Structured Axiomatic Semantics for UML Models

K. Lano
Dept. of Computer Science,
King's College London,
Strand, London, WC2R 2LS
kcl@dcs.kcl.ac.uk
J. Bicarregui
Rutherford Appleton Laboratory
Oxford OX11 0QX
A. Evans
Dept. of Computing,
University of York
andye@cs.york.ac.uk

Abstract. In this paper we provide a systematic formal interpretation for most elements of the UML notation. This interpretation, in a structured temporal logic, enables precise analysis of the properties of these models, and the verification of one model against another. We extend previous work by providing a structured logical interpretation for sequence diagrams, in which object communication is represented using theory morphisms. As an application of the formalisation, we show how the introduction of particular design patterns can be proved to be refinement transformations.

1 Introduction

The UML [10] combines and extends elements of previous OO notations such as OMT, Booch and Objectory. In contrast to these methods, its notations are precisely defined using the Object Constraint Language (OCL) and a meta-model to express the allowed forms of diagrams and their properties. Detailed syntactic description and constraints on model structures are given in [11]. However the semantics of model elements are only given via natural language. As a result, many ambiguities remain. For example: whether objects may be recreated at different times with the same identity; in what order the entry and exit actions of concurrently entered/exited states are performed, and so forth. Here we will use a formal framework to express alternatives for these semantic choices.

2 Outline of Semantics

A mathematical semantic representation of UML models can be given in terms of *theories* in extended first-order set theory as in the semantics presented for

Syntropy in [2] and VDM^{++} in [8]. In order to reason about real-time properties of systems the Real-time Action Logic (RAL) of [8] will be used.

A RAL theory has the form:

theory *Name*

types *local type symbols*

attributes *time-varying data, representing instance or class variables*

actions *actions which may affect the data, such as operations, statechart transitions and methods*

axioms *logical properties and constraints between the theory elements.*

Theories can be used to represent classes, instances, associations and general submodels of a UML model. These models are therefore being understood as *specifications*: they describe the features and properties which should be supported by any implementation that satisfies the model (equivalently, any structure that satisfies the axioms of its theory). An important relationship between theories is that of logical consequence – theory $\mathbf{S}$ *satisfies* (the properties of) theory $\mathbf{T}$ if there is an interpretation σ of the symbols of $\mathbf{T}$ into those of $\mathbf{S}$ under which every property of $\mathbf{T}$ holds:

$$\mathbf{S} \vdash \sigma(\varphi)$$

for every theorem φ of $\mathbf{T}$. This has the effect that any structure satisfying the axioms of $\mathbf{S}$ will also satisfy those of $\mathbf{T}$. A design model $\mathbf{D}$ with theory $\mathbf{S}$ can be considered a correct refinement of an abstract (specification) model $\mathbf{C}$ with theory $\mathbf{T}$ if $\mathbf{S}$ satisfies $\mathbf{T}$.

In addition to Z-style mathematical notation such as $\mathbb{F}$ for "set of finite sets of", $\mathbf{r}^{-1}$ for relational inverse and $\mathbf{r}(\!|\ \mathbf{S}\ |\!)$ for relational image, etc, RAL theories can use the following notations:

1. For each classifier or state $\mathbf{X}$ there is an attribute $\overline{\mathbf{X}} : \mathbb{F}(\mathbf{X})$ denoting the set of existing instances of $\mathbf{X}$. This represents deep equality between objects in the sense that if $\mathbf{x}, \mathbf{y} \in \overline{\mathbf{X}}$ and $\mathbf{x} = \mathbf{y}$ then not only $\mathbf{x}.\mathbf{att} = \mathbf{y}.\mathbf{att}$ for all attributes of $\mathbf{X}$, but also recursively for attributes of $\mathbf{x}.\mathbf{att}$ and $\mathbf{y}.\mathbf{att}$, etc.

2. If α is an action symbol, and $\mathbf{P}$ a predicate, then $[\alpha]\mathbf{P}$ is a predicate which means "every execution of α establishes $\mathbf{P}$ on termination", that is, $\mathbf{P}$ is a *postcondition* of α.

3. For every action α there are functions $\uparrow(\alpha, \mathbf{i})$, $\downarrow(\alpha, \mathbf{i})$, $\leftarrow(\alpha, \mathbf{i})$ and $\rightarrow(\alpha, \mathbf{i})$ of $\mathbf{i} : \mathbb{N}_1$ which denote the activation, termination, request send and request arrival times, respectively, of the $\mathbf{i}$-th invocation of α. These times are ordered as:

$$\leftarrow(\alpha, \mathbf{i}) \ \leq \ \rightarrow(\alpha, \mathbf{i}) \ \leq \ \uparrow(\alpha, \mathbf{i}) \ \leq \ \downarrow(\alpha, \mathbf{i})$$

Also

$$\mathbf{i} \leq \mathbf{j} \;\Rightarrow\; \leftarrow(\alpha, \mathbf{i}) \;\leq\; \leftarrow(\alpha, \mathbf{j})$$

4. If α and β are actions, then $\alpha \,||\, \beta$ is the action γ such that

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot \exists \mathbf{j}, \mathbf{k} : \mathbb{N}_1 \cdot \uparrow(\gamma, \mathbf{i}) = \mathbf{min}(\uparrow(\alpha, \mathbf{j}), \uparrow(\beta, \mathbf{k})) \;\wedge$$
$$\downarrow(\gamma, \mathbf{i}) = \mathbf{max}(\downarrow(\alpha, \mathbf{j}), \downarrow(\beta, \mathbf{k}))$$

5. If α and β are actions, then $\alpha \;\supset\; \beta$ "α calls β" is defined to mean that

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

This definition is used for consistency with the object calculus $\alpha \Rightarrow \beta$ "every execution of α co-executes with one of β". It ensures that $[\beta]\mathbf{P} \;\Rightarrow\; [\alpha]\mathbf{P}$.

6. If α, β are actions, then $\alpha \sqcap \beta$ is the action γ whose execution instances are either an execution instance of α or of β:

$$\forall \mathbf{i} : \mathbb{N}_1 \cdot (\exists \mathbf{j} : \mathbb{N}_1 \cdot \uparrow(\gamma, \mathbf{i}) = \uparrow(\alpha, \mathbf{j}) \wedge \downarrow(\gamma, \mathbf{i}) = \downarrow(\alpha, \mathbf{j})) \;\vee$$
$$(\exists \mathbf{k} : \mathbb{N}_1 \cdot \uparrow(\gamma, \mathbf{i}) = \uparrow(\beta, \mathbf{k}) \wedge \downarrow(\gamma, \mathbf{i}) = \downarrow(\beta, \mathbf{k}))$$

7. Counters $\#\mathbf{act}(\alpha)$, $\#\mathbf{fin}(\alpha)$ and $\#\mathbf{req}(\alpha)$ count the number of initiations, terminations and requests, respectively, for invocations of α. $\#\mathbf{waiting}(\alpha) = \#\mathbf{req}(\alpha) - \#\mathbf{act}(\alpha)$ and $\#\mathbf{active}(\alpha) = \#\mathbf{act}(\alpha) - \#\mathbf{fin}(\alpha)$.

Either Z or OCL notation could be used for axioms in theories, representing the semantics or constraints of UML models. In [7] we define a translation from OCL into Z.

2.1 Object Models

A UML class $\mathbf{C}$ is represented as a theory $\Gamma_{\mathbf{C}}$ of the form given in Figure 1. The write frame of an action (the set of attributes that it may change) is written after its declaration.

Each instance attribute $\mathbf{att_i} : \mathbf{T_i}$ of $\mathbf{C}$ gains an additional parameter of type $\mathbf{C}$ in the class theory $\Gamma_{\mathbf{C}}$ and similarly for operations. Thus the **self** attribute becomes the identity function on object identifiers[1]. Class attributes and actions do not gain the additional $\mathbf{C}$ parameter as they are independent of any particular instance. The standard OO notation $\mathbf{a.att}$ will be used as an alternative for $\mathbf{att(a)}$ for attribute **att** of instance $\mathbf{a}$ and similarly $\mathbf{a.act(x)}$ will be used for actions $\mathbf{act(a, x)}$. $\Gamma_{\mathbf{C}}$ includes $\Gamma_{\mathbf{S}}$ for each supplier $\mathbf{S}$ of $\mathbf{C}$.

Similarly each association **lr** can be interpreted in a theory which contains an attribute $\overline{\mathbf{lr}}$ representing the current extent of the association (the set of

[1] The class theory can be derived from a theory $\mathcal{I}_{\mathbf{C}}$ of a typical $\mathbf{C}$ instance by means of an $\mathbf{A}$-morphism [2].

$$
\begin{array}{l}
\textbf{theory } \Gamma_{\mathbf{C}} \\
\textbf{types } \mathbf{C} \\
\textbf{attributes } \overline{\mathbf{C}} : \mathbb{F}(\mathbf{C}) \\
\quad \mathbf{self} : \mathbf{C} \to \mathbf{C} \\
\quad \mathbf{att}_1 : \mathbf{C} \to \mathbf{T}_1 \\
\quad \ldots \\
\textbf{actions } \mathbf{create}_{\mathbf{C}}(\mathbf{c} : \mathbf{C}) \quad \{\overline{\mathbf{C}}\} \\
\quad \mathbf{kill}_{\mathbf{C}}(\mathbf{c} : \mathbf{C}) \quad \{\overline{\mathbf{C}}\} \\
\quad \mathbf{op}_1(\mathbf{c} : \mathbf{C}, \mathbf{x} : \mathbf{X}_1) : \mathbf{Y}_1 \\
\quad \ldots \\
\textbf{axioms} \\
\qquad \forall \mathbf{c} : \mathbf{C} \cdot \\
\qquad\qquad \mathbf{self}(\mathbf{c}) = \mathbf{c} \ \wedge \\
\qquad\qquad [\mathbf{create}_{\mathbf{C}}(\mathbf{c})](\mathbf{c} \in \overline{\mathbf{C}}) \ \wedge \\
\qquad\qquad [\mathbf{kill}_{\mathbf{C}}(\mathbf{c})](\mathbf{c} \notin \overline{\mathbf{C}})
\end{array}
$$

Fig. 1. Theory $\Gamma_{\mathbf{C}}$ of Class **C**

pairs of objects in it) and actions **add_link** and **delete_link** to add and remove links from this set. Axioms define the cardinality of the association ends and other properties of the association. In particular, if **ab** is an association between classes **A** and **B**, then $\overline{\mathbf{ab}} \subseteq \overline{\mathbf{A}} \times \overline{\mathbf{B}}$, so membership of $\overline{\mathbf{ab}}$ implies existence for the object instances at the ends of the link.

2.2 Statecharts

A statechart specification of the behaviour of instances of a class **C** can be formalised as an extension of the class theory of **C**, as follows.

If **M** is a UML **StateMachine**, then its set of states, $\mathbf{States}_{\mathbf{M}}$, consist of **M.top** (in [11]) and all states (recursively) linked to **M.top** via **subvertex** links – ie., all substates of the top state of **M**.

1. If **M** is the statemachine linked to a class **C**, each state $\mathbf{S} : \mathbf{States}_{\mathbf{M}}$ is represented in the same manner as a subclass of **C**, and in general, nesting of state $\mathbf{S}_1$ in state $\mathbf{S}_2$ is expressed by axioms $\mathbf{S}_1 \ll \mathbf{S}_2$ ($\mathbf{S}_1$ is a subtype of $\mathbf{S}_2$) and $\overline{\mathbf{S}_1} \subseteq \overline{\mathbf{S}_2}$ as for class generalisation.

   If two states are exclusive (ie, they are not related by nesting) then the corresponding $\overline{\mathbf{S}}$ sets are axiomatised as disjoint.

   The subclasses corresponding to states are usually dynamic (Syntropy [1] also treats statechart states as dynamic subclasses): an object instance **x** may move from one state **S** of **C** to another $\mathbf{S}'$ as a result of an action. This is expressed by changes to the $\overline{\mathbf{S}}$ and $\overline{\mathbf{S}'}$ attributes: the deletion of **x** from $\overline{\mathbf{S}}$ and its addition to $\overline{\mathbf{S}'}$.

2. The set of transitions $\mathbf{Trans_M}$ of $\mathbf{M}$ is $\mathbf{M.transition}$ in the UML metamodel. The set of events $\mathbf{Events_M}$ of $\mathbf{M}$ is $\mathbf{M.transition.trigger}$. Each element of $\mathbf{Trans_M} \cup \mathbf{Events_M}$ is represented by a distinct action symbol. Each event $\mathbf{e}$ is the abstract generalisation of the actions $\mathbf{t}_1, \ldots, \mathbf{t_n}$ representing its transitions:

   $$\forall \mathbf{a} : \mathbf{C} \cdot \mathbf{a.t}_1 \supset \mathbf{a.e} \ \wedge \ \ldots \ \wedge \ \mathbf{a.t_n} \supset \mathbf{a.e}$$

   where the $\mathbf{t_i}$ are all transitions in the statechart whose trigger event (in the sense of the UML semantics [11]) is $\mathbf{e}$.

3. The axiom for the effect of a transition $\mathbf{t}$ from state $\mathbf{S}_1$ to state $\mathbf{S}_2$ with label

   $$\mathbf{e(x)[G]/Post} \frown \mathbf{Act}$$

   where $\mathbf{G}$ is the guard condition and $\mathbf{Post}$ is some postcondition constraint on the resulting state, is

   $$\forall \mathbf{a} : \mathbf{C} \cdot \mathbf{a.G} \wedge \mathbf{a} \in \overline{\mathbf{S}_1} \ \Rightarrow \ [\mathbf{a.t(x)}](\mathbf{a.Post} \wedge \mathbf{a} \in \overline{\mathbf{S}_2})$$

4. The transition only occurs if the trigger event occurs whilst the object is in the correct state:

   $$\forall \mathbf{a} : \mathbf{C} \cdot \mathbf{a} \in \overline{\mathbf{S}_1} \wedge \mathbf{a.G} \ \Rightarrow \ (\mathbf{a.e(x)} \supset \mathbf{a.t(x)})$$

   We assume that distinct transitions from the same source state have non-overlapping guard conditions.

5. Asynchronously generated actions must occur at some future time (after $\mathbf{t}$ has occurred):

   $$\mathbf{a.t(x)} \ \Rightarrow \ \bigcirc \diamond (\mathbf{a.Act}_1 \wedge \diamond(\ldots \diamond \mathbf{a.Act_m}) \ldots)$$

   $\bigcirc$ is the "next" operator of temporal logic, interpreted as "next method execution initiation time" in RAL, $\diamond$ is the "eventually" operator. $\mathbf{Act}$ is the list $\mathbf{Act}_1 \frown \ldots \frown \mathbf{Act_m}$ of generated actions of $\mathbf{t}$, ie $\mathbf{t.effect}$ in the sense of [11] (**ActionSequence** is a subclass of **Action** in the UML metamodel).

6. Synchronously generated actions have the axiom:

   $$\mathbf{a.t(x)} \ \supset \ \mathbf{a.Act}_1; \ \ldots; \ \mathbf{a.Act_m}$$

We can provide a semantics for general UML statechart models by a series of transformations into a smaller statechart language (Section 5) and then apply the above axiomatisation. However to simplify verification of critical systems, we have developed a smaller statechart language with strong modularity and scoping restrictions on message sending [9] which uses the same semantics.

3 The Core Package

3.1 Association

An association $\mathbf{r}$ with name **rname**, and linked classifiers $\mathbf{r}$.**connection**.**type** $=$ $\langle \mathbf{C}_1, \ldots, \mathbf{C_n} \rangle$ (**connection** gives the **AssociationEnd** elements of $\mathbf{r}$, **type** the classifiers connected to these ends, [11, page 2-15]) is formalised by an attribute

$$\overline{\mathbf{r}} : \mathbb{F}(\mathbf{C}_1 \times \ldots \times \mathbf{C_n})$$

such that

$$\overline{\mathbf{r}} \subseteq \overline{\mathbf{C}_1} \times \ldots \times \overline{\mathbf{C_n}}$$

That is, the elements of $\overline{\mathbf{r}}$ are tuples $(\mathbf{c}_1, \ldots, \mathbf{c_n})$ of elements of the classifiers that it links, where each of the $\mathbf{c_i}$ is an existing instance.

Subsequently, we will only consider binary associations, as more general n-ary associations can be transformed into $\mathbf{n}$ binary associations together with a new class and logical constraints[2].

3.2 Association Class

A theory representing an association class $\mathbf{C}$ has an attribute $\overline{\mathbf{r}}$ describing the extent of the association, and an attribute $\overline{\mathbf{C}}$ describing the extent of the class. We require that these are isomorphic at all times, ie:

$$\mathbf{card}(\overline{\mathbf{C}}) = \mathbf{card}(\overline{\mathbf{r}})$$

This is ensured if there is a function $\mathbf{i} : \mathbf{C} \to \mathbf{C}_1 \times \mathbf{C}_2$ such that $\mathbf{i}$ is an isomorphism[3] between $\overline{\mathbf{C}}$ and $\overline{\mathbf{r}}$ and such that

$$\begin{array}{l}
\mathbf{create_link_r}(\mathbf{a}, \mathbf{b}) \supset \mathbf{create_C}(\mathbf{i}^{-1}(\mathbf{a}, \mathbf{b})) \\
\mathbf{delete_link_r}(\mathbf{a}, \mathbf{b}) \supset \mathbf{kill_C}(\mathbf{i}^{-1}(\mathbf{a}, \mathbf{b})) \\
\mathbf{create_C}(\mathbf{x}) \supset \mathbf{create_link_r}(\mathbf{i}(\mathbf{x})) \\
\mathbf{kill_C}(\mathbf{x}) \supset \mathbf{delete_link_r}(\mathbf{i}(\mathbf{x}))
\end{array}$$

3.3 Association End

Assume a binary association $\mathbf{r}$ with linked classifiers $\mathbf{C}_1$ (target) and $\mathbf{C}_2$ (source) and name **rname** (Figure 2).

---

[2] If $\mathbf{r}$ is an $\mathbf{n}$-ary association between classes $\mathbf{C}_1, \ldots, \mathbf{C_n}$, then it can be replaced by a class $\mathbf{C}$ and $\mathbf{n}$ many-one associations $\mathbf{r}_1, \ldots, \mathbf{r_n}$ from $\mathbf{C}$ to $\mathbf{C}_1, \ldots, \mathbf{C_n}$ respectively, such that $\exists\, \mathbf{c} : \overline{\mathbf{C}} \cdot (\mathbf{x}_1, \mathbf{c}) \in \overline{\mathbf{r}_1} \wedge \ldots \wedge (\mathbf{x_n}, \mathbf{c}) \in \overline{\mathbf{r_n}}$ is equivalent to $(\mathbf{x}_1, \ldots, \mathbf{x_n}) \in \overline{\mathbf{r}}$.

[3] Notice that we cannot take $\mathbf{C} = \mathbf{C}_1 \times \mathbf{C}_2$ as there may be two association classes between these two classes, thus creating confusion over which one a particular association object (object pair) belongs to.

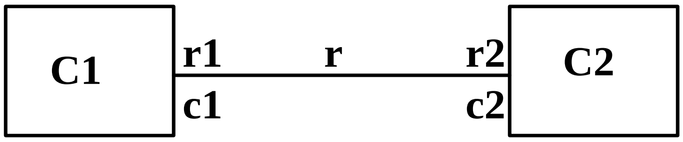

Fig. 2. Typical Binary Association

Aggregation If this meta-attribute of **AssociationEnd** is set to **none** there are no extra axioms.

If it is set to **aggregate** a CASE tool should check that the source end of the association has aggregation value **none** if the target end has value **aggregate**. In addition **r** must be transitive and irreflexive (page 2-58 of [11]).

If the meta-attribute is set to **composite** then **r** has the specific properties:

1. One-many (page 2-21 of [11]):

$$\mathbf{x} \in \overline{\mathbf{C}_1} \wedge \mathbf{x}' \in \overline{\mathbf{C}_1} \wedge (\mathbf{x}, \mathbf{y}) \in \overline{\mathbf{r}} \wedge (\mathbf{x}', \mathbf{y}) \in \overline{\mathbf{r}} \;\Rightarrow\; \mathbf{x} = \mathbf{x}'$$

2. Deletion propogating (page 2-57 of [11]):

$$\mathbf{x} \in \overline{\mathbf{C}_1} \wedge (\mathbf{x}, \mathbf{y}) \in \overline{\mathbf{r}} \;\Rightarrow\; \mathbf{kill}_{\mathbf{C}_1}(\mathbf{x}) \;\supset\; \mathbf{kill}_{\mathbf{C}_2}(\mathbf{y})$$

3. Transitivity (page 2-58 of [11]):

$$(\mathbf{x}, \mathbf{y}) \in \overline{\mathbf{r}} \wedge (\mathbf{y}, \mathbf{z}) \in \overline{\mathbf{r}} \;\Rightarrow\; (\mathbf{x}, \mathbf{z}) \in \overline{\mathbf{r}}$$

4. Irreflexivity (page 2-58 of [11]):

$$\mathbf{x} \in \overline{\mathbf{C}_1} \;\Rightarrow\; \mathbf{not}((\mathbf{x}, \mathbf{x}) \in \overline{\mathbf{r}})$$

Changeable If this meta-attribute has value **frozen**, then the set of $\mathbf{C}_1$ objects linked to a particular $\mathbf{C}_2$ object cannot change while the latter exists:

$$\mathbf{y} \in \overline{\mathbf{C}_2} \wedge \mathbf{y} \in \bigcirc\overline{\mathbf{C}_2} \;\Rightarrow\; \overline{\mathbf{r}}^{-1}(\!|\, \{\mathbf{y}\} \,|\!) \;=\; \bigcirc\overline{\mathbf{r}}^{-1}(\!|\, \{\mathbf{y}\} \,|\!)$$

An alternative interpretation could be that the set of *existing* $\mathbf{C}_1$ objects cannot change, but these can be deleted:

$$\mathbf{y} \in \overline{\mathbf{C}_2} \wedge \mathbf{y} \in \bigcirc\overline{\mathbf{C}_2} \;\Rightarrow\; \overline{\mathbf{r}}^{-1}(\!|\, \{\mathbf{y}\} \,|\!) \cap \bigcirc\overline{\mathbf{C}_1} \;=\; \bigcirc\overline{\mathbf{r}}^{-1}(\!|\, \{\mathbf{y}\} \,|\!)$$

If this is **addOnly** then the above constraints are weakened with $\subseteq$ in place of $=$.

IsOrdered If this meta-attribute has value **true**, there is an additional attribute

$$\mathbf{ord}_{\mathbf{C}_1,\mathbf{r}} : \mathbf{C}_2 \to \mathrm{seq}(\mathbf{C}_1)$$
$$\mathbf{y} \in \overline{\mathbf{C}_2} \;\Rightarrow\; \overline{\mathbf{r}}^{-1}(\!|\, \{\mathbf{y}\} \,|\!) = \mathrm{ran}(\mathbf{ord}_{\mathbf{C}_1,\mathbf{r}}(\mathbf{y}))$$

In other words, **ord** gives an ordering to the sets of $\mathbf{C}_1$ elements linked to each $\mathbf{y} \in \overline{\mathbf{C}_2}$.

IsNavigable If this meta-attribute has value **true**, it implies a tool check that messages are only sent (in statecharts, sequence diagrams, activity diagrams and collaboration diagrams) along the association in a navigable direction.

Multiplicity A cardinality restriction can be interpreted as a subset of $\mathbb{N}$. For example, $1, 5\,..\,7, 9\,..\,*$ defines the set

$$\{1,5,6,7\} \cup \{\mathbf{n} : \mathbb{N} \mid \mathbf{n} \geq 9\}$$

Hence a cardinality restriction $\mathbf{c}_1 : \mathbb{P}(\mathbb{N})$ at the $\mathbf{C}_1$ end of $\mathbf{r}$ yields the axiom:

$$\mathbf{y} \in \overline{\mathbf{C}_2} \;\Rightarrow\; \mathbf{card}(\overline{\mathbf{r}}^{-1}(\!|\ \{\mathbf{y}\}\ |\!)) \in \mathbf{c}_1$$

Name If the role name attached to $\mathbf{C}_1$ is $\mathbf{r}_1$, then there is an attribute

$$\begin{aligned}&\mathbf{r}_1 : \mathbf{C}_2 \to \mathbb{F}(\mathbf{C}_1)\\ &\mathbf{y} \in \overline{\mathbf{C}_2} \;\Rightarrow\; \mathbf{r}_1(\mathbf{y}) \;=\; \overline{\mathbf{r}}^{-1}(\!|\ \{\mathbf{y}\}\ |\!)\end{aligned}$$

In the case that the cardinality restriction at the $\mathbf{C}_1$ end is 1 (ie, $\mathbf{c}_1 = \{1\}$) we can define instead

$$\begin{aligned}&\mathbf{r}_1 : \mathbf{C}_2 \to \mathbf{C}_1\\ &\mathbf{y} \in \overline{\mathbf{C}_2} \;\Rightarrow\; \{\mathbf{r}_1(\mathbf{y})\} \;=\; \overline{\mathbf{r}}^{-1}(\!|\ \{\mathbf{y}\}\ |\!)\end{aligned}$$

3.4 Attribute

If an attribute of classifier **C** has name **att** and type **T**, has **instance** scope and multiplicity 1, then it is expressed as an attribute symbol

$$\mathbf{att} : \mathbf{C} \to \mathbf{T}$$

and may be written as $\mathbf{att}(\mathbf{c})$ or $\mathbf{c.att}$ for specific $\mathbf{c} \in \overline{\mathbf{C}}$.

In the case that the **ownerScope** is **classifier** then there is no need for a **C** parameter:

$$\mathbf{att} : \mathbf{T}$$

since the same value is shared by all instances of **C**.

If the multiplicity constraint **c** is not $\{1\}$ then **att** is represented as an attribute symbol

$$\mathbf{att} : \mathbf{C} \to \mathrm{seq}(\mathbf{T})$$

where $\mathbf{size}(\mathbf{att}(\mathbf{x})) \in \mathbf{c}$ for each $\mathbf{x} \in \overline{\mathbf{C}}$. Similarly for multiple classifier scope attributes.

Changeable If this meta-attribute is **frozen** then we have the axiom

$$\mathbf{x} \in \overline{\mathbf{C}} \wedge \mathbf{x} \in \bigcirc\overline{\mathbf{C}} \;\Rightarrow\; \mathbf{att}(\mathbf{x}) = \bigcirc\mathbf{att}(\mathbf{x})$$

A logically stronger version is

$$\exists\, \mathbf{v} : \mathbf{T} \cdot \Box^{\tau}(\mathbf{x} \in \overline{\mathbf{C}} \;\Rightarrow\; \mathbf{att}(\mathbf{x}) = \mathbf{v})$$

where $\Box^{\tau}$ means "at all future times" in contrast to $\Box$ which means "at all future method initiation times".

This version implies that even if an object is 'reborn' then it always has the same value for **att** even in discontinuous portions of its life.

If this is **addOnly** then

$$\mathbf{x} \in \overline{\mathbf{C}} \wedge \mathbf{x} \in \bigcirc\overline{\mathbf{C}} \;\Rightarrow\; \mathrm{ran}(\mathbf{att}(\mathbf{x})) \subseteq \mathrm{ran}(\bigcirc\mathbf{att}(\mathbf{x}))$$

Initial Value This defines the value set at creation:

$$[\mathbf{create_C}(\mathbf{x})](\mathbf{att}(\mathbf{x}) = \mathbf{initval})$$

3.5 Behavioural Feature

These are expressed as action symbols. If behavioural feature $\mathbf{f}$ of classifier $\mathbf{C}$ has parameters $\mathbf{p}_1 : \mathbf{T}_1, \ldots, \mathbf{p_m} : \mathbf{T_m}$ then it is represented as an action symbol $\mathbf{f}(\mathbf{C}, \mathbf{T}_1, \ldots, \mathbf{T_m})$ in the case of **instance** scope, or $\mathbf{f}(\mathbf{T}_1, \ldots, \mathbf{T_m})$ in the case of **classifier** scope.

IsQuery If this is true, then the write frame of $\mathbf{f}$ is the empty set.

3.6 Constraint

These are interpreted (where possible) as predicates in the theory of the smallest model containing all model elements that they constrain. They are true at all times in the history of an instance of such a model.

3.7 Data Type

These are represented by data types in our logic. A **utility** class $\mathbf{C}$ is one all of whose attributes and actions are of class scope (page 2-28 of [11]).

3.8 Feature

The **ownerScope** of a feature is represented as explained in Sections 3.4 and 3.5 above.

The **visibility** of an attribute is not distinguished in our semantics and requires checks by CASE tools for the notation.

3.9 Generalisable Element

If a classifier $\mathbf{C}$ has **isAbstract** true, and $\mathbf{C}_1, \ldots, \mathbf{C_n}$ are all its immediate descendants, then:

$$\mathbf{create_C(x)} \supset \mathbf{create_{C_1}(x)} \sqcap \ldots \sqcap \mathbf{create_{C_n}(x)}$$

In other words, creation of an instance of $\mathbf{C}$ implies it is actually created as an instance of one of $\mathbf{C}$'s proper subclassifiers.

In addition

$$\bigwedge_{\mathbf{j \neq i}} \neg\, \mathbf{create_{C_j}(x)} \;\wedge\; \bigwedge_{\mathbf{j \neq i}} \mathbf{x} \notin \overline{\mathbf{C_j}} \;\Rightarrow\; \mathbf{kill_{C_i}(x)} \supset \mathbf{kill_C(x)}$$

Together with the normal axioms $\mathbf{create_{C_i}(x)} \supset \mathbf{create_C(x)}$ for generalisation, these establish by induction that $\overline{\mathbf{C}} = \overline{\mathbf{C}_1} \cup \ldots \cup \overline{\mathbf{C_n}}$ at all times.

The **isLeaf** and **isRoot** meta-attributes declare whether the classifier cannot or can allow redefinition of its response to signals in descendants. These will not be represented in the semantics as their treatment is primarily a CASE tool issue.

3.10 Generalisation

If $\mathbf{T}$ is a generalisation of $\mathbf{S}$ then

$$\mathbf{S} \ll \mathbf{T} \wedge \overline{\mathbf{S}} \subseteq \overline{\mathbf{T}}$$

This means that any feature of $\mathbf{T}$ is also defined for $\mathbf{S}$.

The second formula is ensured by axioms:

$$\mathbf{create_S(x)} \supset \mathbf{create_T(x)}$$
$$\mathbf{x} \in \overline{\mathbf{S}} \Rightarrow \mathbf{kill_T(x)} \supset \mathbf{kill_S(x)}$$

In other words, if $\mathbf{x}$ is added to $\overline{\mathbf{S}}$ it must also be added to $\overline{\mathbf{T}}$, and if $\mathbf{x}$ exists as a subclass instance, then removing it from $\overline{\mathbf{T}}$ must also remove it from $\overline{\mathbf{S}}$.

A theory $\Gamma_{\mathbf{S}}$ for $\mathbf{S}$ can be defined as an extension of the theory $\Gamma_{\mathbf{T}}$ for $\mathbf{T}$ with these extra data types attributes and axioms, together with the attributes, actions and axioms derived from the declarations contained in the text of $\mathbf{S}$. If *locality* axioms

$$\alpha_1 \vee \ldots \vee \alpha_{\mathbf{p}} \vee \mathbf{att} = \bigcirc \mathbf{att}$$

are included in $\Gamma_{\mathbf{T}}$ however, for each attribute $\mathbf{att}$ of $\mathbf{T}$, the $\alpha_{\mathbf{i}}$ being all actions with $\mathbf{att}$ in their write frame, then $\Gamma_{\mathbf{S}}$ in general does not satisfy the axioms of $\Gamma_{\mathbf{T}}$: the locality axiom for $\mathbf{att}$ is only true in $\Gamma_{\mathbf{S}}$ if any action declared in $\mathbf{S}$ only modifies $\mathbf{att}$ by invoking one of the $\alpha_{\mathbf{i}}$ from $\mathbf{T}$ – so called 'strict' inheritance, which provides a form of semi-private scoping (this is also the semantics of INCLUDES in B [5]).

3.11 Operation

Operations are represented by action symbols.

Concurrency If this meta-attribute is **sequential** then there is at most one invocation of the operation, $\mathbf{m}$, of classifier $\mathbf{C}$, executing or waiting to be executed:

$$\mathbf{a} \in \overline{\mathbf{C}} \Rightarrow \#\mathbf{waiting}(\mathbf{m}(\mathbf{a})) + \#\mathbf{active}(\mathbf{m}(\mathbf{a})) \leq 1$$

If this is **guarded** then there can be many waiting invocation requests for $\mathbf{m}$, but only one active:

$$\mathbf{a} \in \overline{\mathbf{C}} \Rightarrow \#\mathbf{active}(\mathbf{m}(\mathbf{a})) \leq 1$$

There are no restrictions on **concurrent** operations.

IsPolymorphic This meta-attribute indicates that the operation is polymorphic. Any polymorphic use of an operation should be checked against the value of this by CASE tools; it is not represented in the semantics.

Specification If this is expressed as code or as pre/post conditions, then it can be formalised as an action symbol definition.

3.12 Qualifier

If association $\mathbf{r}$ between classifiers $\mathbf{C}_1$ and $\mathbf{C}_2$ has qualifier attributes $\mathbf{q}_1 : \mathbf{T}_1$, $\ldots$, $\mathbf{q_k} : \mathbf{T_k}$ at the $\mathbf{C}_1$ end, and cardinality constraint $\mathbf{c}$ at the $\mathbf{C}_2$ end, then we have the axiom:

$$\forall \mathbf{x} : \overline{\mathbf{C}_1};\ \mathbf{v}_1 : \mathbf{T}_1;\ \ldots;\ \mathbf{v_k} : \mathbf{T_k} \cdot \\ \mathbf{card}(\{\mathbf{y} : \mathbf{C}_2 \mid (\mathbf{x},\mathbf{y}) \in \overline{\mathbf{r}} \wedge \mathbf{q}_1(\mathbf{x}) = \mathbf{v}_1 \wedge \ldots \wedge \mathbf{q_k}(\mathbf{x}) = \mathbf{v_k}\}) \in \mathbf{c}$$

4 Behavioural Elements Package: Common Behaviour

4.1 Action

An action is represented as an action symbol. If an integer recurrence $\mathbf{n}$ is specified then this describes an n-fold iteration of the named action, where this is the sequential composition of $\mathbf{n}$ copies of the action.

The **target** may indicate a set $\mathbf{s}$ of objects instead of a single object. In this case $\mathbf{s}.\alpha$ is interpreted as the concurrent composition

$$||_{\mathbf{x} \in \mathbf{s}} \mathbf{x}.\alpha$$

of the individual actions: this composition allows the individual actions to be performed in an arbitrary order, and to overlap in their executions.

4.2 Call Action

Calls are represented by the operator $\supset$ between actions in the case of a sequential call:

$$\alpha \supset \beta$$

meaning that every invocation instance $(\alpha, \mathbf{i})$ of α coincides in time with some invocation instance of β.

In the case of an asynchronous call the invoked action can take place at some future time: $\alpha \Rightarrow \diamond\beta$.

4.3 Create Action

For a classifier $\mathbf{C}$ the create action is $\mathbf{create_C}(\mathbf{C})$.

4.4 Destroy Action

For a classifier $\mathbf{C}$ this is $\mathbf{kill_C}(\mathbf{C})$.

4.5 Instance

An instance $\mathbf{a}$ of a classifier $\mathbf{C}$ is represented as a member of the extension $\overline{\mathbf{C}}$ of $\mathbf{C}$: $\mathbf{a} \in \overline{\mathbf{C}}$.

4.6 Link

A link is represented as a particular pair $(\mathbf{x}, \mathbf{y})$ of elements in the extent $\overline{\mathbf{r}}$ of the association to which it belongs.

5 Reductive Transformations

In order to simplify the semantic treatment of statecharts, we assume that the following reductive transformations have been applied to eliminate nesting (OR-composition of states), concurrent composition (AND-composition) and entry and exit actions. The restrictions are that the original statechart must not contain deferred events or history entry states or conditions depending upon attributes (as opposed to states).

5.1 A: Eliminating Nesting

States can be *nested*, that is, a state $\mathbf{s}$ of a statechart $\mathbf{A}$ may enclose a statechart $\mathbf{smach_A}(\mathbf{s})$. If a statechart $\mathbf{B}$ is nested within a state $\mathbf{s}$ of statechart $\mathbf{A}$, ie, $\mathbf{B} = \mathbf{smach_A}(\mathbf{s})$, we can eliminate the nesting to produce a statechart or state machine $\mathbf{A'}$ by replacing $\mathbf{s}$ by $\mathbf{B}$ within $\mathbf{A}$, adding transitions $\mathbf{t_x} : \mathbf{x} \rightarrow \mathbf{T}$ for each state $\mathbf{x}$ of $\mathbf{B}$ for each original transition $\mathbf{t} : \mathbf{s} \rightarrow \mathbf{T}$ in $\mathbf{A}$ (if $\mathbf{T} \neq \mathbf{s}$), where

$$\neg\ \exists \mathbf{y} : \mathbf{States_B};\ \mathbf{tr} : \mathbf{Trans_A} \cdot \mathbf{x} \sqsubseteq \mathbf{y}\ \wedge\ \mathbf{event_A}(\mathbf{tr}) = \mathbf{event_A}(\mathbf{t})\ \wedge \\ \mathbf{y} = \mathbf{source_A}(\mathbf{tr})$$

$\mathbf{x} \sqsubseteq \mathbf{y}$ denotes that $\mathbf{x}$ is a substate of $\mathbf{y}$.

Transitions $\mathbf{t} : \mathbf{s'} \rightarrow \mathbf{s}$ in the original model ($\mathbf{s'} \neq \mathbf{s}$) are redirected to be transitions $\mathbf{t} : \mathbf{s'} \rightarrow \mathbf{init_B}$. Self transitions $\mathbf{t}$ on $\mathbf{s}$ are replaced by transitions $\mathbf{t_x} : \mathbf{x} \rightarrow \mathbf{init_B}$ for each state $\mathbf{x}$ of $\mathbf{B}$. If $\mathbf{s} = \mathbf{init_A}$ then $\mathbf{init_B}$ becomes the new initial state of $\mathbf{A'}$.

If $\mathbf{s}$ has an entry action $\mathbf{Act}$ then this is added as the last action of any transition to the boundary of $\mathbf{s}$ (ie, to $\mathbf{init_B}$), and to any transition into a state of $\mathbf{B}$.

If $\mathbf{s}$ has an exit action $\mathbf{Act}$ then this is added as the first action of any transition out of $\mathbf{s}$ (ie, which does not have target any state of $\mathbf{s}$).

5.2 B: Eliminating Concurrent States

Similarly, an AND-composition $\mathbf{A} \mid \mathbf{B}$ of state machines can be expanded out to a state machine $\mathbf{C}$. The states of the expanded machine are pairs $(\mathbf{a}, \mathbf{b})$ of states $\mathbf{a}$ of $\mathbf{A}$ and $\mathbf{b}$ of $\mathbf{B}$. If $\mathbf{a}$ and $\mathbf{b}$ are basic states, then $(\mathbf{a}, \mathbf{b})$ is a basic state with entry action the composition $\mathbf{Entry_a} \mid\mid \mathbf{Entry_b}$ of the individual entry actions and exit action the composition $\mathbf{Exit_a} \mid\mid \mathbf{Exit_b}$ of the individual exit actions.

Transitions $\mathbf{t} : (\mathbf{a}, \mathbf{b}) \rightarrow (\mathbf{a'}, \mathbf{b'})$ arise either as:

1. A synchronised pair $\mathbf{t_1} : \mathbf{a} \rightarrow \mathbf{a'}$ and $\mathbf{t_2} : \mathbf{b} \rightarrow \mathbf{b'}$ of transitions of $\mathbf{A}$ and $\mathbf{B}$ respectively, where $\mathbf{event_A}(\mathbf{t_1}) = \mathbf{event_B}(\mathbf{t_2})$, and this event is then taken as $\mathbf{event_C}(\mathbf{t})$. The actions of $\mathbf{t}$ are then $\mathbf{Act_1} \mid\mid \mathbf{Act_2}$ where these are the separate actions of $\mathbf{t_1}$ and $\mathbf{t_2}$.

2. The lifting of a transition $\mathbf{t_1} : \mathbf{a} \rightarrow \mathbf{a'}$ of $\mathbf{A}$, and $\mathbf{b} = \mathbf{b'}$ and $\mathbf{event_A}(\mathbf{t_1}) \in \mathbf{Events_A} - \mathbf{Events_B}$. $\mathbf{event_C}(\mathbf{t_1})$ is defined to be $\mathbf{event_A}(\mathbf{t_1})$.

3. The lifting of a transition $\mathbf{t_2} : \mathbf{b} \rightarrow \mathbf{b'}$ of $\mathbf{B}$, and $\mathbf{a} = \mathbf{a'}$, and $\mathbf{event_B}(\mathbf{t_2}) \in \mathbf{Events_B} - \mathbf{Events_A}$. $\mathbf{event_C}(\mathbf{t_2})$ is defined to be $\mathbf{event_B}(\mathbf{t_2})$.

Conditions on transitions which refer to concurrent states can also be eliminated in the expansion of an AND composition of state machines to a single state machine. Such conditions are predicates $\mathbf{Condition_C}(\mathbf{t})$ for each transition $\mathbf{t}$ of a statechart $\mathbf{C}$, of the form **in S**, **not**(**in S**) and propositional logic combinations of these. If transition $\mathbf{t} : \mathbf{a} \rightarrow \mathbf{a'}$ in $\mathbf{A}$ has a condition $\mathbf{Condition_A}(\mathbf{t})$ **in s** on it, where $\mathbf{s} \in \mathbf{States_B}$, then the only transitions $\mathbf{t} : (\mathbf{a}, \mathbf{b}) \rightarrow (\mathbf{a'}, \mathbf{b'})$ in the expansion of $\mathbf{A} \mid \mathbf{B}$ are those where $\mathbf{s} = \mathbf{b}$. Similarly for the other forms of conditions.

In the resulting statechart $\mathbf{C}$ formed from $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{Condition_C}(\mathbf{t})$ is $\mathbf{Condition_A}(\mathbf{t})$ with each **in b** predicate replaced by **true**.

If transition $\mathbf{t}$ from $\mathbf{a}$ to $\mathbf{a}'$ includes an invocation of an event $\mathbf{e}$ in a concurrent state: $\mathbf{Act}_1 \frown \mathbf{e}(\mathbf{v}) \frown \mathbf{Act}_2$ then for each transition $\mathbf{t}'$ for $\mathbf{e}(\mathbf{x})$ from $\mathbf{b}$ to $\mathbf{b}'$ with labelling $\mathbf{Act}$ we obtain a modified transition for $\mathbf{t}$ with labelling $\mathbf{Act}_1 \frown \mathbf{Act}[\mathbf{v}/\mathbf{x}] \frown \mathbf{Act}_2$ from $(\mathbf{a}, \mathbf{b})$ to $(\mathbf{a}', \mathbf{b}')$.

The formula $\mathbf{t} \supset \mathbf{Act}_1;\ \mathbf{e}(\mathbf{v});\ \mathbf{Act}_2$ is added as a logical constraint to the resulting theory. Notice that $\mathbf{t}'$ may be fired by other occurrences of $\mathbf{e}$ in addition to those resulting from $\mathbf{t}$, so there may still be transitions of the form $\mathbf{t}' : (\mathbf{c}, \mathbf{b}) \to (\mathbf{c}, \mathbf{b}')$ in the expanded state machine.

6 Logical Representation of Sequence Diagrams

An object lifeline in a sequence chart can be expressed as a term in a process algebra language OHA (Object history algebra). Class and instance theories can be extended to include a *trace* which is an axiom expressing the allowed values that the object history can take within this algebra. The histories of different objects, ie, different lifelines within the same sequence diagram, can be composed by co-limit constructions of their theories, in which symbols are identified.

The OHA language for objects of a class $\mathbf{C}$ consists of the following atomic actions α:

1. $(\mathbf{a}, \mathbf{b})!\mathbf{m}(\mathbf{e})$ – send of message $\mathbf{m}(\mathbf{e})$ from object $\mathbf{a}$ to object $\mathbf{b}$;
2. $(\mathbf{a}, \mathbf{b})?\mathbf{m}(\mathbf{e})$ – reception of message $\mathbf{m}(\mathbf{e})$ by $\mathbf{b}$ from $\mathbf{a}$, where $\mathbf{m}$ is a method of $\mathbf{C}$.

These correspond to points in the history of an object on a sequence diagram, ie, points which are the source or target of message arrows or control transfers.

The terms of the OHA are then of form

$$\begin{aligned}\mathbf{P} \ ::= \ & \mathbf{STOP} \mid \\ & \alpha \to \mathbf{P} \mid \\ & \mathbf{P}||\mathbf{Q}\end{aligned}$$

where α is an atomic action of the OHA, $\mathbf{P}$ and $\mathbf{Q}$ are terms. This language is therefore a subset of a CSP algebra in which channels are identified by pairs $(\mathbf{a}, \mathbf{b}) : \mathbf{C} \times \mathbf{D}$ of objects. The same definitions of $\mathbf{traces}(\mathbf{P})$ are taken as in the traces semantics of CSP [4].

An additional feature is added to theories, called the *trace*, and containing an axiom defining the possible elements of the OHA which the trace of objects satisfying the theory may take.

As an example, consider a sequence chart with two object lifelines, for $\mathbf{ob}1 : \mathbf{C}$ and $\mathbf{ob}2 : \mathbf{D}$, where there is an association from $\mathbf{C}$ to $\mathbf{D}$ named $\mathbf{b}$ at the $\mathbf{D}$ end, ie, there is an attribute $\mathbf{b} : \mathbf{D}$ in the theory $\mathcal{I}_{\mathbf{C}}$ of instances of $\mathbf{C}$ (Figure 3).

The lifeline for $\mathbf{ob}1$ contains a send of message $\mathbf{m}$ to its linked $\mathbf{b}$ value, so the theory $\mathbf{L}_1$ of a general lifeline of this form, extending $\mathcal{I}_{\mathbf{C}}$, includes the trace component

$$(\mathbf{self}, \mathbf{b})!\mathbf{m}() \to \mathbf{STOP}$$

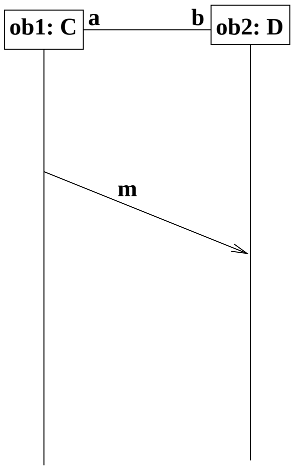

Fig. 3. Example Sequence Diagram

This abbreviates the axiom $\mathbf{trace} \in \mathbf{traces}(\mathbf{P})$ where $\mathbf{P}$ is the quoted process.

The lifeline for $\mathbf{ob}2$ contains a corresponding receive, so the general theory is $\mathbf{L}_2$ in this case with the trace component:

$$(\mathbf{a}, \mathbf{self})?\mathbf{m}() \rightarrow \mathbf{STOP}$$

We connect these lifeline theories and make them specific to $\mathbf{ob}1$ and $\mathbf{ob}2$ by defining theory morphisms:

1. $\mathbf{f} : \mathbf{L}_1 \rightarrow \mathbf{L}_3$ mapping $\mathbf{b} \mapsto \mathbf{ob}2$, $\mathbf{self} \mapsto \mathbf{ob}1$,
2. $\mathbf{g} : \mathbf{L}_2 \rightarrow \mathbf{L}_3$ mapping $\mathbf{a} \mapsto \mathbf{ob}1$, $\mathbf{self} \mapsto \mathbf{ob}2$.

$\mathbf{L}_3$ is the union of the two renamed theories $\mathbf{L}_1$ and $\mathbf{L}_2$. In $\mathbf{L}_3$ the trace is defined as the parallel composition of the renamed traces

$$(\mathbf{ob}1, \mathbf{ob}2)!\mathbf{m}() \rightarrow \mathbf{STOP}$$
$$(\mathbf{ob}1, \mathbf{ob}2)?\mathbf{m}() \rightarrow \mathbf{STOP}$$

of $\mathbf{L}_1$ and $\mathbf{L}_2$. As in CSP this can be reduced to a communication of $\mathbf{m}$ on the channel $(\mathbf{ob}1, \mathbf{ob}2)$:

$$(\mathbf{ob}1, \mathbf{ob}2).\mathbf{m}() \rightarrow \mathbf{STOP}$$

The interpretation of $\mathbf{traces}$ from $\mathbf{L}_1$ is $\mathbf{traces} \rhd \alpha\mathbf{P}$ in $\mathbf{L}_3$, where $\alpha\mathbf{P}$ is the language of the trace process of $\mathbf{L}_1$, and similarly for $\mathbf{L}_2$.

7 Proving Pattern Introductions as Refinements

One application of the semantics is to prove that the introduction of particular *design patterns* [3] are refinements. Here we will consider the Strategy pattern.

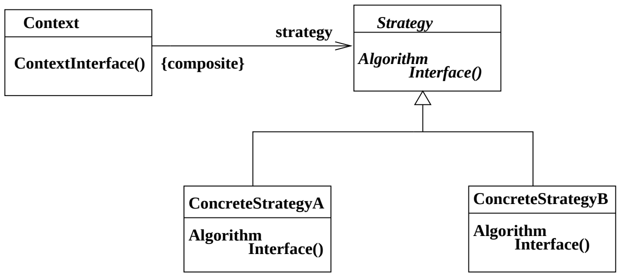

Fig. 4. Strategy Pattern Structure

Figure 4 describes the general form of this pattern. The pattern allows multiple algorithms to be used for a particular abstract operation, the choice between these algorithms (embedded in particular **Strategy** subclasses) being made by polymorphism.

Strategy can be used to transform a system of the form of the left hand side of Figure 5 to the form given on the right hand side.

The theory interpretation in the case of the Strategy pattern is given in Table 1.

Symbol of $\Gamma_{\mathbf{Client}}$	Term of $\Gamma_{\mathbf{Client}1}$
$\mathbf{Client}$	$\mathbf{Client}1$
$\mathbf{Context}$	$\mathbf{Context}1$
$\mathbf{create}_{\mathbf{Client}}$	$\mathbf{create}_{\mathbf{Client}1}$
$\mathbf{create}_{\mathbf{Context}}$	$\mathbf{create}_{\mathbf{Context}1}$
$\mathbf{obj}.\mathbf{aContext}$	$\mathbf{obj}.\mathbf{aContext}$
$\overline{\mathbf{Client}}$	$\overline{\mathbf{Client}1}$
$\overline{\mathbf{Context}}$	$\overline{\mathbf{Context}1}$
$\mathbf{obj}.\mathbf{m}$	$\mathbf{obj}.\mathbf{m}$

Table 1. Interpretation of **Client** into **Client**1

Additionally, $\mathbf{obj}.\mathbf{aContext}.\mathbf{strategy\_type}$ will be interpreted by the conditional expression

$$
\begin{array}{l}
\mathbf{if}\ \mathbf{obj}.\mathbf{aContext}.\mathbf{strategy}\ \in\ \overline{\mathbf{ConcreteStrategyA}} \\
\mathbf{then}\ \mathbf{type}1 \\
\mathbf{else} \\
\quad \mathbf{if}\ \mathbf{obj}.\mathbf{aContext}.\mathbf{strategy}\ \in\ \overline{\mathbf{ConcreteStrategyB}} \\
\quad \mathbf{then}\ \mathbf{type}2 \\
\quad \mathbf{else}\ \mathbf{nil}
\end{array}
$$

All actions of the abstract system are interpreted by actions of the same name in the new system. The only significant change in representation is the in-

```
class Client
instance variables
  aContext : Context;
init objectstate ==
  create_Context(aContext)
methods
  m() ==
    (aContext.set_type1(); ...
     aContext.ContextInterface())
end Client

class Context
instance variables
  strategy_type : { type1, type2 }
methods
  set_type1() ==
    strategy_type := type1;

  set_type2() == ...;

  ContextInterface() ==
    if strategy_type = type1
    then Code1
    else Code2

end Context
```

```
class Client1
instance variables
  aContext : Context1;
init objectstate ==
  create_Context1(aContext)
methods
  m() ==
    (aContext.set_type1(); ...
     aContext.ContextInterface())
end Client1

class Context1
instance variables
  strategy : Strategy
methods
  ContextInterface() ==
     strategy.AlgorithmInterface();

  set_type1() ==
     create_ConcreteStrategyA(strategy);

  set_type2() ==
     create_ConcreteStrategyB(strategy)
end Context1

class Strategy
methods
  AlgorithmInterface()
     is subclass responsibility
end Strategy

class ConcreteStrategyA
  is subclass of Strategy
methods
  AlgorithmInterface() == Code1
end ConcreteStrategyA
```

Fig. 5. Abstract and Concrete Strategy Structures

terpretation of **obj.aContext.strategy_type** by a conditional expression in **obj.aContext.strategy**.

The typing axioms of the abstract system are therefore directly provable, in their interpreted versions, in the concrete system. For example, the constraint that **obj.aContext** $\in$ **Context** for **obj** $\in \overline{\textbf{Client}}$ in the theory $\Gamma_{\textbf{Client}}$ is interpreted by the predicate

$$\textbf{obj} \in \overline{\textbf{Client1}} \Rightarrow \textbf{obj.aContext} \in \textbf{Context1}$$

in $\Gamma_{\textbf{Client1}}$. But this is a theorem of $\Gamma_{\textbf{Client1}}$ from its own typing axiom for

aContext, as required.

The effect of **obj**.**aContext**.**set_type**1 in **Client** is given by the axiom:

$$
\begin{array}{l}
\mathbf{obj} \in \overline{\mathbf{Client}} \;\wedge \\
\mathbf{obj}.\mathbf{aContext} \in \overline{\mathbf{Context}} \;\Rightarrow \\
\quad (\mathbf{obj}.\mathbf{aContext}).\mathbf{set\_type}1 \;\supset \\
\qquad \mathbf{obj}.\mathbf{aContext}.\mathbf{strategy\_type} := \;\mathbf{type}1
\end{array}
$$

That is, calls of (**obj**.**aContext**).**set_type**1 result in **obj**.**aContext**.**strategy_type** having the value **type**1 at their conclusion.

Under the above interpretation this becomes:

$$
\begin{array}{l}
\mathbf{obj} \in \overline{\mathbf{Client}1} \;\wedge \\
\mathbf{obj}.\mathbf{aContext} \in \overline{\mathbf{Context}1} \;\Rightarrow \\
\quad (\mathbf{obj}.\mathbf{aContext}).\mathbf{set\_type}1 \;\supset \\
\qquad\qquad \mathbf{pre}\,\mathbf{true}\;\; \mathbf{post}\,\mathbf{e} = \;\mathbf{type}1
\end{array}
$$

where **e** is the conditional expression (1):

$$
\begin{array}{l}
\mathbf{if}\ \mathbf{obj}.\mathbf{aContext}.\mathbf{strategy} \;\in\; \overline{\mathbf{ConcreteStrategyA}} \\
\mathbf{then}\ \mathbf{type}1 \\
\mathbf{else} \\
\quad \mathbf{if}\ \mathbf{obj}.\mathbf{aContext}.\mathbf{strategy} \;\in\; \overline{\mathbf{ConcreteStrategyB}} \\
\quad \mathbf{then}\ \mathbf{type}2 \\
\quad \mathbf{else}\ \mathbf{nil}
\end{array}
$$

In other words:

$$
\begin{array}{l}
\mathbf{obj} \in \overline{\mathbf{Client}1} \;\wedge \\
\mathbf{obj}.\mathbf{aContext} \in \overline{\mathbf{Context}1} \;\Rightarrow \\
\quad (\mathbf{obj}.\mathbf{aContext}).\mathbf{set\_type}1 \;\supset \\
\qquad \mathbf{pre}\ \mathbf{true} \\
\qquad \mathbf{post}\ \mathbf{obj}.\mathbf{aContext}.\mathbf{strategy} \in \;\overline{\mathbf{ConcreteStrategyA}}
\end{array}
$$

But this is provable from the axiom for **obj**.**aContext**.**set_type** in **Client**1, which is:

$$
\begin{array}{l}
\mathbf{obj} \in \overline{\mathbf{Client}1} \;\wedge \\
\mathbf{obj}.\mathbf{aContext} \in \overline{\mathbf{Context}1} \;\Rightarrow \\
\quad (\mathbf{obj}.\mathbf{aContext}).\mathbf{set\_type}1 \;\supset \\
\qquad \mathbf{create}_{\mathbf{ConcreteStrategyA}}(\mathbf{obj}.\mathbf{aContext}.\mathbf{strategy})
\end{array}
$$

Similar reasoning shows that the interpretation of the axiom for the effect of **m** in $\Gamma_{\mathbf{Client}}$ is provable from the corresponding axiom for **m** in $\Gamma_{\mathbf{Client}1}$.

We can now make precise the assumptions required for the transformation to be correct:

– $\mathbf{obj}.\mathbf{aContext}.\mathbf{strategy} \in \overline{\mathbf{Strategy}}$ at the point where **ContextInterface** is called;

– **obj.aContext.strategy** must remain in the same subclass of **Strategy** during the execution of **ContextInterface**.

Other applications of the semantics include proving that transformations such as source and target splitting on statecharts [1] are also refinements.

Conclusions

The definition of a precise semantics for the UML is an important aim if UML models are to be used for development of critical systems. The semantics we have given supports verification of refinement and a transformational approach to development which reduces the proof burden associated with traditional formal techniques [6].

The use of an axiomatic framework enables a direct relationship to be established between proof tools for the UML and the semantics. It is also more appropriate than a denotational approach in the case of a general specification notation such as the UML which has no specific executable interpretation.

Work is continuing to extend the semantics to other UML notations, and to develop tools to support verified transformations on UML models.

References

1. S. Cook, J. Daniels, *Designing Object Systems: Object-oriented Modelling with Syntropy*, Prentice Hall, 1994.
2. J C Bicarregui, K C Lano, T S E Maibaum, *Objects, Associations and Subsystems: a hierarchical approach to encapsulation*, ECOOP 97, LNCS, 1997.
3. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley.
4. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
5. K Lano. *The B Language and Method: A Guide to Practical Formal Development*, Springer-Verlag, 1996.
6. K. Lano, J. Bicarregui, *Formalising the UML in Structured Temporal Theories*, ECOOP 98 Workshop on Behavioural Semantics, Technical Report TUM-I9813, Technische Universitat Muchen, 1998.
7. K. Lano, J. Bicarregui, *UML Refinement and Abstraction Transformations*, ROOM 2 Workshop, University of Bradford, 1998.
8. K Lano, *Logical Specification of Reactive and Real-Time Systems*, *Journal of Logic and Computation*, Vol. 8, No. 5, pp 679–711, 1998.
9. K. Lano, D. Clark, *Demonstrating Preservation of Safety Properties in Reactive Control System Development*, 4th Australian Workshop on Industrial Experience with Safety Critical Systems and Software, Canberra, ACT, November 1999.
10. Rational Software et al, *OMG Unified Modeling Language Specification Version 1.3*, June 1999.
11. Rational Software et al, *OMG Unified Modeling Language Specification Version 1.3: UML Semantics*, June 1999.

This article was processed using the LaTeX macro package with LLNCS style