the postcondition of action σ or of action σ' it is also a postcondition of the simultaneous step $\sigma \,\&\, \sigma'$.

In MODL this property can be expressed as $[\sigma]\phi \vee [\sigma']\phi \rightarrow [\sigma\&\sigma']\phi$. We think this is a natural requirement to make. Denying this property would be to accept that the postconditions (effects) of the actions σ and σ' do not in any way relate to the postconditions (effects) of $\sigma \,\&\, \sigma'$. All other logics [8] [7] [12] [14] [13] [26] that introduce simultaneous/concurrent actions or processes in Dynamic Logic satisfy the property P1, which makes it reasonable to suggest that this property is generally accepted as being desirable. The second desired entailment property we mention concerns the reverse of property P1:

> **P2: Effects may augment each other.** So a postcondition of the simultaneous step $\sigma \,\&\, \sigma'$ is not necessarily a postcondition of action σ or of action σ'.

To show that this is a desirable property, we look at the MODL-expression for the assertion that two actions σ and σ' can not be performed together. In MODL this can be expressed as $[\sigma \,\&\, \sigma']\bot$. Now if a logic does not have property P2, we are able to ascribe the postcondition $\bot$ to one of the actions σ or σ'. This inference is to strong to be useful for a declarative specification language for multi-object systems. Concurrent Dynamic Logic, defined by Peleg [14] [13], however has this property. In CDL "effect"-properties of compound processes can always be completely ascribed to some part. In CDL we have:

$$[\sigma\&\sigma']\phi \leftrightarrow [\sigma]\phi \vee [\sigma']\phi \qquad (\text{Dual: } \langle\sigma\&\sigma'\rangle\phi \leftrightarrow \langle\sigma\rangle\phi \wedge \langle\sigma'\rangle\phi)$$

This disqualifies CDL for the kind of reasoning and specification we want to employ.

3 DESIRED PROPERTIES OF MODELS

When logic is used for specification purposes, the models of the logic play a more concrete role than in other uses of logic. For instance, a model (or one of the models that is considered the **intended** one) may form the basis for an execution or simulation

of the specification. As a consequence of this we are also interested in properties of models that are not necessarily expressible in the logic that is used for specification. This brings us to the following desired property:

P3: Models should be natural representations of the systems the logic reasons about.

Standard Propositional DL-models can be easily related to single-object systems: the states of the models are system states and the transitions of the models are named actions that relate these states. But when we make the step from single object systems to multi-object systems, standard PDL-structures are no longer natural representations. A natural way to define models for multi-object systems is to generalize transition labels to sets of actions (see section 4.1), thus interpreting steps by individual transitions in the structure. However, in the logic $\mathcal{DIFR}$, defined by De Giacomo et al. [8], standard PDL-structures are used to represent simultaneous actions. In $\mathcal{DIFR}$ simultaneous actions are represented by intersecting reachability relations. Simultaneous execution of a and b is associated with the two transitions as visualized in figure 1.

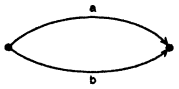

Figure 1. Simultaneity of actions a and b in $\mathcal{DIFR}$.

This is non-standard because normally two transitions leaving a state are interpreted as a choice. Interpreting this situation as simultaneous execution, as is done in $\mathcal{DIFR}$, therefore requires looking at these models in a different, unnatural way. This disqualifies $\mathcal{DIFR}$ for our purposes. We now formulate a second desirable property of models for specification languages.

P4: Models should contain as little redundant information as possible.

Redundant information is sometimes introduced in models as the result of explicitly 'coding' some desired entailment property concerning action connectives. An example of this is the models used by Lodaya et al. [12]. The desired property they code explicitly in their models is that each state that is reachable concurrently, is also reachable by any sequential interleaving of the concurrent actions ($\langle \sigma \& \sigma' \rangle \phi \rightarrow (\langle \sigma ; \sigma' \rangle \phi \wedge \langle \sigma' ; \sigma \rangle \phi)$). This reduces concurrency to interleaving, something we avoid in our logic of steps. In their models they provide for each concurrent action a 'multi-dimensional cube' of interleavings of the constituent actions. Since the 'cubes of interleavings' have no independent relevance because they always accompany simultaneous actions, we think it should be possible to eliminate them and accomplish the desired entailment properties through adaptation of the definition of evaluation of formulas in states.

4 MODL, A MULTI-OBJECT GENERALIZATION OF DYNAMIC LOGIC

Syntax and semantics of Dynamic Logic can be thought of as consisting of two layers: the layer of processes, which are interpreted as a reachability relation over states, and a layer of well formed formulas (containing processes as sub-formulas) that are

evaluated in states of a Kripke model. In MODL, we add an extra layer of steps beneath the layer of processes. So the level of steps in this logic is conceptually and technically distinguished from the level of processes (programs in DL).

4.1 Syntax and Semantics

In the following definition of the syntax of MODL the extra layer we add to DL is easily recognized: it is layer 3 which defines the syntax of step symbols σ.

Definition 1 *Given a finite set A of action symbols and a finite set P of proposition symbols, a* ***well formed formula*** *(ϕ), with $a \in A$ and $p \in P$ is defined as follows:*

$$\begin{array}{lcl} \phi & ::= & p \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \phi' \mid \langle\alpha\rangle\phi \\ \alpha & ::= & \sigma \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^{+} \mid \alpha^{inv} \mid \phi? \\ \sigma & ::= & a \mid any \mid \sigma \,\&\, \sigma' \mid \sim\sigma \end{array}$$

We call an element $a \in A$ an action and elements α processes. The intuitive meaning of any is 'execution of any action or composition of actions'. The intuitive meaning of & is 'simultaneous execution' and of $\sim$ 'execution of any action (step) but this one'. As in DL, $\cup$ stands for 'choice', ; for sequention, $^{+}$ for (non reflexive) iteration, ^{inv} for 'inverse' and ? for 'test'. In addition to the usual ones, we apply the following syntactic abbreviations:

$$\begin{array}{llll} \sim(\sim\sigma \,\&\, \sim\sigma') & \text{to} & \sigma + \sigma' & \qquad \sim\sigma + \sigma' \qquad\qquad \text{to} \quad \sigma \Rightarrow \sigma' \\ \phi \vee \langle\alpha^{+}\rangle\phi & \text{to} & \langle\alpha^{*}\rangle\phi & \qquad (\sim\sigma \Rightarrow \sigma') \,\&\, (\sim\sigma' \Rightarrow \sigma) \quad \text{to} \quad \sigma \Leftrightarrow \sigma' \end{array}$$

The intuitive meaning of $+$ is 'choice between steps', of $\Rightarrow$ 'step (action) implication' and of $\Leftrightarrow$ 'step equivalence'.

Definition 2 *Given a finite set P of proposition symbols and a finite set A of action symbols, a MODL-structure $\mathcal{S} = (S, \mathcal{I}_P, \mathcal{R}_{2^A})$ is defined as follows:*

- *S is a nonempty set of possible states*
- *$\mathcal{I}_P$ is a total function $P \to 2^S$*
- *$\mathcal{R}_{2^A}$ is a set $\{R_x | x \in 2^A\}$ of mutually non-intersecting relations over states*

The difference with standard PDL-structures is that relations over states S are indexed with sets of action symbols in stead of with individual action symbols. This is done to meet property P3 of the foregoing section. The requirement of non-intersection of the reachability relations is introduced to meet property P4. After we have defined the semantics of MODL we will sketch a proof for the assertion that dropping the requirement of non-intersection only introduces redundancy in models and does not affect logical entailment properties.

As the syntax, the semantics is also levelled.

Definition 3 *Given a finite set P of proposition symbols and a finite set A of action symbols and a MODL-structure $\mathcal{S} = (S, \mathcal{I}_P, \mathcal{R}_{2^A})$, the interpretation of a step σ*

denoted by $\mathcal{I}_s(\sigma)$, the interpretation of a process α denoted by $\mathcal{I}_p(\alpha)$ and validity of a wff ϕ in a state s of a structure $\mathcal{S}$ denoted by $\mathcal{S}, s \models \phi$ are simultaneously defined by ($\models_{PL}$ stands for propositional satisfyability):

$$\mathcal{I}_s(\sigma) \equiv \{(s,s') \mid (s,s') \in R_x,\ x \in 2^A \text{ and } x \models_{PL} \sigma\}$$

$$\begin{array}{lcllcl}
\mathcal{I}_p(\sigma) & \equiv & \mathcal{I}_s(\sigma) & \mathcal{I}_p(\alpha \cup \beta) & \equiv & \mathcal{I}_p(\alpha) \cup \mathcal{I}_p(\beta) \\
\mathcal{I}_p(\alpha;\beta) & \equiv & \mathcal{I}_p(\alpha) \circ \mathcal{I}_p(\beta) & \mathcal{I}_p(\alpha^+) & \equiv & (\mathcal{I}_p(\alpha))^+ \\
 & & & \mathcal{I}_p(\phi?) & \equiv & \{(s,s) \mid \mathcal{S}, s \models \phi\}
\end{array}$$

$$\begin{array}{lcllcl}
\mathcal{S}, s \models \bot & & \textit{never} & \mathcal{S}, s \models \top & & \textit{always} \\
\mathcal{S}, s \models p & \textit{iff} & s \in \mathcal{I}_P(p) & \mathcal{S}, s \models \phi \vee \psi & \textit{iff} & \mathcal{S}, s \models \phi \textit{ or } \mathcal{S}, s \models \psi \\
\mathcal{S}, s \models \neg\phi & \textit{iff} & \textit{not } \mathcal{S}, s \models \phi & \mathcal{S}, s \models \langle\alpha\rangle\phi & \textit{iff} & \textit{for some } s' \in S \textit{ holds} \\
 & & & & & (s,s') \in \mathcal{I}_p(\alpha) \textit{ and } \mathcal{S}, s' \models \phi
\end{array}$$

The interpretation $\mathcal{I}_s(\sigma)$ interprets step-formulas σ as a reachability relation over states. This is accomplished by seeing a formula σ as a propositional logic formula (by associating $\&, \sim, any$ with respectively $\wedge, \neg, \top$), and seeing the labels (sets of action symbols) of relations over states as propositional models. Now σ is interpreted by those relational elements that have a label that propositionally satisfies it. This defines the semantics of the operators $\&, \sim$ and any and the operators that are introduced by syntactical abbreviations. For instance $\langle a \ \& \ \sim b\rangle\top$ means 'there is a transition (element of a relation) that among the elements of its label has a but not b'. It is not difficult to see that the semantics of $\&$ obeys properties P1 and P2.

Now we formulate the assertion that demanding non-intersection of MODL-structures does not restrict us in any way.

Proposition 1 *Dropping the requirement of non-intersection of $\mathcal{R}_x$ for different $x \in 2^A$ does not affect entailment properties.*

To prove this we can follow an approach taken by Van der Hoek [22] to prove that intersection is not modally definable. Basically it comes down to the observation that every 'intersecting' structure can be transformed to a structure that bisimulates with a non-intersecting one so that precisely the same set of MODL-formulas holds in both structures. The transformation just makes copies of worlds in a way that eliminates intersection. Copies of worlds have identical valuations.

4.2 Action connectives

The semantics of the action (step) negation in MODL is such that the transitions satisfying $\sim\sigma$ are the complement of the transitions satisfying σ. This means that negation of a step σ (action a) means "all steps (actions) different from σ (a)". This notion of action negation allows compact expression of frame properties. The following formula expresses 'σ is the only step that may make ϕ true:

$$\neg\phi \rightarrow \neg\langle\sim\sigma\rangle\phi$$

If σ is just a, we can interpret this as bringing about ϕ can only be accomplished by a step that includes the action a. It does not state that other actions should not be involved as well. But other actions can only be involved if they are performed simultaneously with a. In a way that resembles Reiters 'solution' to the frame problem [16] [2] we can 'collect' the set $\{a, b, c, \ldots\}$ of actions that influence a predicate P. Now we can express that **only** these actions change the predicate with the formula

$$\neg P \rightarrow \neg\langle\sim(a + b + c + \ldots)\rangle P$$

In section 5 we will need this way of expressing frame properties in our translation of graphical specifications to MODL-formulas.

MODL also includes an intuitive notion of action (step) implication. The assertion that step σ implies step σ' is expressed by the formula:

$$[\sim(\sigma \Rightarrow \sigma')]\bot$$

The formula states that all actions for which it does not hold that if you do a σ, you also do a σ', lead to a falsum, which is to say that such a step cannot occur. This expresses that performing a transition σ implies doing σ' at the same time (simultaneously). Because this is a rather clumsy formula for a very often used concept we define the abbreviation $\sigma >> \sigma'$ for it. In the example in section 5 this action implication will be related to the event-triggering concept of graphical specification languages. The following conditional action implication formula helps to understand the intuitive meaning of the action (step) implication defined.

$$power\_supply\_ok \rightarrow (press\_button >> light\_turns\_on)$$

5 APPLICATION TO AN EXAMPLE

We will now show how MODL can be used to give semantics and reasoning capacity to graphical specification languages for specifying multi-object reactive systems. We will do this by giving a simple example of a railroad crossing control system. The example is taken from an article by Shaw [21]. In the graphical language the behavior of the objects is defined with the help of State Transition Diagrams (STD's). Communication between objects is realized by an event-triggering mechanism. We will translate the graphical specification in a systematic way into formulas of MODL. As postconditions, we will allow first order formulas. This is not in correspondence with the propositional language explored so far, but using a first order language throughout all of this paper would have obscured the central ideas by introducing lots of technical complications.

The three Mealy style transition diagrams of figure 2 specify the behavior of the three context objects of the controller, an entry and exit sensor and a gate. The two

sensors are mounted at the end points of a rail segment in which the gate is located. When there is a train in the segment, the gate should be closed, otherwise, it should be opened. Figure 3 shows the Mealy diagrams of the two controller components. The monitor object counts the number of trains in the segment, and the gate controller opens and closes the gate. Transition labels have the form $trigger[guard]/actions$, where the trigger is a triggering event, the guard an enabling condition that must be true for the transition to be enabled, and the actions are a finite set of actions that are performed when the transition is taken.

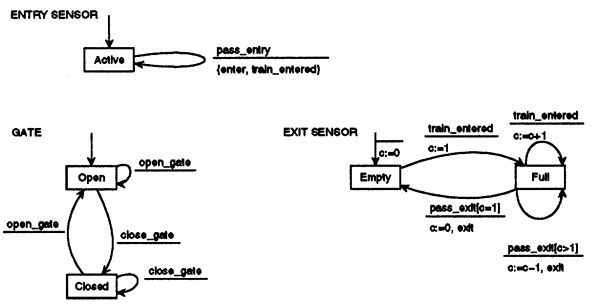

Figure 2. Objects forming the context of the railway crossing controller.

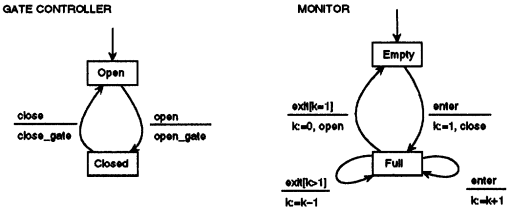

Figure 3. Objects of the railway crossing controller.

We translate the STD's for both the context and the controller into formulas of MODL according to the following rules:

- Each STD-state st of an object obj is provided with a state predicate $obj.st$. State predicates provide a means to 'observe' STD-states. We can identify observing that the system is in some STD-state with the assertion that its state predicate holds. We also add static constraints that guarantee that an object can not be in more than one of its STD-states. These static constraints are just boolean combinations of state predicates.

- Each STD-transition to a state st of an object obj is given a unique name trs and is supplied with a formula $[trs]obj.st$. We need to specify these state predicates as postconditions to ensure that the result of taking a step can be 'observed' through the status of the state predicates. The static constraints added in the first rule above assure that no frame formulas are needed to specify the intended effect of STD-transitions on state predicates.

- For each STD-transition **to** a state st of an object obj that has the form $\frac{trigger[cond]}{action1,action2,\ldots}$ we provide a set of axioms $[action1]\varphi 1, [action2]\varphi 2, \ldots,$ where $\varphi 1$ is the effect of $action1$ expressed as a postcondition. For these effects we need to provide frame axioms. These have the form $\neg\phi \rightarrow \neg\langle\sim action\rangle\phi$, as discussed in section 4.2.

- Each STD-transition **from** a state st of an object obj that has the form $\frac{trigger[cond]}{action1,action2,\ldots}$ is translated into a MODL-formula:

  $$cond \wedge obj.st \rightarrow (trigger >> (action1 \& action2 \& \ldots)),$$

  that expresses a conditional action implication. By doing so we identify the event-triggering mechanism of the STD's with the concept of conditional action implication in MODL. Note that we formulate a guard here as a sufficient condition for enabling a transition.

The most interesting idea behind the translation that is defined by the above rules is to identify the event triggering mechanism of the STD's with action implication in MODL. Earlier [24] we defined a semantics for UML-statecharts that dealt with communication by means of signal predicates.

Applying the above rules to the 5 objects results in a set of over 35 axioms. Instead of writing down all these axioms, we think it gives more insight to present the unique MODL-structure that satisfies these axioms as is done in figure 4.

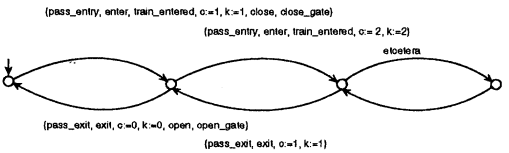

Figure 4. The MODL-structure for the control system and its environment.

Note that in practice there will be a limit to the number of trains that can be in the segment of the railway between the entry and the exit sensor. This implies that the repetition in the above model will not go on indefinitely.

Having characterized in MODL-formulas the unique structure that determines our semantics for STD's, we can legitimately ask whether properties (expressable in MODL) are entailed by the specification. In particular we are interested in safety and liveness properties possibly. Safety properties can be formulated as assertions about all individual states in the model of figure 4. We give one example safety property.

- If the gate is open, there is no train in the segment between entry and exit sensor: $G.open \rightarrow c = 0$

Liveness properties can be formulated as assertions about how the states of the model in figure 4 are related. Liveness properties are typically stated in temporal logics. We will give one example of a liveness property, stated in the branching time temporal logic CTL [4].

- Under the assumption that the segment eventually will be empty, it holds that if the gate is closed, eventually it will be opened again. In CTL: $\forall F(c = 0) \wedge G.closed \rightarrow \forall F(G.open)$.

In the next section we will discuss to what extend MODL is capable of expressing temporal properties such as the ones expressable in CTL.

6 EXPRESSING TEMPORAL PROPERTIES IN MODL

To express temporal properties we need to abstract form actions, which can be achieved by using the any-construct. We make a comparison with the branching time temporal logic CTL [4]. The temporal operators of CTL can all be expressed in terms of four basic ones: $EX\phi$, $AX\phi$, $E(\phi U_s^* \psi)$ and $A(\phi U_s^* \psi)$ (The subscript s denotes that in CTL we have a strong until, and the superscript * denotes that the until in CTL is reflexive). We show how we can translate the first three into a MODL-formula.

$$\begin{array}{lcl} EX\phi & \equiv & \langle any \rangle \phi \\ AX\phi & \equiv & \neg\langle any \rangle \neg\phi \text{ or } [any]\phi \\ E(\phi U_s^* \psi) & \equiv & \langle (\phi?; any)^* \rangle \psi \end{array}$$

The fourth basic temporal operator of CTL, $A(\phi U_s^* \psi)$, is not translatable to an equivalent MODL-formula.

Proposition 2 *The CTL-formula $A(\phi U_s^* \psi)$ is not expressable in MODL.*

We do not provide a proof here, because of space limitations. Although we can not express the CTL formula $A(\phi U_s^* \psi)$, we can express the weak version of this formula: $A(\phi U_w^* \psi)$.

$$A(\phi U_w^* \psi) \equiv [(\neg\psi?; any)^*]\phi$$

This raises the conjecture that we might be able to express weak-CTL, which we define to be the fragment of CTL with only weak until operators. The basic operators of weak-CTL are: $EX\phi$, $AX\phi$, $E(\phi U_w^* \psi)$ and $A(\phi U_w^* \psi)$. But in MODL we can not express the formula $E(\phi U_w^* \psi)$.

Proposition 3 *The weak CTL-formula $E(\phi U_w^* \psi)$ is not expressable in MODL.*

Again we omit the proof. Conversely weak-CTL can not express the property $E(\phi U_s^* \psi)$, which is expressable in MODL. In MODL we can also express mixed temporal/dynamic properties. To express these properties we do not need, and do not want to abstract from actions. Scheerder and Wieringa define a mixture of Temporal and Dynamic Logic called TDL [19]. The basic operators of TDL all have the form: $x\ U_w\ y$, where x and y refer to actions or to conditions in states. The intuition behind TDL's definitions is that $x\ U_w\ y$ holds in a state s if along each path starting in s, x is ensured until we reach an y. All basic operators of TDL can be defined in terms of MODL. Below we show how one of the TDL-operators is defined in MODL.

Definition 4

$$\phi \; U_w^* \; \sigma \quad \equiv \quad [(\sim\sigma)^*]\phi$$

The TDL formula $\phi \; U_w^* \; \sigma$ states that 'on all paths, ϕ holds at least until an action (step) σ is done'. The MODL-formula $[(\sim\sigma)^*]\phi$ says exactly the same thing. It can be read as 'proceding through all actions that are not σ, ϕ holds all the time along the way, and when eventually an action σ is performed, ϕ is not required to hold anymore'.

7 CONCLUDING REMARKS

In this article we mainly accomplished three things: we defined the logic MODL, we showed how to express action implication, frame properties, temporal and mixed temporal/dynamic properties in it, and we showed how MODL can be applied to give semantics and reasoning capacity to graphical specification languages for communicating multi-object systems. In the translation from graphical specifications to MODL-specifications, it was shown that communication of objects can be suitably identified with the notion of action implication as present in MODL. The main purpose of the example was to show how the logic can be applied. Furthermore, the example supports our conjecture that MODL (or MODL-like logics) might prove very useful as a logic underlying Object Oriented system specification in general. In the near future we intend to explore the different MODL-translations of statecharts that may exist as counterparts of the several semantics [1] that are defined for them. Another direction of research is the definition of a suitable notion of **intended model** [3], for MODL-specifications.

References

[1] M. von der Beeck. A comparison of Statecharts variants. In H. Langmaack, W.P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 128–148. Springer, 1994. Lecture Notes in Computer Science 863.

[2] A. Borgida, J. Mylopoulos, and R. Reiter. On the frame problem in procedure specifications. *IEEE Transactions on Software Engineering*, 21:785–798, 1995.

[3] J.M. Broersen, R.J. Wieringa, and R.B. Feenstra. Minimal Semantics for Action Specifications in PDL. In Joeri Engelfriet and Martijn Spaan, editors, *Proceedings Accolade '96*, pages 15–30, Department of Mathematics and Computer Science, University of Amsterdam, 1996. Dutch Graduate School in Logic.

[4] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), April 1986.

[5] H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for specifying concurrent information systems. In J. Chomicki and G. Saake, editors, *Logic for Databases and Information Systems*. Kluwer Academic Publishers, 1997. In print.

[6] J. Fiadeiro and T. Maibaum. Temporal theories as modularisation units for concurrent system specifications. *Formal Aspects of Computing*, 4:239–272, 1992.

[7] Giuseppe De Giacomo and Xiao Jun Chen. Reasoning about Nondeterministic and Concurrent Actions: A Process Algebra Approach. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)* , pages 658–663. AAAI-Press/MIT-Press, 1996.

[8] Giuseppe De Giacomo and Maurizio Lenzerini. PDL-based framework for reasoning about actions. In *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence (AI*IA'95)*, Lecture Notes in Artificial Intelligence 992, pages 103–114. Springer-Verlag, 1995.

[9] D. Harel. *First Order Dynamic Logic*. Springer, 1979. Lecture Notes in Computer Science 68.

[10] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987. Preliminary version appeared as Technical Report CS 84-05, The Weizmann Institute of Science, Rehovot, Israel, February 1984.

[11] P. Jeremaes, S. Khosla, and T.S.E. Maibaum. A modal (action) logic for requirements specification. In D. Barnes and P. Brown, editors, *Software Engineering 86*, pages 278–294. Peter Peregrinus Ltd., 1986.

[12] K. Lodaya, R. Parikh, R. Ramanujan, and P.S. Thiagarajan. A logical study of distributed transition systems. *Information and Computation*, 119:91–118, 1995.

[13] D. Peleg. Communication in concurrent dynamic logic. *Journal of Computer and System Sciences*, 35:23–58, 1987.

[14] D. Peleg. Concurrent dynamic logic. *Journal of the ACM*, 34:450–479, 1987.

[15] Rational. *Unified Modeling Language: Notation Guide, Version 1.1*. Rational Software Corporation, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, 1 September 1997. URL http://www.rational.com/uml/1.1/.

[16] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: P apers in Honor of John McCarthy*. Academic Press, 1991.

[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, 1991.

[18] M. Ryan, J. Fiadeiro, and T. Maibaum. Sharing actions and attributes in modal action logic. In T. Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 569–593. Springer, 1991. Lecture Notes in Computer Science 526.

[19] J. Scheerder and R.J. Wieringa. A modal temporal dynamic logic – doing the deadline. Technical Report IR-433, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, 1997.

[20] A. Sernadas, C. Sernadas, and J. F. Costa. Object specification logic. *Journal of Logic and Computation*, 5(5):603–630, 1995. Available as Research Report since 1992.

[21] A.C. Shaw. Communicating real-time state machines. *IEEE Transactions on Software Engineering*, 18(9):805–816, September 1992.

[22] Wiebe van der Hoek. *Modalities for reasoning about Knowledge and Quantities.* PhD thesis, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, 1992. PHD-thesis.

[23] R.J. Wieringa. A formalization of objects using equational dynamic logic. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *2nd International Conference on Deductive and Object-Oriented Databases (DOOD'91)*, pages 431–452. Springer, 1991. Lecture Notes in Computer Science 566.

[24] R.J. Wieringa and J.M. Broersen. Minimal transition system semantics for lightweight class- and behavior diagrams. Technical Report TUM-I9803, Institut fur informatik, technische universitat munchen, 1998. Proceedings PSMT-Workshop om Precise Semantics for Software Modeling Techniques.

[25] R.J. Wieringa, W. de Jonge, and P.A. Spruit. Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems*, 1(1):61–83, 1995.

[26] R.J. Wieringa and J.-J.Ch. Meyer. Actors, actions, and initiative in normative system specificati on. *Annals of Mathematics and Artificial Intelligence*, 7:289–346, 1993.