A Logic-Based Approach for Real-Time Object-Oriented Software Development

Fernando Náufel do Amaral and Edward Hermann Haeusler *

Abstract

This paper discusses how RETOOL, an action logic featuring an operator that expresses necessary conditions, postconditions and time bounds of actions, can be combined with MTL, a linear-time temporal logic with time-bounded operators, to reason about general properties of timed transition systems, an abstract model for the behavior of objects in a real-time, object-oriented software system. The parallel composition of such objects, which can be modeled as a colimit in the category of RETOOL theories, is also discussed.

Keywords: Modal Logic in Computing; Action Logic; Timed Transition Systems; Metric Temporal Logic; Category Theory in Concurrency

*Department of Informatics
PUC-RJ (Catholic University of Rio de Janeiro) e-mail: {fnaufel, hermann}@inf.puc-rio.br
This is work developed as part of the MEFIA Project – CNPq/NSF project no. 68.0089/99-3

1. Introduction

This paper is inserted in the context of an environment for real-time object-oriented software development based on Formal Methods. The framework depicted here has been considered for use at a research laboratory at PUC-RJ in the development of the prototype of a CASE environment for an industrial partner in the field of telecommunications ([HHMF98]).

The basic architecture of the environment consists of a set of tools implemented over an integration platform (currently CORBA).

Conceptually, the environment consists of several planes that relate the tools to various aspects of functionality. It has a user plane, a formal plane and an implementation plane. In the user plane, software designers model applications and libraries via class relationships, inter-object communication and state diagrams. The user plane can be used to produce a so-called *scenario* (i.e., fixing parameters in the design to specific values), which is then used by a model checker (SMV – Symbolic Model Verifier [CMCH96]) for verifying properties of the design. The formal plane is also a target for a mapping from designs to the action logic RETOOL ([Ama00, CFH97, FH98]) in order to support general reasoning about the design, as opposed to verification and validation tasks. The implementation plane is responsible for deriving code, through transformation rules, in a high level programming language (DDL – see [Car96]) associated with the diagrammatic class notation of the environment. Finally, executable code is generated from the DDL code, again through transformation rules.

The focus of this paper is on the support that the formal plane was designed to provide for object-oriented development. More specifically, the paper concentrates on the way the logic RETOOL allows reasoning about the enabling conditions, postconditions and time bounds of the actions involved in the design. The version of RETOOL presented here is sound and complete, as opposed to the tentative axiomatizations of [CFH97] and [FH98].

The paper is structured as follows: section 2. discusses the action logic RETOOL and its combination with a linear-time temporal logic (MTL – see [Cha95]) to abstract temporal properties of actions; section 3. introduces the concrete model of Timed Action Transition Diagrams, used to represent designs; section 4. provides an example of the use of these formalisms to reason about a specific design; finally, section 5. discusses, also through an example, how RETOOL theories can be combined to generate a theory that describes the parallel composition of the objects described by the original theories.

2. The Logics

This section presents the action logic RETOOL in detail and discusses how it can be combined with a temporal logic (MTL) to abstract temporal properties of actions.[†]

2.1 RETOOL: the Language

The primitive syntactic entities of RETOOL are *attribute symbols* (which, in this propositional version of the logic, are simple propositional letters), and *action symbols*. We denote the (non-empty) set of attribute symbols as A, and the (non-empty) set of action symbols as Γ.

The logic presupposes an infinite totally ordered set (TIME, $\leq$), with minimum 0. A constant ∞ is available, such that $\infty \notin$ TIME and $t \leq \infty, \forall t \in$ TIME.[‡] The other syntactic categories are:

- State Propositions (SP): $p ::= a \mid \neg p \mid p \rightarrow p'$, where $a \in A$;
- Action terms (AT): $t ::= g \mid p_l \delta^u q$, where $g \in \Gamma$, $p, q \in SP$, $l \in$ TIME, $u \in$ TIME $\cup \{\infty\}$, and $l \leq u$;
- Time terms (TT): $T ::= l(t) \mid u(t) \mid \infty \mid 0 \mid 1 \mid 2 \mid ...$
- Formulae: $\phi ::= a \mid t_1 \supset t_2 \mid \neg\phi \mid \phi \rightarrow \phi' \mid [t]\phi \mid [\ ]p \mid T_1 \leq T_2$, where $a \in A$, $t, t_1, t_2 \in AT$, $p \in SP$, and $T_1, T_2 \in TT$.

It should be noted that, in the definition of time terms, l and u are *not* function symbols; in fact, as will be seen in a later section, $l(t)$ and $u(t)$, for t an action term, are mere abbreviations of constant symbols that denote elements of TIME $\cup \{\infty\}$.

2.2 RETOOL: the Semantics

The semantics of RETOOL is defined over structures that are based on the notion of *timed transition systems* [HMP92]: given a set A of attribute symbols and a set Γ of action symbols, a *timed frame* $\mathcal{F}$ for A and Γ is a sextuple $(W, \rightarrow, l, u, I, w_0)$, where

- W is a set of states;
- For each $g \in \Gamma$, $\xrightarrow{g} \subseteq W \times W$ is the transition relation for action g;
- l maps each $g \in \Gamma$ to an element $l(g) \in$ TIME;

[†] In order to make the presentation clearer, the propositional versions of the logics are used, but the reasoning can easily be extended to a first-order context, with typed variables as attributes and arbitrary assignments as actions.

[‡] For our purposes, time can be modelled by the set of natural numbers and the corresponding $\leq$ relation.

- u maps each $g \in \Gamma$ to an element $u(g) \in$ TIME $\cup\{\infty\}$ such that $u(g) \geq l(g)$;
- $I : A \rightarrow 2^W$ is an interpretation of the attributes, where each $a \in A$ is assigned the set of worlds where a is true;
- w_0 is the initial state.

Every action $g \in \Gamma$ has a lower bound $l(g)$ and an upper bound $u(g)$. Intuitively, the lower bound defines the minimum delay that must be observed for the transition to take place (provided all the necessary conditions for the occurrence of such a transition are satisfied). The upper bound defines the maximum delay during which the transition must occur (again, provided all necessary conditions are satisfied). Formally, lower and upper bounds are defined through the use of the notion of *computation*:

A *timed state sequence* [HMP92] for a timed frame is a pair $\rho =< \sigma, \mathrm{T} >$, where σ is an infinite sequence of states ($\sigma_i \in W$) and T is an infinite sequence of corresponding times ($\mathrm{T}_i \in$ TIME), satisfying:

- monotonicity: for all $i \geq 0$, either $\mathrm{T}_{i+1} = \mathrm{T}_i$, or $\mathrm{T}_{i+1} > \mathrm{T}_i$ and $\sigma_{i+1} = \sigma_i$.
- progress: for every $t \in$ TIME, there is $i \geq 0$ such that $\mathrm{T}_i \geq t$.

A *computation* [HMP92] over a timed frame is a timed state sequence $< \sigma, \mathrm{T} >$ such that

- σ is a computation of the underlying transition system, i.e., for every $i \geq 0$, there is a transition $\sigma(i)$ such that $\sigma_i \stackrel{\sigma(i)}{\rightarrow} \sigma_{i+1}$;
- (lower bound): for every $i \geq 0$ in the domain of σ, there is a $j \leq i$ such that $\mathrm{T}_i - \mathrm{T}_j > l(\sigma(i))$ and $\sigma(i)$ is enabled in every state σ_k for $j \leq k \leq i$.
- (upper bound): for every $g \in \Gamma$ and $i \geq 0$, there is $j \geq i$ with $\mathrm{T}_j - \mathrm{T}_i \leq u(g)$ such that either g is not enabled at σ_j or $g = \sigma(j)$.

The denotation of a state proposition p in a timed frame $\mathcal{F}$ is the set of states defined as follows:

- $[\![a]\!] = I(a)$;
- $[\![\neg p]\!]^{\mathcal{F}} = W \setminus [\![p]\!]^{\mathcal{F}}$;
- $[\![p \rightarrow p']\!]^{\mathcal{F}} = (W \setminus [\![p]\!]^{\mathcal{F}}) \cup [\![p']\!]^{\mathcal{F}}$.

The denotation of an action term t in a timed frame $\mathcal{F}$ is the set of transitions defined as follows (where $en(g)$ is the set of states where g is enabled):

- $[\![g]\!]^{\mathcal{F}} = \{(w, w') \mid w \xrightarrow{g} w'\}$;
- $[\![p_l \delta^u q]\!]^{\mathcal{F}} = \{(w, w') \mid \exists g \in \Gamma \;\; [(w \xrightarrow{g} w') \;\; \wedge \;\; en(g) \subseteq [\![p]\!]^{\mathcal{F}} \;\; \wedge$
  $\forall v, v'((v \xrightarrow{g} v') \Rightarrow (v' \in [\![q]\!]^{\mathcal{F}})) \;\; \wedge \;\; (l \leq l(g) \leq u(g) \leq u)]\}$

Note that the denotation of an action term built with the δ operator – $p_l \delta^u q$ – is the set of all transitions labeled by actions whose necessary condition is p, whose postcondition is q, and whose time limits are l and u.

The denotation of a time term is an element of TIME $\cup \{\infty\}$: we have included in our language one constant symbol for each element of TIME, as well as a constant symbol for ∞. For each action symbol $g \in \Gamma$, $l(g)$ and $u(g)$ denote the time bounds of g, which are extra-logical information (hence, $l(g)$ and $u(g)$ can also be seen as mere constant symbols denoting elements of TIME $\cup \{\infty\}$). For action terms of the form $p_x \delta^y q$, the time terms $l(p_x \delta^y q)$ and $u(p_x \delta^y q)$ can be seen as mere abbreviations of x and y, respectively.

Finally, the satisfaction of a formula by a timed frame $\mathcal{F}$ at a state w is defined by:

- $\mathcal{F}, w \models p$ iff $w \in [\![p]\!]^{\mathcal{F}}$;
- $\mathcal{F}, w \models (t_1 \supset t_2)$ iff $[\![t_1]\!]^{\mathcal{F}} \subseteq [\![t_2]\!]^{\mathcal{F}}$;
- $\mathcal{F}, w \models \neg\phi$ iff not $\mathcal{F}, w \models \phi$;
- $\mathcal{F}, w \models \phi \rightarrow \phi'$ iff $\mathcal{F}, w \models \phi$ implies $\mathcal{F}, w \models \phi'$;
- $\mathcal{F}, w \models [t]\phi$ iff $\mathcal{F}, w' \models \phi$ for every w' such that $(w, w') \in [\![t]\!]^{\mathcal{F}}$;
- $\mathcal{F}, w \models [\,]p$ iff $\mathcal{F}, w_0 \models p$;
- $\mathcal{F}, w \models T_1 \leq T_2$ iff $[\![T_1]\!]^{TIME} \leq [\![T_2]\!]^{TIME}$.

"$\supset$" is the *subsumption* operator. To say that action term t_1 subsumes action term t_2 is to say that every action denoted by t_1 is also denoted by t_2 (but not necessarily the other way around). Subsumption can be seen as a refinement on actions: the actions denoted by t_1 refine those denoted by t_2.

2.3 An Axiomatization for RETOOL

The axiom schemes and rules of inference in Figure 1 comprise an adequate axiomatization of RETOOL. We assume given an adequate calculus for deriving properties involving members of TIME $\cup \{\infty\}$ and the relation $\leq$.

In the axiomatization, Λ represents a set of RETOOL formulae, the derivability relation $\vdash$ is defined in the usual manner, and *enabled*(t) is an abbreviation for the

(PC) All axioms and rules of propositional calculus

(K) $[t](\phi \to \psi) \to ([t]\phi \to [t]\psi)$
$[\,](p \to q) \to ([\,]p \to [\,]q)$

(I) $[\,]p \to [t][\,]p$
$[t][\,]p \to (enabled(t) \to [\,]p)$
$[\,]\neg p \leftrightarrow \neg[\,]p$

(N) $\dfrac{\Lambda \vdash \phi}{\Lambda \vdash [t]\phi} \quad \dfrac{\Lambda \vdash p}{\Lambda \vdash [\,]p}$

(δ) $\dfrac{\Lambda \vdash enabled(t) \to p \quad \Lambda \vdash [t]q \quad \Lambda \vdash l \leq l(t) \leq u(t) \leq u}{\Lambda \vdash t \supset p_l \delta^u q}$

(S1) $t \supset t$

(S2) $(t_1 \supset t_2) \to ((t_2 \supset t_3) \to (t_1 \supset t_3))$

(S3) $(t_1 \supset t_2) \to ([t_2]\phi \to [t_1]\phi)$

(NC) $(t \supset p_l \delta^u q) \to (enabled(t) \to p)$

(Post) $(t \supset p_l \delta^u q) \to ([t]q)$

(Bounds) $(t \supset p_l \delta^u q) \to (enabled(t) \to l \leq l(t) \leq u(t) \leq u)$

Figure 1: An axiomatization for RETOOL

formula $\neg[t]\bot$ (which is true in a given state w iff there is at least one transition leaving w labeled by an action in the denotation of t).

The soundness and completeness of this axiomatization is proved in detail in [Ama00]. It should be noted that we employ the following notion of consequence, or entailment: a set Λ of formulae entails a formula ϕ (written $\Lambda \models \phi$) if and only if every timed frame which satisfies Λ *at all states* also satisfies ϕ *at all states*. This notion of consequence is called "global consequence" in [Har84]. It is a weaker notion than the alternative "local" definition of entailment (which stipulates that $\Lambda \models \phi$ if and only if every state which satisfies Λ also satisfies ϕ). For our purposes, it is reasonable to employ the notion of global consequence because, in constructing a logical description Λ of a reactive system S, we expect the formulas of Λ to hold at all states of the timed frame which represents S.

2.4 MTL

MTL (see [Cha95]) is a linear-time temporal logic with time-bounded operators. Its models are computations of timed transition systems. We extend the language of MTL with the action terms of RETOOL taken as propositions (with the intended meaning that an action term t is true at a given point σ_i of a computation iff the denotation of t contains the action responsible for the transition from σ_i to σ_{i+1}). We also add subsumption formulae of the form $t_1 \supset t_2$ to MTL, yielding the following language over a set A of attribute symbols, a set Γ of action symbols, and order (TIME, $\leq$) as defined for RETOOL:

$$\tau ::= a \mid t \mid t_1 \supset t_2 \mid \neg\tau \mid \tau \rightarrow \tau' \mid \mathbf{X}_{Rc}\,\tau \mid \tau_1 \mathbf{U}_{Rc}\,\tau_2$$

where $a \in A$, and $t, t_1, t_2 \in AT$, $c \in$ TIME $\cup\{\infty\}$, and R is a relation on TIME $\cup\{\infty\}$.

The semantics of this language is defined over a computation $<\sigma, \mathrm{T}>$ as follows:

- $\sigma_i, T_i \models a$ iff $\sigma_i \in I(a)$;
- $\sigma_i, T_i \models t$ iff $\sigma(i) = t$;
- $\sigma_i, T_i \models t_1 \supset t_2$ iff $[\![t_1]\!] \subseteq [\![t_2]\!]$;
- $\sigma_i, T_i \models \neg\tau$ iff not $\sigma_i, T_i \models \tau$;
- $\sigma_i, T_i \models \tau \rightarrow \tau'$ iff $\sigma_i, T_i \models \tau$ implies $\sigma_i, T_i \models \tau'$;
- $\sigma_i, T_i \models \mathbf{X}_{Rc}\,\tau$ iff $\sigma_i, T_{i+1} \models \tau$ and $(T_{i+1} - T_i)\ R\ c$;
- $\sigma_i, T_i \models \tau_1 \mathbf{U}_{Rc}\ \tau_2$ iff, for some $k \geq i$, it is the case that $\sigma_k, T_k \models \tau_2$ and $(T_k - T_i)\ R\ c$, and, for every j such that $i \leq j < k$, it is the case that $\sigma_j, T_j \models \tau_1$.

The temporal operators are ne**X**t, referring to the next state in the computation, and **U**ntil, which states the existence of a future state in which τ_2 holds and until which τ_1 holds. Notice that these operators are relativized to the intervals determined by the condition *Rc*. Other operators can be defined through abbreviations, such as

- $\mathbf{F}_{Rc}\,\tau = \top\mathbf{U}_{Rc}\,\tau$ (sometime in the future);
- $\mathbf{G}_{Rc}\,\tau = \neg(\mathbf{F}_{Rc}\,(\neg\tau))$ (always in the future).

An axiomatization of MTL can be found in [Cha95]. The proof rules that relate RETOOL and MTL are as follows:

(R1)

$$\frac{\text{enabled}(t) \rightarrow r \qquad t \supset p_0\delta^{\infty}q}{t \;\rightarrow\; r \wedge \mathbf{X}_{=0}q}$$

Rule **(R1)** deals only with change, i.e. with the transitions performed by actions. Therefore, it uses the delta operator with time bounds 0 and ∞. It states that t, when taken, establishes q. Notice that the post-condition is established in the next state, and that the transition does not take time.

(R2)

$$\frac{\{h \supset \top_0\delta^{\infty}(\neg q) \mid h \in \Gamma - g\} \qquad g \supset \top_0\delta^{\infty}q}{\mathbf{X}_{=0}q \;\rightarrow\; g}$$

Rule **(R2)** allows us to infer that an action occurs by observing that a given proposition was set to true when only that action can establish it as a post-condition. In a way, this rule "completes" R1.

(R3)

$$\frac{p \rightarrow \neg\text{enabled}(t) \qquad t \supset \top_x\delta^{\infty}\top}{p \rightarrow \mathbf{G}_{\leq x}\neg t}$$

Rule **(R3)** establishes safety properties by using the lower bound. If p holds and implies that t is not enabled, then we know that at least x units of time have to elapse before t can be taken, where x is a lower bound for t. The temporal operator $\mathbf{G}_{\leq x}$ means "for the next x time units".

(R4)

$$\frac{\{p \rightarrow [h]p \mid h \in \Gamma - g\} \qquad p \rightarrow \mathit{enabled}(g) \qquad g \supset \top_0\delta^{y}\top}{p \;\rightarrow\; \mathbf{F}_{\leq y}g}$$

Rule **(R4)** establishes liveness properties through the use of the upper bound. If p holds and is an invariant for all actions other than g, and p implies that g is enabled, then we know that g will be taken before y units of time, where y is an upper bound for g.

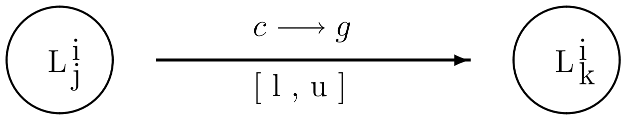

Figure 2: An edge in a TATD

(R5)

$$\frac{\begin{array}{ccc} & \{\Gamma - g\} & \{\Gamma - h\} \\ & | & | \\ \neg g \vee \neg h & \phi & \phi \end{array}}{\phi}$$

Rule **(R5)** asserts that if the same conclusion ϕ is obtained assuming that a certain action g does not happen as well as assuming that a certain other action h does not happen, and, if those actions never happen together, then we obtain the mentioned conclusion. This is the rule that allows deriving conclusions from non-interfering actions.

3. The Concrete Model

In order to allow the software designer to represent a real-time object-oriented system in a manner suitable for formal reasoning, Timed Action Transition Diagrams (TATDs) are introduced as concrete models. This section defines such entities and presents a mapping from TATDs to RETOOL theories.

3.1 Timed Action Transition Diagrams

Given a set A of attribute symbols and a set Γ of action symbols, a Timed Action Transition Diagram (TATD) is a finite directed graph. Each edge in the graph is labelled by a guarded instruction $c \to g$, where $g \in \Gamma$ and c is a state proposition, and by a pair $[l, u]$, where $l \in$ TIME and $u \in$ TIME $\cup\{\infty\}$ and $l \leq u$.

Each node in the graph represents a location of the flow of control of an object in the system. An edge between two locations L_j and L_k of object i is pictorially represented as in Figure 2.

3.2 Mapping TATDs onto RETOOL Theories

In order for the formal plane to reason about the behavior of the system, the TATD corresponding to each object is mapped onto a RETOOL theory. The integration of the behavior of several objects is achieved by means of colimits in the appropriate category of RETOOL theories ($\mathcal{C}_{RETOOL}$), following a well-known approach described by [BLM97] and [FM92] and illustrated in section 5..

Given a TATD over the set A_d of attribute symbols and the set Γ of action symbols, we produce a set of RETOOL formulae over the following attribute symbols:

$$A = A_d \cup \{atL_0, atL_1, ..., atL_{n-1}\}$$

where $L_i, 0 \leq i < n$ are the n locations in the TATD.

The RETOOL theory for a given TATD consists of the following formulae:

$$atL_i \rightarrow (\bigwedge_{i \neq j} \neg atL_j)$$

(At each state, the flow of control is at most in one location.)

$$[\,]((\bigwedge_{0 \leq i < n} \neg atL_i) \wedge \Theta)$$

(The initial state is the one satisfying a set Θ of initial conditions. In the initial state, the flow of control is not yet in any location L_i.)

For each action symbol g labelling edges in the TATD such as the ones in Figure 3, the folowing formulae are produced:

$$\begin{array}{ll} g \supset & ((atL_{i_1} \wedge c_1) \vee (atL_{i_2} \wedge c_2) \vee ... \vee (atL_{i_k} \wedge c_k)) \\ & {}_l\delta^u \\ & (atL_{j_1} \vee atL_{j_2} \vee ... \vee atL_{j_k}) \end{array}$$

$$enabled(g) \rightarrow ((atL_{i_1} \wedge c_1) \vee (atL_{i_2} \wedge c_2) \vee ... \vee (atL_{i_k} \wedge c_k))$$

$$\begin{array}{lcl} atL_{i_1} \wedge c_1 & \rightarrow & [g]atL_{j_1} \\ atL_{i_2} \wedge c_2 & \rightarrow & [g]atL_{j_2} \\ & ... & \\ atL_{i_k} \wedge c_k & \rightarrow & [g]atL_{j_k} \end{array}$$

Finally, the functionality of the actions must be provided through formulae of the form:

$$\begin{array}{lcl} p_1 & \rightarrow & [g]q_1 \\ p_2 & \rightarrow & [g]q_2 \\ ... & & \\ p_n & \rightarrow & [g]q_n \end{array}$$

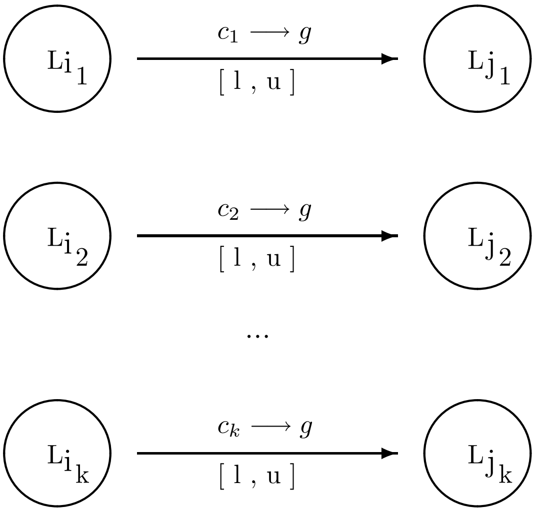

Figure 3: Edges labelled by g in a TATD

4. A Short Example

Consider a machine that can sell cakes and cigars. After it accepts a coin, it is ready to deliver either a cake or a cigar within 10 time units. After the machine delivers the product, it is reset in at most 1 time unit.

The following set of propositional variables will be used for representing the state:

$$OFF,\ ON,\ Waiting,\ DeliveredCake,\ DeliveredCigar$$

The following actions represent the possible activities of the machine:

$$begin,\ coin,\ reset,\ cake,\ cigar$$

The behavior of the machine is specified as follows:

1. $begin \supset OFF\ {}_1\delta^{\infty}\ ON$
2. $coin \supset ON\ {}_1\delta^{\infty}\ Waiting$
3. $cake \supset Waiting\ {}_1\delta^{10}\ DeliveredCake$
4. $cigar \supset Waiting\ {}_1\delta^{10}\ DeliveredCigar$

5. $reset \supset DeliveredCigar \ {}_0\delta^0 \ ON$
6. $reset \supset DeliveredCake \ {}_0\delta^0 \ ON$
7. $OFF \leftrightarrow enabled(begin)$
8. $ON \leftrightarrow enabled(coin)$
9. $Waiting \leftrightarrow enabled(cake)$
10. $Waiting \leftrightarrow enabled(cigar)$
11. $(DeliveredCigar \vee DeliveredCake) \leftrightarrow enabled(reset)$

We omit the axioms that specify that only one propositional variable is true at a time.

The proof rules defined in section 2.4 allow us to derive the following properties:

$$ON \rightarrow \neg\mathbf{F}_{\leq 2}\ (cake) \wedge \neg\mathbf{F}_{\leq 2}\ (cigar) \tag{4.1}$$

$$Waiting \rightarrow \mathbf{F}_{\leq 10}\ (cake \vee cigar) \tag{4.2}$$

In order to prove (4.1), we observe that

$$ON \rightarrow \neg Waiting \ \text{ and } \ \neg Waiting \rightarrow (\neg enabled(cake) \wedge \neg enabled(cigar))$$

Thus, from

$$cake \supset \top_1\delta^\infty\top \ \text{ and } \ cigar \supset \top_1\delta^\infty\top$$

we derive

$$ON \rightarrow \mathbf{G}_{\leq 1}\ \neg cake \ \text{ and } \ ON \rightarrow \mathbf{G}_{\leq 1}\ \neg cigar$$

respectively, using **R3** in both cases. By MTL reasoning, we derive $ON \rightarrow \mathbf{G}_{\leq 2}\ \neg cake$ and $ON \rightarrow \mathbf{G}_{\leq 2}\ \neg cigar$. Recall that $\mathbf{G}_{\leq 2}\ \neg p = \neg\mathbf{F}_{\leq 2}\ p$, by definition.

Let Γ be the set of all actions present in the specification. In order to prove (4.2), first we observe that

$$\forall g \in ((\Gamma - \{cake\}) - \{cigar\}) : Waiting \rightarrow [g]Waiting \tag{4.3}$$

$$Waiting \rightarrow enabled(cigar) \tag{4.4}$$

$$cigar \supset \top_1\delta^{10}\top \tag{4.5}$$

Thus, by applying rule **R4** to (4.3), (4.4), and (4.5), we can conclude

$$Waiting \rightarrow \mathbf{F}_{\leq 10}\ cigar$$

and hence

$$Waiting \rightarrow \mathbf{F}_{\leq 10}\ (cigar \vee cake)$$

Similarly we have

$$\forall g \in ((\Gamma - \{cigar\}) - \{cake\}) : Waiting \rightarrow [g]Waiting \tag{4.6}$$

$$Waiting \rightarrow enabled(cake) \tag{4.7}$$

$$cake \supset \top_1 \delta^{10} \top \tag{4.8}$$

Thus, by applying rule **R4** to (4.6), (4.7), and (4.8), we can conclude

$$Waiting \rightarrow \mathbf{F}_{\leq 10}\ cake$$

and hence

$$Waiting \rightarrow \mathbf{F}_{\leq 10}\ (cigar \vee cake)$$

Thus, by using **R5** and $\neg cake \vee \neg cigar$, we reach the desired conclusion.

5. Composing RETOOL Theories

This paper has contended that the behavior of a real-time reactive system can be represented by a timed action transition diagram or, in a more abstract fashion, by a RETOOL theory. However, the examples and ideas presented so far have been restricted to cases where the behavior of only one object (or process) is described. In order to achieve modularity, a highly desirable feature in software architecture, we would like to be able to apply the same logical framework both to the specification of a single object and to the specification of complex systems formed by several objects. More specifically, we would like to employ RETOOL to reason about the parallel composition of objects.

It is through the use of Category Theory that this goal is to be achieved. [Gog89] established the principle that "given a category of widgets, the operation of putting a system of widgets together to form some super-widget corresponds to taking the colimit of the diagram of widgets that shows how to interconnect them". In our case, the widgets are RETOOL theories, and this section illustrates how this principle can be applied to construct theories describing the behavior of complex systems.[§]

The composition of specifications of processes using Category Theory has also been explored in [BLM97, FM92].

[§]Only elementary notions of Category Theory are used in this section. A basic reference such as [BW90] can provide the necessary definitions.

5.1 $\mathcal{C}_{RETOOL}$ – The Category of RETOOL Theories

First of all, we must define an appropriate category $\mathcal{C}_{RETOOL}$ of RETOOL theories. Objects in this category will be represented by RETOOL theory presentations. A RETOOL theory presentation is a triple (A, Γ, Δ), where A is the set of attribute symbols, Γ is the set of action symbols and Δ is the set of axioms of the theory. We will refer to the pair (A, Γ) as the *signature* of the theory.

Morphisms in $\mathcal{C}_{RETOOL}$ will be defined as signature morphisms which preserve theoremhood. More precisely, given two theory presentations $T_1 = (A_1, \Gamma_1, \Delta_1)$ and $T_2 = (A_2, \Gamma_2, \Delta_2)$, a *signature morphism* σ is a pair $(\sigma_A, \sigma_\Gamma)$ of mappings $\sigma_A : A_1 \longrightarrow A_2$ and $\sigma_\Gamma : \Gamma_1 \longrightarrow \Gamma_2$. In other words, a signature morphism will simply map the attribute symbols of one theory to attribute symbols of the other, and the action symbols of one theory to action symbols of the other. When no ambiguity can arise, we will omit the subscripts of σ. Note that there are no constraints on the definition of these mappings, which can be one-to-one and/or onto. However, not all signature morphisms between the objects of $\mathcal{C}_{RETOOL}$ will be taken as morphisms of the category. We define below which signature morphisms will be the morphisms of $\mathcal{C}_{RETOOL}$.

A signature morphism $\sigma : (A_1, \Gamma_1) \longrightarrow (A_2, \Gamma_2)$ can easily be extended to a mapping of state propositions, action terms and formulae as follows:

- $\sigma(\neg p) = \neg\sigma(p)$
- $\sigma(p \to p') = \sigma(p) \to \sigma(p')$
- $\sigma(p_l \delta^u q) = (\sigma(p))_l \delta^u(\sigma(q))$
- $\sigma(t_1 \supset t_2) = (\sigma(t_1)) \supset (\sigma(t_2))$
- $\sigma(\neg\phi) = \neg\sigma(\phi)$
- $\sigma(\phi \to \phi') = \sigma(\phi) \to \sigma(\phi')$
- $\sigma([t]\phi) = [\sigma(t)]\sigma(\phi)$
- $\sigma([\,]p) = [\,]\sigma(p)$

It is exactly those signature morphisms which preserve theorems that will be taken as the morphisms of $\mathcal{C}_{RETOOL}$. More precisely, given two theories $T_1 = (A_1, \Gamma_1, \Delta_1)$ and $T_2 = (A_2, \Gamma_2, \Delta_2)$, a morphism $\sigma : T_1 \longrightarrow T_2$ is a signature morphism $\sigma : (A_1, \Gamma_1) \longrightarrow (A_2, \Gamma_2)$ such that for all formulae ϕ, if $\Delta_1 \vdash \phi$ then $\Delta_2 \vdash \sigma(\phi)$.

$$\begin{tikzcd} T_1 & \\ T_c \arrow[u, "\sigma_1"] \arrow[r, "\sigma_2"'] & T_2 \end{tikzcd}$$

Figure 4: Component theories T_1 and T_2 and a “communication channel” T_c

5.2 Pushouts in $\mathcal{C}_{RETOOL}$

The categorial mechanism which corresponds to the parallel composition of processes is the *pushout*. Consider two theory presentations T_1 and T_2 which are to be composed together. The details of this composition are determined by a third theory presentation T_c and two morphisms $\sigma_1 : T_c \longrightarrow T_1$ and $\sigma_2 : T_c \longrightarrow T_2$.

T_c can be seen as a “communication channel” connecting certain attributes and actions of T_1 to certain attributes and actions of T_2. In fact, the morphisms σ_1 and σ_2 force the *identification* of certain attribute symbols and action symbols of T_1 with certain attribute symbols and action symbols of T_2. Identified attribute symbols will correspond to shared attributes, and identified action symbols will correspond to synchronized (simultaneous) events.

More formally, if a_c is an attribute symbol of T_c, and the morphisms σ_1 and σ_2 map a_c to attribute symbols a_1 of T_1 and a_2 of T_2, respectively, then this means that a_1 and a_2 correspond to one single, shared attribute of the parallel system represented by $T_1 \,||_{T_c}\, T_2$ (the parallel composition of theory presentations T_1 and T_2 according to channel T_c).

Analogously, if g_c is an action symbol of T_c, and the morphisms σ_1 and σ_2 map g_c to action symbols g_1 of T_1 and g_2 of T_2, respectively, then this means that g_1 and g_2 correspond to one single, joint action of the parallel system represented by $T_1 \,||_{T_c}\, T_2$. This situation is pictorially represented in figure 4.

A pushout of the diagram in figure 4 consists of a theory presentation $T_1 \,||_{T_c}\, T_2$ together with two morphisms $\mu_1 : T_1 \longrightarrow T_1 \,||_{T_c}\, T_2$ and $\mu_2 : T_2 \longrightarrow T_1 \,||_{T_c}\, T_2$ satisfying the following conditions:

1. the diagram of figure 5 commutes; i.e., $\sigma_1; \mu_1 = \sigma_2; \mu_2$.

2. for every other T', μ'_1 and μ'_2 such that the diagram of figure 6 commutes, there is a unique morphism $\mu : T_1 \,||_{T_c}\, T_2 \longrightarrow T'$ such that $\mu_1; \mu = \mu'_1$ and $\mu_2; \mu = \mu'_2$. This situation is shown in figure 7.

$$
\begin{tikzcd}
T_1 \arrow[r, "\mu_1"] & T_1 \; ||_{T_c} \; T_2 \\
T_c \arrow[u, "\sigma_1"] \arrow[r, "\sigma_2"'] & T_2 \arrow[u, "\mu_2"']
\end{tikzcd}
$$

Figure 5: Pushout

$$
\begin{tikzcd}
T_1 \arrow[r, "\mu_1'"] & T' \\
T_c \arrow[u, "\sigma_1"] \arrow[r, "\sigma_2"'] & T_2 \arrow[u, "\mu_2'"']
\end{tikzcd}
$$

Figure 6: Another commutative diagram

$$
\begin{tikzcd}
 & & T' \\
T_1 \arrow[rru, "\mu_1'"] \arrow[r, "\mu_1"] & T_1 \; ||_{T_c} \; T_2 \arrow[ru, dashed, "\mu"] & \\
T_c \arrow[u, "\sigma_1"] \arrow[r, "\sigma_2"'] & T_2 \arrow[u, "\mu_2"'] \arrow[ruu, "\mu_2'"'] &
\end{tikzcd}
$$

Figure 7: Morphism μ is unique

The intuition behind this construction can be summarized as follows: as the morphisms in $\mathcal{C}_{RETOOL}$ mean preservation of theoremhood, commutativity of the diagram in figure 5 means that the (translations of) theorems of T_c specified by σ_1 and σ_2 are preserved by $T_1 \,||_{T_c}\, T_2$ in the same way; i.e., in $T_1 \,||_{T_c}\, T_2$, T_1 and T_2 "share" T_c.

Furthermore, the uniqueness of morphism $\mu : T_1 \,||_{T_c}\, T_2 \longrightarrow T'$ for all T' means that $T_1 \,||_{T_c}\, T_2$ is the "minimal" combination of T_1 and T_2 that respects the interaction specified by T_c, σ_1 and σ_2.

Only the attribute symbols and action symbols which are the images of the symbols in the signature of T_c must be identified in $T_1 \,||_{T_c}\, T_2$. If T_1 and T_2 had other symbols in common, they would be automatically renamed in the construction of the pushout. This automatic "management of namespaces" is another attractive feature of the categorial approach to system composition.

Of course, it remains to be verified that $\mathcal{C}_{RETOOL}$ is finitely cocomplete, i.e., that every finite diagram has a colimit, and hence pushouts will always exist. We do not prove this here.

5.3 A Simple Example

Consider an automobile equipped with an automatic transmission engine. We will take the braking mechanism to be one process, described by the TATD in figure 8 and the corresponding RETOOL theory T_b below. We will see the engine itself as another process, extremely simplified to account only for the behavior of the (automatic) clutch. This second process is represented by the TATD in figure 9 and the corresponding theory T_e below. Notice that while the brakes operate instantaneously, the activation and deactivation of the clutch mechanism both have a minimal delay of 1 time unit.

$$\begin{array}{rl} T_b = & atb1 \rightarrow \neg atb2 \\ & atb2 \rightarrow \neg atb1 \\ & [\,](\neg atb1 \wedge \neg atb2) \\ & brake \supset atb1_0\delta^{\infty} atb2 \\ & release \supset atb2_0\delta^{\infty} atb1 \end{array}$$

$$\begin{array}{rl} T_e = & ate1 \rightarrow \neg ate2 \\ & ate2 \rightarrow \neg ate1 \\ & [\,](\neg ate1 \wedge \neg ate2) \\ & unlink \supset ate1_1\delta^{\infty} ate2 \\ & link \supset ate2_1\delta^{\infty} ate1 \end{array}$$

We will compose T_b and T_e using a third theory, T_c, and morphisms σ_b and σ_e (shown below) as the communication channel.

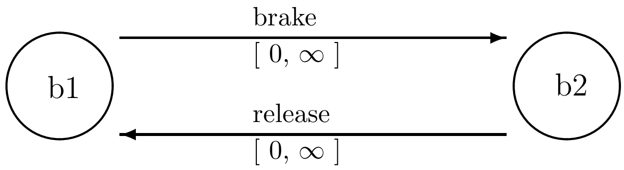

Figure 8: TATD for the braking mechanism of an automobile

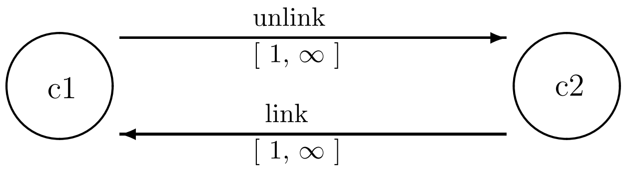

Figure 9: TATD for the automatic clutch of the engine of an automobile

$$
\begin{array}{rl}
T_c = & atc1 \rightarrow \neg atc2 \\
& atc2 \rightarrow \neg atc1 \\
& [\,](\neg atc1 \wedge \neg atc2) \\
& g \supset atc1_0 \delta^{\infty} atc2 \\
& h \supset atc2_0 \delta^{\infty} atc1
\end{array}
$$

$$
\begin{array}{rlcl}
\sigma_b : & T_c & \longrightarrow & T_b \\
& atc1 & \longmapsto & atb1 \\
& atc2 & \longmapsto & atb2 \\
& g & \longmapsto & brake \\
& h & \longmapsto & release
\end{array}
$$

$$
\begin{array}{rlcl}
\sigma_e : & T_c & \longrightarrow & T_e \\
& atc1 & \longmapsto & ate1 \\
& atc2 & \longmapsto & ate2 \\
& g & \longmapsto & unlink \\
& h & \longmapsto & link
\end{array}
$$

By computing the pushout as explained in the previous subsection, we produce theory T_p, which represents the joint (parallel) behavior of the brakes and the clutch interacting together. As the pushout object is unique up to isomorphism, we are free to rename the attributes and actions in T_p (as long, of course, as no new identifications of symbols arise). Here, we simply choose the concatenation of parts of the names in T_b and T_e:

$$
\begin{array}{rl}
T_p = & atb1e1 \rightarrow \neg atb2e2 \\
& atb2e2 \rightarrow \neg atb1e1 \\
& [\,](\neg atb1e1 \wedge \neg atb1e2) \\
& brakeunlink \supset atb1e1_1\delta^{\infty} atb2e2 \\
& releaselink \supset atb2e2_1\delta^{\infty} atb1e1
\end{array}
$$

Notice that, in identifying action symbols *brake* and *unlink*, which had different lower bounds, the stricter minimal delay (1, the lower bound of *unlink*) prevailed. A similar situation occurred with *release* and *link*.

6. Concluding Remarks

The approach adopted for modeling real-time aspects (timed transition systems for the object's lifecycle specification) relies on an extension of Timed Transition Systems and Timed Action Transition Diagrams as presented in [HMP92]. The extension consists in working with a specificational level based on the use of action modalities and the δ operator. The use of action names allows us to separate methods from their functionality and, therefore, model their reactive and real-time aspects. Action modalities are then used for specifying their functionality (pre/postconditions). The δ operator refers to the necessary conditions and time bounds of actions. The Metric Temporal Logic (MTL) of [Cha95] was extended with action terms as propositions and related to their specification by several inference rules.

It was also illustrated how the integration of different objects in a system is supported. The framework for integration is based on the categorial approach presented in [BLM97] and [FM92].

Work in progress includes the search for an automatic theorem-proving strategy for the RETOOL/MTL combination and the application of this framework to different fields, e.g. the design of hypermedia documents, in which real-time constraints also occur naturally.

References

[Ama00] Amaral, F.N., "RETOOL: Uma Lógica de Ações para Sistemas de Transição Temporizados", M.Sc. Dissertation, Dept. of Informatics, PUC-RJ, Brazil, 2000.

[BW90] Barr, M., Wells, C., *Category Theory for Computing Science*, Prentice Hall International, 1990.

[BLM97] Bicarregui, J., Lano, K. and Maibaum, T., "Towards a Compositional Interpretation of Object Diagrams", in *Proc IFIP Working Conference on Algorithmic Languages and Calculi*, Chapman and Hall, 1997.

[Car96] Carvalho, S., "The DDL Programming Language", Technical Report, Dept. of Informatics, PUC-RJ, Brazil, 1996.

[CFH97] Carvalho, S., Fiadeiro, J. e Haeusler, E.H., "A Formal Approach to Real-Time Object-Oriented Software", in *Proc. 22nd IFAC/IFIP Workshop on Real-Time Programming WRTP'97*, Elsevier 1997.

[Cha95] Chang, E., *Compositional Verification of Reactive and Real-Time Systems*, PhD Thesis, Stanford University, 1995.

[CMCH96] Clarke, E.M., McMillan, K.L., Campos, S., Hartonas-Garmhausen, "Symbolic Model-Checking", in CAV 96, LNCS 1102, 1996.

[FH98] Fiadeiro, J. and Haeusler, E.H., "Bringing It About On Time (Extended Abstract)", in *Proc. I IMLLAI, Fortaleza, CE, Brazil*, 1998.

[FM92] Fiadeiro, J. and Maibaum, T., "Temporal Theories as Modularization Units for Concurrent Systems Specification", in *Formal Aspects of Computing* 4(3), 1992.

[Gog89] Goguen, J., *A Categorical Manifesto*, Technical Report PRG-72, Programming Research Group, University of Oxford, March 1989.

[Har84] Harel, D., "Dynamic Logic", in *Handbook of Philosophical Logic Vol II* (Dov Gabbay, F. Guenthner (eds.)), D. Reidel, 1984.

[HHMF98] Haeusler, E.H., Haeberer, A., Maibaum, T., Fiadeiro, J.L., "ARTS: A Formally Supported Environment for Object Oriented Software Development", in *Proc. Workshop on Automating the Process of Software Development, ECOOP 98*, 1998.

[HMP92] Henzinger, T., Manna, Z., e Pnuelli, A., "Timed Transition Systems", in *Real Time: Theory in Practice* (J.W. de Bakker, C. Huizing, W.P. de Roever e G. Hozenberg (eds.)), LNCS 600, Springer-Verlag, 1992.