A Real-time Action Logic of Objects

K. Lano, **J. Bicarregui**, Dept. of Computing, Imperial College, 180 Queens Gate, London, SW7 2BZ.

S. Kent, Dept. of Computing, University of Brighton.

Abstract

This paper presents work performed in the EPSRC "Object-oriented Specification of Reactive and Real-time Systems" project. It aims to extend the Object Calculus of Fiadeiro and Maibaum to cover durative actions and real-time constraints.

We define a core logic, termed "Real-time action logic" (RAL) which can provide an axiomatic semantics and reasoning framework for concurrent, real-time and object-oriented specification languages. The logic could also be viewed as providing the basis of a specification language in its own right.

We show how a modal action logic (MAL) and real-time logic (RTL) for reasoning about concurrent object-oriented programs and specifications can be derived from RAL, and indicate how this formalism can be used to provide an axiomatic semantics for a large part of the object-oriented specification language VDM^{++}.

1 Introduction

A variety of semantics for object-orientation have been developed, from the perspectives of logic [5], set theory [18], type theory [2], category theory [6, 17] and process algebra [8].

The logic approach is exemplified by the object calculus of [5]. First order temporal logic specifications define the properties of objects, including the effects, permission constraints and liveness requirements of methods. Methods are interpreted as action symbols in the logic, whilst attributes are interpreted as attribute symbols. The approach in this paper will be based on elements of the object calculus (structured temporal theory presentations with interpretations between theories), but will generalise it by using *durative* actions which can overlap in their executions. The issue of *dynamic reconfiguration*, one of the main shortcomings of the object calculus, is addressed for RAL in [9].

Temporal logic is relevant to object-orientation since systems of objects are potentially highly dynamic in nature: both the location of execution activity and the interconnections between objects will change over time. Some object-oriented concepts, such as aggregation, and subtype migration, are inherently time-based, and can only be given a precise meaning by considering how sets of objects or relationships between objects can change over time.

A particular dilemma that we address is the need to avoid the "next" operator $\bigcirc$ of linear temporal logic (LTL) in the context of real-time and distributed systems, where no such (global) next instant can be sensibly identified, or in the context of refinement, where a system and its refinement may use distinct granularities of time. The "next" operator is however a very intuitive and natural means to specify state transitions, so we wish to retain it in some form. This is achieved by interpreting "next time" as "next method invocation time of a method of the class on the current object", so that it becomes local to a particular object and the class of that object.

2 Real-time Action Logic

We will use RAL to express the semantics of the VDM^{++} language [11, 4].

2.1 Logic

RAL can be presented independently of any decomposition of a specification into modules (classes), although in this paper we will focus on a class-oriented view. The syntactic elements of an RAL theory are:

- action symbols α, for example, invocations **a!m** of a method **m** on an object **a**. These symbols may be parameterised. Actions may also be defined by their effect, as in TLA [10];
- attribute symbols, denoting values which can change from world to world (time to time). For example **x.att**, the attribute **att** owned by an object **x**. The attribute symbols always include **now**, representing the (global) current time;
- the usual type, function and predicate symbols of typed predicate calculus, including the operators $\in$, set comprehension, $\cup$, $\mathbb{F}$, etc, of ZF set theory;
- the type **TIME**, assumed to be totally ordered by a relation $<$, with a least element 0, and with $\mathbb{N} \subseteq$ **TIME**. It satisfies the axioms of the set of non-negative elements of a totally ordered topological ring, with addition operation + and

unit 0, and multiplication operation $*$ with unit 1;

- predicate logic connectives and quantifiers;

- modal operators: $\circledcirc$ "holds at a time" and $\circledast$ "value at a time" and event time terms: $\rightarrow(\alpha, \mathbf{i})$ the time of request of the $\mathbf{i}$-th invocation of action α, $\uparrow(\alpha, \mathbf{i})$ the time of activation of this invocation, and $\downarrow(\alpha, \mathbf{i})$ the time of termination of this invocation. $\mathbf{i}$ ranges over $\mathbb{N}_1$, the non-zero natural numbers.

The following operators can be defined in terms of the above symbols:

- the modal action formulae $[\alpha]\mathbf{P}$ "α establishes $\mathbf{P}$". $\mathbf{P}$ may contain references $\overleftarrow{\mathbf{e}}$ to the value of $\mathbf{e}$ at commencement of the invocation of α being considered;

- the operator $\supset$ representing the calling relation between two actions;

- the RTL [7] event-time operators $\clubsuit(\varphi := \mathbf{true}, \mathbf{i})$ and $\clubsuit(\varphi := \mathbf{false}, \mathbf{i})$ giving the times of the $\mathbf{i}$-th occurrences of the events of a predicate φ becoming true or false, respectively;

- counters $\#\mathbf{req}(\alpha)$, $\#\mathbf{act}(\alpha)$ and $\#\mathbf{fin}(\alpha)$ for request, activation and termination events;

- the temporal logic operators $\Box$, $\diamond$, $\bigcirc$;

- action combinators ; , || (parallel non-interfering execution), assignment, etc.

If the set of action, attribute, function and predicate symbols is denoted by Σ, we denote the RAL formalism based on this set by RAL(Σ).

Specific to the object-oriented view are types @**Any** of all possible object identifiers, and subsorts @**C** of this type which represent the possible object identifiers of objects of class **C**.

A predicate added for concurrent object-oriented systems is a test for *enabling* of an action α (whether a request for execution of α will be serviced or not). This is expressed by $\mathbf{enabled}(\alpha)$.

2.1.1 Examples of Specification

Some examples of the type of properties that can be specified in an abstract declarative manner using RAL are *periodic* timing constraints: "$\mathbf{m}$ initiates every $\mathbf{t}$ seconds, and in the order of its requests":

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot \uparrow(\mathbf{m}(\mathbf{x}), \mathbf{i}+1) \;=\; \uparrow(\mathbf{m}(\mathbf{x}), \mathbf{i}) + \mathbf{t}$$

Sporadic constraints can also be directly expressed.

We can express fairness requirements such as the 'first come, first served' queueing discipline for method requests:

$$\forall \mathbf{i}, \mathbf{j} : \mathbb{N}_1 \cdot \rightarrow(\mathbf{m}, \mathbf{i}) < \rightarrow(\mathbf{m}, \mathbf{j}) \;\Rightarrow\; \uparrow(\mathbf{m}, \mathbf{i}) \leq \uparrow(\mathbf{m}, \mathbf{j})$$

"If the $\mathbf{i}$-th request for invocation of $\mathbf{m}$ is received before the $\mathbf{j}$-th, then the $\mathbf{i}$-th invocation instance of $\mathbf{m}$ will be activated before the $\mathbf{j}$-th."

Prioritisation constraints, permission constraints, timeouts and responsiveness constraints can all be specified in a direct manner using RAL [9]. All the forms of method invocation protocols for concurrent objects described in [1] can be precisely described in this logic in a similar way. The Ada rendezvous interpretation is the default for VDM^{++}.

2.1.2 Attributes and Actions

For a specification **S** consisting a set of classes, the attribute symbols are as follows:

- $\mathbf{x}.\mathbf{att}$ for $\mathbf{x}$: @**C** and **att** an attribute of a class **C** of **S**;

- $\mathbf{x}.\mathbf{now}$ for $\mathbf{x}$: @**C**, representing the local time of object $\mathbf{x}$. Here this will be equated to the global time attribute **now**;

- $\overline{\mathbf{C}}$ for each class **C**, representing the set of *existing* objects of **C**. This is of type $\mathbb{F}$(@**C**).

Derived attributes of a class will include *event counters* $\#\mathbf{act}(\mathbf{m})$, $\#\mathbf{fin}(\mathbf{m})$ as defined below.

The action symbols are:

- $\mathbf{new_C}(\mathbf{c})$ for **C** a class of **S** and $\mathbf{c}$: @**C**;

- $\mathbf{x}!\mathbf{m}(\mathbf{e})$ for $\mathbf{x}$: @**C** and $\mathbf{m}$ a method of **C**, with $\mathbf{e} : \mathbf{X_{m,C}}$ a term in the type of the input parameters of $\mathbf{m}$ in **C**.

  If $\mathbf{m}$ has both a synchronous ("secured") and asynchronous ("relaxed") component, then we have two actions $\mathbf{m_s}$ and $\mathbf{m_r}$ in place of $\mathbf{m}$;

- **pre Guard post Post** where **Guard** is an expression over a set of attributes, and **Post** can additionally contain expressions of the form $\overleftarrow{\mathbf{e}}$ referring to the value of the expression $\mathbf{e}$ at commencement of execution of the action.

  These are similar to the actions of TLA, and represent time intervals where **Guard** holds at the start of the interval, and **Post** holds at the end of the interval.

We write $\mathbf{x}.\uparrow(\mathbf{m}(\mathbf{e}), \mathbf{i})$ for $\uparrow(\mathbf{x}!\mathbf{m}(\mathbf{e}), \mathbf{i})$ etc to make the notation used for objects more uniform.

2.1.3 Derived Actions and Attributes

For an object $\mathbf{x} : @\mathbf{C}$ event occurrence times $\clubsuit(\varphi := \mathbf{true}, \mathbf{i})$ and $\clubsuit(\varphi := \mathbf{false}, \mathbf{i})$ can be defined from the above language.

Event counters are also derived operators:

$$\begin{aligned}&\mathbf{x}.\#\mathbf{act}(\mathbf{m}(\mathbf{e})) \ = \\ &\qquad \mathbf{card}(\{\mathbf{j} : \mathbb{N}_1 \mid \mathbf{x}.\uparrow(\mathbf{m}(\mathbf{e}), \mathbf{j}) \ < \ \mathbf{now}\})\end{aligned}$$

This definition involves $<$ because we consider $\#\mathbf{act}(\mathbf{m})$ to be incremented *just after* the moment at which $\mathbf{m}$ initiates execution.

$$\begin{aligned}&\mathbf{x}.\#\mathbf{req}(\mathbf{m}(\mathbf{e})) \ = \\ &\qquad \mathbf{card}(\{\mathbf{j} : \mathbb{N}_1 \mid \mathbf{x}.\rightarrow(\mathbf{m}(\mathbf{e}), \mathbf{j}) \ < \ \mathbf{now}\})\end{aligned}$$

$$\begin{aligned}&\mathbf{x}.\#\mathbf{fin}(\mathbf{m}(\mathbf{e})) \ = \\ &\qquad \mathbf{card}(\{\mathbf{j} : \mathbb{N}_1 \mid \mathbf{x}.\downarrow(\mathbf{m}(\mathbf{e}), \mathbf{j}) \ \leq \ \mathbf{now}\})\end{aligned}$$

In contrast, $\#\mathbf{fin}(\mathbf{m}(\mathbf{e}))$ is incremented *just before* $\mathbf{m}$ terminates execution.

$\mathbf{void}(\mathbf{obj})$ abbreviates $\mathbf{obj} \notin \overline{\mathbf{C}}$, where $\mathbf{obj} : @\mathbf{C}$.

$\circledast$ binds more closely than any other binary operator on terms (although the name constructor . binds more closely), but less than any unary term operator. Thus $\mathbf{v} \geq \#\mathbf{act}(\mathbf{m})\circledast\mathbf{v}$ denotes $\mathbf{v} \geq (\#\mathbf{act}(\mathbf{m})\circledast\mathbf{v})$.

The actions $\mathbf{pre}\,\mathbf{G}\ \ \mathbf{post}\,\mathbf{P}$ name actions α with the following properties:

$$\begin{aligned}&\forall\, \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{now} = \uparrow(\alpha, \mathbf{i}) \ \Rightarrow \ \mathbf{G}\circledcirc\uparrow(\alpha, \mathbf{i}) \\ &\forall\, \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{now} = \uparrow(\alpha, \mathbf{i}) \ \Rightarrow \\ &\qquad\qquad \mathbf{P}[\mathbf{att}\circledast\uparrow(\alpha, \mathbf{i})/\overleftarrow{\mathbf{att}}]\circledcirc\downarrow(\alpha, \mathbf{i})\end{aligned}$$

In other words, $\mathbf{G}$ must be true at each invocation time of α, whilst $\mathbf{P}$, with each "hooked" attribute $\overleftarrow{\mathbf{att}}$ interpreted as the value $\mathbf{att}\circledast\uparrow(\alpha, \mathbf{i})$ of $\mathbf{att}$ at initiation of α, holds at the corresponding termination time.

2.1.4 Formulae

For any class $\mathbf{C}$ the following are the formulae in its RAL language.

1. $\mathbf{P}(\mathbf{e}_1, \ldots, \mathbf{e_n})$ for an $\mathbf{n}$-ary predicate symbol $\mathbf{P}$ and terms $\mathbf{e}_1$, ..., $\mathbf{e_n}$;
2. $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \Rightarrow \psi$, $\neg\, \phi$ for formulae ϕ and ψ;
3. $\phi\circledcirc\mathbf{t}$ for formulae ϕ and time-valued terms $\mathbf{t}$ – "ϕ holds at time $\mathbf{t}$";
4. $\forall\, \mathbf{SD} \cdot \phi$, $\exists\, \mathbf{SD} \cdot \phi$ for declarations $\mathbf{SD}$ and formulae ϕ;
5. $\Box^\tau\theta$, $\Box_{\mathbf{a},\mathbf{C}}\theta$ and $\bigcirc_{\mathbf{a},\mathbf{C}}\theta$ for formulae θ;
6. $\diamond^\tau\theta$, $\diamond_{\mathbf{a},\mathbf{C}}\theta$ for formulae θ;
7. $\mathbf{enabled}(\mathbf{x}!\mathbf{m})$ and $\mathbf{enabled}(\mathbf{x}!\mathbf{m}(\mathbf{e}))$ for methods $\mathbf{m}$, $\mathbf{e}$ in the input type of $\mathbf{m}$, $\mathbf{x} : @\mathbf{C}$, $\mathbf{m}$ a method of $\mathbf{C}$.

$\Box_{\mathbf{a},\mathbf{C}}\phi$ denotes that ϕ holds at each future initiation time of a method invocation $\mathbf{a}!\mathbf{m}$ on an object $\mathbf{a} : @\mathbf{C}$, where $\mathbf{m}$ is a method of the class $\mathbf{C}$. In other words it abbreviates

$$\begin{aligned}&\forall\, \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{a}.\uparrow(\mathbf{m}_1, \mathbf{i}) \geq \mathbf{now} \Rightarrow \phi\circledcirc\mathbf{a}.\uparrow(\mathbf{m}_1, \mathbf{i}) \\ &\wedge \ldots \ \wedge \\ &\forall\, \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{a}.\uparrow(\mathbf{m_n}, \mathbf{i}) \geq \mathbf{now} \Rightarrow \phi\circledcirc\mathbf{a}.\uparrow(\mathbf{m_n}, \mathbf{i})\end{aligned}$$

where $\underline{\mathbf{methods}}(\mathbf{C}) = \{\mathbf{m}_1, \ldots, \mathbf{m_n}\}$.

Similarly, $\diamond_{\mathbf{a},\mathbf{C}}\phi$ and $\bigcirc_{\mathbf{a},\mathbf{C}}\phi$ can be defined in terms of the basic RAL operators.

$\Box^\tau\phi$ is: $\quad \forall\, \mathbf{t} : \mathbf{TIME} \cdot \mathbf{t} \geq \mathbf{now} \Rightarrow \phi\circledcirc\mathbf{t}$ whilst $\diamond^\tau\phi$ is: $\quad \exists\, \mathbf{t} : \mathbf{TIME} \cdot \mathbf{t} \geq \mathbf{now} \wedge \phi\circledcirc\mathbf{t}$.

$\Box^\tau\phi$ denotes that ϕ holds at all future times – it is not relative to a class $\mathbf{C}$.

An action symbol α can be used as a formula – it then denotes $\#\mathbf{active}(\alpha) > 0$.

The calling operator $\supset$ is defined by:

$$\begin{aligned}&\alpha \supset \beta \ \equiv \\ &\quad \forall\, \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{now} = \uparrow(\alpha, \mathbf{i}) \ \Rightarrow \\ &\qquad \exists\, \mathbf{j} : \mathbb{N}_1 \cdot \uparrow(\beta, \mathbf{j}) = \uparrow(\alpha, \mathbf{i}) \ \wedge \\ &\qquad\qquad \downarrow(\beta, \mathbf{j}) = \downarrow(\alpha, \mathbf{i})\end{aligned}$$

In other words: every invocation interval of α is also one of β.

The MAL operator $[\alpha]\mathbf{P}$ is defined as:

$$\begin{aligned}&[\alpha]\mathbf{P} \ \equiv \\ &\quad \forall\, \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{now} = \uparrow(\alpha, \mathbf{i}) \ \Rightarrow \\ &\qquad\qquad \mathbf{P}[\mathbf{att}\circledast\uparrow(\alpha, \mathbf{i})/\overleftarrow{\mathbf{att}}]\circledcirc\downarrow(\alpha, \mathbf{i})\end{aligned}$$

where the same substitution is used as for the definition of $\mathbf{pre}\,\mathbf{G}\ \ \mathbf{post}\,\mathbf{P}$ above.

Notice that therefore $[\mathbf{pre}\,\mathbf{G}\ \ \mathbf{post}\,\mathbf{P}](\overleftarrow{\mathbf{G}} \wedge \mathbf{P})$ as expected, and that

$$(\alpha \supset \beta) \ \Rightarrow \ ([\beta]\mathbf{P} \Rightarrow [\alpha]\mathbf{P})$$

for any $\mathbf{P}$ in the language concerned.

Assignment $\mathbf{t}_1 := \mathbf{t}_2$ can be defined as the action $\mathbf{pre}\,\mathbf{true}\ \ \mathbf{post}\,\mathbf{t}_1 = \overleftarrow{\mathbf{t}_2}$ where $\mathbf{t}_1$ is an attribute symbol. Similarly sequential composition ; and parallel composition $||$ of actions can be expressed as derived combinators:

$$\begin{aligned}&\forall\, \mathbf{i} : \mathbb{N}_1 \cdot \exists\, \mathbf{j}, \mathbf{k} : \mathbb{N}_1 \cdot \\ &\qquad \uparrow(\alpha;\ \beta, \mathbf{i}) = \uparrow(\alpha, \mathbf{j}) \ \wedge \\ &\qquad \downarrow(\alpha;\ \beta, \mathbf{i}) = \downarrow(\beta, \mathbf{k}) \ \wedge \\ &\qquad \uparrow(\beta, \mathbf{k}) = \downarrow(\alpha, \mathbf{j})\end{aligned}$$

and

$$\begin{array}{l} \forall \mathbf{j}, \mathbf{k} : \mathbb{N}_1 \cdot \\ \quad \uparrow(\beta, \mathbf{k}) = \downarrow(\alpha, \mathbf{j}) \Rightarrow \\ \quad \exists \mathbf{i} : \mathbb{N}_1 \cdot \\ \qquad \uparrow(\alpha;\ \beta, \mathbf{i}) = \uparrow(\alpha, \mathbf{j}) \wedge \\ \qquad \downarrow(\alpha;\ \beta, \mathbf{i}) = \downarrow(\beta, \mathbf{k}) \end{array}$$

These two conditions yield the usual axiom that

$$[\alpha;\ \beta]\varphi \ \equiv\ [\alpha][\beta]\varphi$$

Conditionals have the expected properties:

$$\mathbf{E} \Rightarrow (\mathbf{if\ E\ then\ S_1\ else\ S_2} \quad \supset \quad \mathbf{S}_1)$$
$$\neg\, \mathbf{E} \Rightarrow (\mathbf{if\ E\ then\ S_1\ else\ S_2} \quad \supset \quad \mathbf{S}_2)$$

Similarly, **while** loops can be defined.

A synchronous method invocation $\mathbf{a!m(e)}$ is interpreted as an **invoke** statement:

$$\mathbf{invoke\ a!m(e)}$$

An instance $(\mathbf{S}, \mathbf{i})$ of this statement has the properties:

$$\begin{array}{l} \forall \mathbf{i} : \mathbb{N}_1 \cdot \exists \mathbf{j} : \mathbb{N}_1 \cdot \\ \quad \uparrow(\mathbf{S}, \mathbf{i}) \ =\ \mathbf{a}.\!\rightarrow\!(\mathbf{m(e)}, \mathbf{j}) \ \wedge \\ \quad \downarrow(\mathbf{S}, \mathbf{i}) \ =\ \mathbf{a}.\!\downarrow\!(\mathbf{m(e)}, \mathbf{j}) \end{array}$$

The [ ] operator can be used to concisely express properties of action invocations without requiring reference to the index of these invocations. For example, the property that all actions take non-zero time to execute can be expressed by: $[\alpha](\mathbf{now} > \overleftarrow{\mathbf{now}})$.

$\circledcirc$ binds more closely than $\Box_{\mathbf{a},\mathbf{C}}$, $\Diamond_{\mathbf{a},\mathbf{C}}$, $\bigcirc_{\mathbf{a},\mathbf{C}}$, $\Box^{\tau}$ and $\Diamond^{\tau}$. These latter operators bind as for $\neg$.

2.1.5 Axioms

The axioms of predicate calculus and ZF set theory are adopted, with some modifications.

For example, the quantifier axiom:

$$(\forall\, \mathbf{v} : \mathbf{T} \cdot \varphi) \Rightarrow \varphi[\mathbf{e}/\mathbf{v}]$$

is only asserted for φ such that $\mathbf{e}$ is free for the variable $\mathbf{v}$ in φ (that is, no variable free in $\mathbf{e}$ is bound at the locations of the substituted occurrences of $\mathbf{v}$ in φ), and such that the substitution does not introduce occurrences of attributes within modal operators in φ.

The core logical axioms are:

$$(\mathbf{C}1):\ \forall\, \mathbf{i} : \mathbb{N}_1 \cdot \rightarrow\!(\mathbf{m(e)}, \mathbf{i}) \le\ \rightarrow\!(\mathbf{m(e)}, \mathbf{i}+1)$$

"the $\rightarrow\!(\mathbf{m(e)}, \mathbf{i})$ times are enumerated in order of their occurrence."

$$\begin{array}{l} (\mathbf{C}2):\ \forall\, \mathbf{i} : \mathbb{N}_1 \cdot \\ \qquad \rightarrow\!(\mathbf{m(e)}, \mathbf{i}) \le\ \uparrow\!(\mathbf{m(e)}, \mathbf{i}) <\ \downarrow\!(\mathbf{m(e)}, \mathbf{i}) \end{array}$$

"every invocation must be requested before it can initiate, and initiates before it terminates."

The *compactness* condition is that for all $\mathbf{p} \in \mathbb{N}_1$ there are only finitely many $\mathbf{i} : \mathbb{N}_1$ such that $\uparrow(\alpha, \mathbf{i}) < \mathbf{p}$, for each action α. Similar conditions are required for the $\rightarrow$ and $\downarrow$ times.

$$(\mathbf{C}3):\ \varphi \equiv \varphi \circledcirc \mathbf{now}$$

$$\begin{array}{l} (\mathbf{C}4): \\ \quad (\psi \wedge \varphi)\circledcirc\mathbf{t} \ \equiv\ \psi\circledcirc\mathbf{t} \wedge \varphi\circledcirc\mathbf{t} \\ \quad (\psi \vee \varphi)\circledcirc\mathbf{t} \ \equiv\ \psi\circledcirc\mathbf{t} \vee \varphi\circledcirc\mathbf{t} \\ \quad (\psi \Rightarrow \varphi)\circledcirc\mathbf{t} \ \equiv\ \psi\circledcirc\mathbf{t} \Rightarrow \varphi\circledcirc\mathbf{t} \\ \quad (\exists\, \mathbf{v} : \mathbf{T} \cdot \varphi)\circledcirc\mathbf{t} \ \equiv\ \exists\, \mathbf{v} : \mathbf{T} \cdot \varphi\circledcirc\mathbf{t} \\ \quad (\forall\, \mathbf{v} : \mathbf{T} \cdot \varphi)\circledcirc\mathbf{t} \ \equiv\ \forall\, \mathbf{v} : \mathbf{T} \cdot \varphi\circledcirc\mathbf{t} \end{array}$$

for any time-valued term $\mathbf{t}$ and formulae ψ and φ. In the last two formulae, $\mathbf{t}$ has no free variables.

$$\begin{array}{l} (\mathbf{C}5): \\ \mathbf{e} = (\mathbf{e}\circledast\mathbf{now}) \qquad\qquad \mathbf{now}\circledast\mathbf{t} \ =\ \mathbf{t} \\ (\mathbf{e}\circledast\mathbf{t}_1)\circledast\mathbf{t}_2 = \mathbf{e}\circledast(\mathbf{t}_1\circledast\mathbf{t}_2) \end{array}$$

In general $\mathbf{e}\circledast\mathbf{t} = \mathbf{e}$ holds if $\mathbf{e}$ contains no variables or attribute symbols.

$$(\mathbf{C}6): \qquad \theta\circledcirc\mathbf{t} \equiv \theta^{*\mathbf{t}}$$

for θ without modal operators, and where $\theta^{*\mathbf{t}}$ is θ with each (outermost) term $\mathbf{e}$ occurring in a subformula of θ replaced by $\mathbf{e}\circledast\mathbf{t}$, and $\mathbf{t}$ is a time-valued term without free variables. Similarly

$$\mathbf{g}(\mathbf{e}_1\circledast\mathbf{t}, \ldots, \mathbf{e_n}\circledast\mathbf{t}) = \mathbf{g}(\mathbf{e}_1, \ldots, \mathbf{e_n})\circledast\mathbf{t}$$

for each function symbol $\mathbf{g}$. This means we can eliminate $\circledcirc$ as an operator and only use $\circledast$.

Of key importance for reasoning about objects is a *framing* or *locality* constraint [5], which asserts that over any interval in which no action executes, no attribute changes in value except for **now**:

Any interval which satisfies the following action $\mathbf{x!idle_C}$:

pre true
post

$$\begin{array}{l} \mathbf{x.\#act(m_1)} \ =\ \overleftarrow{\mathbf{x.\#act(m_1)}} \ = \\ \mathbf{x.\#fin(m_1)} \ =\ \overleftarrow{\mathbf{x.\#fin(m_1)}} \quad \wedge \\ \ldots \ \wedge\ \mathbf{x.\#act(m_n)} \ =\ \overleftarrow{\mathbf{x.\#act(m_n)}} \\ \qquad = \ \mathbf{x.\#fin(m_n)} \ =\ \overleftarrow{\mathbf{x.\#fin(m_n)}} \end{array}$$

where the $\mathbf{m_i}$ are all the methods of $\mathbf{C}$, must also satisfy the axioms:

$$[\mathbf{x}!\mathbf{idle_C}](\mathbf{x.att} = \overleftarrow{\mathbf{x.att}})$$

for every attribute of the object, except $\mathbf{x.now}$.

This locality principle reduces to that of the object calculus in the case that all actions have duration 1 and $\mathbf{TIME} = \mathbb{N}$.

The frame axiom restricts the subtyping relation in a way similar to that of Liskov's definition of subtyping [16]. If it is accepted as a part of the theory $\Gamma_\mathbf{C}$ of a class, then we cannot prove that a class

```
class C
instance variables x :  ℤ
methods
  inc()  ==  x  :=  x  +  1;

  val()  value ℤ ==
      return x
end C
```

is a supertype of

```
class D is subclass of C
methods
  dec()  ==  x  :=  x  –  1
end D
```

because there are state changes possible for $\mathbf{d} : @\mathbf{D}$ which are not possible for any instances of $\mathbf{C}$ (where we take subtyping as being equivalent to theory extension).

2.1.6 Theorems

Some useful theorems are:

$(\mathbf{vi})$:
$\Box^\tau(\mathbf{x.\#fin(m)} \leq \mathbf{x.\#act(m)} \leq \mathbf{x.\#req(m)})$

The following axioms of LTL [19] hold:

$(\mathbf{xiii})$:
$\Box_{\mathbf{a,C}}\psi \Rightarrow \bigcirc_{\mathbf{a,C}}\psi$
$\Box_{\mathbf{a,C}}(\Box_{\mathbf{a,C}}\psi \Rightarrow \psi)$
$\neg \bigcirc_{\mathbf{a,C}}\, \varphi \Rightarrow \bigcirc_{\mathbf{a,C}}\neg\, \varphi$
$\bigcirc_{\mathbf{a,C}}(\varphi \Rightarrow \psi) \Rightarrow (\bigcirc_{\mathbf{a,C}}\varphi \Rightarrow \bigcirc_{\mathbf{a,C}}\psi)$
$\Box_{\mathbf{a,C}}\psi \Rightarrow \bigcirc_{\mathbf{a,C}}\Box_{\mathbf{a,C}}\psi$
$\neg \bigcirc_{\mathbf{a,C}}\, \mathbf{false} \;\Rightarrow\; (\bigcirc_{\mathbf{a,C}}\mathbf{P(e)} \;\equiv\; \mathbf{P}(\bigcirc_{\mathbf{a,C}}\mathbf{e}))$
for predicate symbols $\mathbf{P}$.
$\forall\, \mathbf{v} : \mathbf{T} \cdot \bigcirc_{\mathbf{a,C}}\psi \;\equiv\; \bigcirc_{\mathbf{a,C}}\forall\, \mathbf{v} : \mathbf{T} \cdot \psi$

Also:

$$\Box^\tau\psi \Rightarrow \Box_{\mathbf{a,C}}\psi \quad \Box^\tau\psi \Rightarrow \Diamond^\tau\psi \quad \Diamond_{\mathbf{a,C}}\,\psi \Rightarrow \Diamond^\tau\psi$$

Note that the axiom $\Box_{\mathbf{a,C}}\psi \Rightarrow \Diamond_{\mathbf{a,C}}\psi$ need not be valid since there may not be any method activations at or after the current time.

Axioms of the modal logic $\mathbf{S}_5$ hold:

$(\mathbf{xvii})$:
$\Box_{\mathbf{a,C}}(\Box_{\mathbf{a,C}}\varphi \Rightarrow \varphi)$
$\Box_{\mathbf{a,C}}\varphi \vee \neg\, \Box_{\mathbf{a,C}}\varphi$
$\Box_{\mathbf{a,C}}(\varphi \Rightarrow \psi) \Rightarrow (\Box_{\mathbf{a,C}}\varphi \Rightarrow \Box_{\mathbf{a,C}}\psi)$
$\Box_{\mathbf{a,C}}\varphi \Rightarrow \Box_{\mathbf{a,C}}\Box_{\mathbf{a,C}}\varphi$

The same axioms hold for $\Box^\tau$ in place of $\Box_{\mathbf{a,C}}$.

$$\Box^\tau(\theta \circledcirc \mathbf{t}) \equiv \theta \circledcirc \mathbf{t} \qquad \theta \circledcirc \mathbf{t} \Rightarrow \Box_{\mathbf{a,C}}(\theta \circledcirc \mathbf{t})$$
$$\Diamond^\tau(\theta \circledcirc \mathbf{t}) \equiv \theta \circledcirc \mathbf{t} \qquad \Diamond_{\mathbf{a,C}}(\theta \circledcirc \mathbf{t}) \Rightarrow \theta \circledcirc \mathbf{t}$$

where in the last four formulae, $\mathbf{t}$ is a time-valued term without free variables.

2.1.7 Inference Rules

The usual inference rules of predicate logic are taken. In addition the following rule is adopted:

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall\, \mathbf{t} : \mathbf{TIME} \cdot \varphi \circledcirc \mathbf{t}}$$

Derivability in the logic is denoted by $\vdash$ as usual.

3 Interpretations of Class Features

The following axioms enable an RAL theory to be given to a class specification. This means that practical development can use a combination of declarative RAL formulae and the more procedurally oriented VDM^{++} class descriptions, whilst reasoning about both parts of a specification can be carried out in a uniform formalism.

If we have a method definition in class $\mathbf{C}$ of the form:

```
m(x :  X_{m,C}) value y :  Y_{m,C}
    pre Pre_{m,C}  ==
                     Code_{m,C};
```

then the action $\mathbf{a}!\mathbf{m}(\mathbf{e})$ has the properties:

$$\mathbf{a.Pre}_{\mathrm{m,C}}[\mathbf{e}/\mathbf{x}] \;\wedge\; \mathbf{a} \;\in\; \overline{\mathbf{C}} \;\Rightarrow\; \mathbf{a}!\mathbf{m}(\mathbf{e}) \;\supset\; \mathbf{a.Code}_{\mathrm{m,C}}[\mathbf{e}/\mathbf{x}]$$

where each attribute $\mathbf{att}$ of $\mathbf{C}$ occurring in $\mathbf{Pre_{m,C}}$ is renamed to $\mathbf{a.att}$ in $\mathbf{a.Pre_{m,C}}$ and similarly for $\mathbf{Code_{m,C}}$. Additionally, invocations of actions $\mathbf{b}!\mathbf{n}(\mathbf{f})$ within $\mathbf{Code}$ are explicitly written as $\mathbf{invoke\ b}!\mathbf{n_s}(\mathbf{f})$ statements.

The operator $\bigcirc_{\mathbf{a,C}}$ can be used to simplify statements about the effects of methods in a mutex class $\mathbf{C}$. If $\mathbf{Code_{m,C}}$ is a specification statement $[\mathbf{ext\ wr\ v\ post\ P}]$ then we have:

$$\Box_{a,C} \forall\ \mathbf{e} : \mathbf{X_{m,C}} \cdot \\ \mathbf{a.Pre_{m,C}}[e/x] \wedge \mathbf{a} \in \overline{\mathbf{C}} \Rightarrow \\ (\mathbf{a!m(e)} \Rightarrow \bigcirc_{a,C}\mathbf{P'}[e/x])$$

where occurrences of $\overleftarrow{\mathbf{v_i}}$ in $\mathbf{P}$ are replaced by $\bullet_{\mathbf{a,C}}\mathbf{v_i}$ in $\mathbf{P'}$. This form is close to the usual characterisation of action effects in the object calculus.

In the case that $\mathbf{C}$ has a thread with an asynchronous code segment $\mathbf{Defn_{m,C}}$ for $\mathbf{m}$, the above axiom defines the properties of the $\mathbf{a!m_s(e)}$ action, and $\mathbf{a!m_r(e)}$ is defined by

$$\mathbf{a} \in \overline{\mathbf{C}} \Rightarrow \\ \mathbf{a!m_r(e)} \supset \mathbf{a.Defn_{m,C}}[e/x]$$

We also have that $\mathbf{a!m_s(e)}$ is followed by a corresponding invocation of $\mathbf{a!m_r(e)}$, with no other intervening method activation on the object:

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{a}.\downarrow(\mathbf{m_s(e)}, \mathbf{i}) = \mathbf{a}.\uparrow(\mathbf{m_r(e)}, \mathbf{i}) \vee \\ \exists \mathbf{j} : \mathbb{N}_1 \cdot \mathbf{a}.\uparrow(\mathbf{idle_C}, \mathbf{j}) = \mathbf{a}.\downarrow(\mathbf{m_s(e)}, \mathbf{i}) \wedge \\ \mathbf{a}.\downarrow(\mathbf{idle_C}, \mathbf{j}) = \mathbf{a}.\uparrow(\mathbf{m_r(e)}, \mathbf{i})$$

The initialisation of a class $\mathbf{C}$ can be regarded as a method $\mathbf{init_C}$ which is called automatically when an object $\mathbf{c}$ is created by the action $\mathbf{new_C}$:

$$\mathbf{new_C(c)} \supset \mathbf{c!init_C}$$

$\mathbf{new_C}$ itself has the properties:

$$\mathbf{new_C(c)} \Rightarrow \mathbf{c} \notin \overline{\mathbf{C}}$$

$$[\mathbf{new_C(c)}](\overline{\mathbf{C}} = \overleftarrow{\overline{\mathbf{C}}} \cup \{\mathbf{c}\})$$

A method must be enabled when it initiates execution:

$$\forall \mathbf{x} : @\mathbf{C};\ \mathbf{i} : \mathbb{N}_1;\ \mathbf{e} : \mathbf{X_{m,C}} \cdot \\ \mathbf{enabled(x!m(e))} \circledcirc \mathbf{x}.\uparrow(\mathbf{m(e)}, \mathbf{i})$$

for all methods $\mathbf{m}$ of $\mathbf{C}$.

The invariant of a class is true at every method initiation and termination time:

$$\Box_{\mathbf{a,C}}\mathbf{Inv_C} \wedge \forall \mathbf{i} : \mathbb{N}_1 \cdot \mathbf{Inv_C} \circledcirc \mathbf{a}.\downarrow(\mathbf{m_j}, \mathbf{i})$$

for each method $\mathbf{m_j}$ of $\mathbf{C}$ and $\mathbf{a} : @\mathbf{C}$. However, the typing constraints for attributes are *always* true:

$$\Box^{\tau}(\mathbf{a.att} \in \mathbf{T})$$

for each attribute declaration $\mathbf{att} : \mathbf{T}$ of $\mathbf{C}$.

Permission guards for a method $\mathbf{m}$ give conditions which must be implied by $\mathbf{enabled(m)}$:

```
per m  =>  G
```

yields the axiom $\mathbf{enabled(m)} \Rightarrow \mathbf{G}$.

The `whenever` construct of VDM^{++} is interpreted as follows. A statement

```
whenever χ
   also from δ ==> φ
```

asserts that φ must be true at some point in each interval of the form $[\mathbf{t}, \mathbf{t} + \delta]$ where $\mathbf{t}$ is a time at which χ becomes true.

Thus it can be expressed directly as:

$$\forall \mathbf{i} : \mathbb{N}_1;\ \exists \mathbf{t} : \mathbf{TIME} \cdot \\ \clubsuit(\chi := \mathbf{true}, \mathbf{i}) \leq \mathbf{t} \leq \clubsuit(\chi := \mathbf{true}, \mathbf{i}) + \delta \wedge \\ \varphi \circledcirc \mathbf{t}$$

This definition yields a transitivity principle.

Techniques for concurrent reasoning which arise from the RAL formalisation are: (i) induction principles based on the frame axiom. In VDM^{++} we divide classes into *active* mutex classes, and *passive* classes. The latter usually obey some weakening of mutual exclusion, to allow *reader* methods to co-execute on the same object. However, such passive objects can be treated as internally mutex in some respects, because updater methods execute in exclusion with themselves and other methods. Thus, if φ is true at creation of an object, and is preserved by every updater $\mathbf{m(e)}$, then it is true at every method activation and termination time (it may fail during updater executions):

$$[\mathbf{new_C(c)}]\varphi \wedge \\ (\bigwedge_{\mathbf{m} \in \underline{\mathbf{methods}}(\mathbf{C})} \forall \mathbf{e} : \mathbf{X_{m,C}} \cdot \mathbf{c} \in \overline{\mathbf{C}} \wedge \varphi \Rightarrow \\ [\mathbf{c!m(e)}]\varphi) \Rightarrow \\ \Box_{\mathbf{c,C}}(\mathbf{c} \in \overline{\mathbf{C}} \Rightarrow \varphi) \wedge \\ \forall \mathbf{i} : \mathbb{N}_1 \cdot (\mathbf{c} \in \overline{\mathbf{C}} \Rightarrow \varphi) \circledcirc \mathbf{c}.\downarrow(\mathbf{m_j}, \mathbf{i})$$

for each method $\mathbf{m_j}$ of $\mathbf{C}$, where $\mathbf{c} : @\mathbf{C}$.

(ii) Characterisation of asynchronous processes as sets of action instances and axioms relating these [14].

4 Morphisms and Semantics

The concept of a theory morphism for RAL is similar to that for the object calculus. A morphism $\sigma : \mathbf{Th1} \to \mathbf{Th2}$ maps each type symbol $\mathbf{T}$ of $\mathbf{Th1}$ to a type symbol $\sigma(\mathbf{T})$ of $\mathbf{Th2}$, each function symbol of $\mathbf{Th1}$ to a function symbol of $\mathbf{Th2}$, and each attribute of $\mathbf{Th1}$ to an attribute of $\mathbf{Th2}$. Actions of $\mathbf{Th1}$ are mapped to actions of $\mathbf{Th2}$.

The type $\mathbf{TIME}$ is always mapped to itself.

We must have that $\mathbf{Th}2 \vdash \sigma(\varphi)$ for each theorem φ of $\mathbf{Th}1$. In particular, the locality property of $\mathbf{Th}1$ must be true under interpretation via σ in $\mathbf{Th}2$. As in the object calculus, this will mean that actions of $\mathbf{Th}2$ not in the range of σ can only modify (the interpretations of) attributes of $\mathbf{Th}1$ by executing concurrently with (interpretations of) actions of $\mathbf{Th}1$.

We can construct a category of theories with theory morphisms as categorical arrows as usual.

A semantics of RAL can be given based on that for modal action logic in [9]. A soundness proof can be given, and a completeness proof with respect to a simplified semantics can be derived.

5 Conclusions

We have introduced a formalism for reasoning about concurrent object-oriented programs and specifications. This formalism possesses a sound semantics, and it is therefore consistent relative to ZF set theory. The advantage of the formalism over other real-time and concurrency formalisms is the conciseness of the core syntax and axiomatisation, and its ability to express the full range of reactive and real-time system behaviour via derived constructs. The TAM formalism of [15] can be regarded as a subset of RAL, and could be used to transform specification and code fragments that are purely local to one class and that are within its language. For practical development, we also need higher-level design transformations such as design patterns.

Examples of using the logic to express properties of distributed and concurrent systems can be found in the papers and books [11, 13].

References

[1] A Burns and A Wellings. HRT-HOOD: A structured design method for hard real-time systems. *Real-Time Systems*, 6(1):73–114, January 1994.

[2] Cook W. R., Palsberg J.: *A Denotational Semantics of Inheritance and its Correctness*, Proceedings of OOPSLA, pages 433-443, 1989.

[3] S Cook and J Daniels. *Designing Object Systems: Object-Oriented Modelling with Syntropy.* Prentice Hall, Sept 1994.

[4] E Durr and E Dusink. The role of VDM^{++} in the development of a real-time tracking and tracing system. In J Woodcock and P Larsen, editors, FME '93, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[5] J Fiadeiro and T Maibaum. Sometimes "Tomorrow" is "Sometime". In *Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 48–66. Springer-Verlag, 1994.

[6] J. Goguen, *Sheaf Semantics for Concurrent Interacting Objects*, Mathematical Structures in Computer Science, 11:159–191, 1992.

[7] F Jahanian, A K Mok. Safety Analysis of Timing Properties in Real-time Systems, *IEEE Transactions on Software Engineering*, SE-12, pp. 890–904, September 1986.

[8] Jones C. B.: An object-based design method for concurrent programs. Technical Report UMCS-92-12-1, Manchester University, 1993.

[9] S Kent, K Lano. *Axiomatic Semantics for Concurrent Object Systems*, AFRODITE Technical Report AFRO/IC/SKKL/SEM/V1, Dept. of Computing, Imperial College, 180 Queens Gate, London SW7 2BZ.

[10] L Lamport. The Temporal Logic of Actions, Technical Report 79, Digital Equipment Corporation, Systems Research Center, December 1991.

[11] Lano K., *Formal Object-oriented Development*, FACIT series, Springer-Verlag, 1995.

[12] Lano K., *Reactive System Specification and Refinement*, Proceedings of TAPSOFT '95, Springer-Verlag LNCS 915, 1995.

[13] Lano K., *Distributed System Specification in VDM^{++}*, FORTE '95 Proceedings, Chapman and Hall, 1995.

[14] Lano K., Goldsack S., *Refinement Rules for Concurrency and Real-time*, MEDICIS Workshop, April 1996.

[15] G Lowe and H Zedan. Refinement of complex systems: A case study. *The Computer Journal*, 38(10):785–800, 1995.

[16] Liskov B., Data abstraction and hierarchy. In *OOPSLA '87 Conference Proceedings*, 1987.

[17] Malcolm G.: *Interconnections of Object Specifications*, Proceedings of BCS FACS Workshop on Formal Methods and Object-orientation, to appear, Springer-Verlag, 1995.

[18] Maung I.: *Behavioural Subtyping and Substitutability*, Proceedings of BCS FACS Workshop on Formal Methods and Object-orientation, to appear, Springer-Verlag, 1995.

[19] A Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In J de Bakker, W P de Roever, and G Rozenberg, editors, *Current Trends in Concurrency*, LNCS vol. 224, Springer-Verlag, 1986.