RSDS: A Subset of UML with Precise Semantics

K. Lano, D. Clark, K. Androutsopoulos*

December 13, 2001

Abstract

In this paper we describe the semantic foundation of a subset, RSDS, of the UML notation. This subset is specifically intended to support reactive system specification, and has a formal semantics expressed in temporal logic. We define this semantics, and use the semantics to justify RSDS techniques for modular decomposition and verification of reactive systems, involving a structure-preserving translation to SMV.

1 Introduction: Specification in RSDS

RSDS (Reactive System Design Support) supports the specification of reactive systems using statemachines, class diagrams and invariants [12, 13]. Its philosophy is to express such specifications in the simplest terms possible: no formal notation apart from these three notations are used, and as much automated support for analysis, design and code generation as possible is built into the RSDS tool.

An RSDS system is a tuple $(\mathbf{Sens}, \mathbf{Invs}, \mathbf{Conts}, \mathbf{Acts})$ where **Sens** is a set of named statemachines representing sensor components, **Invs** is a set of invariants in the syntax described in the following section, **Conts** is a set of named statemachines representing controllers, and **Acts** is a set of named statemachines representing actuator components.

1.1 Invariants

The use of invariants as the primary means to specify discrete control systems has been inspired by the corresponding use of "transfer functions" in continuous control theory to express the required behaviour of a continuous controller [9]. Invariants have also been central in formal synthesis of programs [8], formal design decomposition strategies [10], and in critical applications of formal methods [16].

In a discrete system such invariants are principally of three kinds:

1. Static (or single state) invariants: $\mathbf{P} \Rightarrow \mathbf{Q}$ where both **P** and **Q** consist only of constraints on the current states of sensors and actuators in the system. No temporal operators or event names are used. For example:

   ```
   sw = On & s2 = Off  =>  bm = On
   ```

   "if the switch is on and the belt end sensor is off, the belt motor is on", for the feed belt of the simple production cell [13], where **s2** is the blank sensor at the end of the belt and **bm** the belt motor (Figure 1).

2. Operational invariants α & $\mathbf{P} \Rightarrow \mathbf{AX}(\mathbf{Q})$ where **P** and **Q** are state constraints as in 1, and α an event of some sensor. **AX** is SMV notation for "in all next states". For example

   ```
   swon & sw = Off & s2 = Off  =>  AX(bm = On)
   ```

*Department of Computer Science, King's College London, Strand, London WC2R 2LS

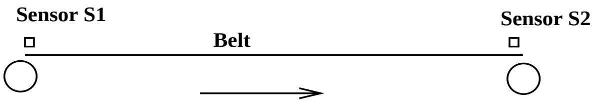

Figure 1: Belt of Simple Production Cell

3. Temporal invariants $\mathbf{P} \Rightarrow \mathbf{AF}(\mathbf{Q})$ or $\mathbf{P} \Rightarrow \mathbf{AG}(\mathbf{Q})$. $\mathbf{AF}$ is "in some future state" and $\mathbf{AG}$ is "in all future states". For example

```
s2 = Off & bm = On  =>  AF(s2 = On)
```

Invariants are divided into system constraints (which the developer needs to ensure by implementing appropriate control code, or synthesising this code from the invariants using the RSDS tool) and environmental constraints (assumed to be true, and which the developer can use in verification, but which may be checked in a fault-tolerant architecture in order to put the system in a safe state if these invariants fail).

Invariants are also divided into critical (for example invariants which guarantee safe behaviour of the EUC) and non-critical invariants. Critical invariants are treated differently by the tool (in particular there is tracing of these through to the components which implement them) but are treated in the semantics the same way as non-critical invariants.

Formally, an invariant of an RSDS system $\mathbf{Sys} = (\mathbf{Sens}, \mathbf{Invs}, \mathbf{Conts}, \mathbf{Acts})$ is any formula in the following CTL [2] language:

1. Atomic formulae: propositional constants **true** and **false**; action symbols α for each event of an element $\mathbf{Sm} \in \mathbf{Sens} \cup \mathbf{Acts}$; equality formulae $\mathbf{sm} = \mathbf{v}$ where $\mathbf{sm}$ is an attribute symbol representing the state variable of an element $\mathbf{Sm} \in \mathbf{Sens} \cup \mathbf{Conts} \cup \mathbf{Acts}$, and $\mathbf{v}$ is a constant representing an element of the state set of $\mathbf{Sm}$.

2. If φ and ϕ are formulae, so are $\varphi \Rightarrow \phi$, $\varphi \vee \phi$, $\varphi \ \& \ \phi$

3. If φ is a formula, so are $\mathbf{AX}\varphi$, $\mathbf{EX}\varphi$, $\mathbf{AG}\varphi$, $\mathbf{EG}\varphi$, $\mathbf{AF}\varphi$, and $\mathbf{EF}\varphi$.

The RSDS tool currently processes formulae of the form $\mathbf{A} \Rightarrow \mathbf{S}$ where $\mathbf{A}$ is a disjunction of conjunctions of atomic formulae, and $\mathbf{S}$ is a conjunction of atomic formulae or $\mathbf{S}$ is of the form $\mathbf{Ops}\ \mathbf{P}$ where $\mathbf{Ops}$ is a non-empty sequence of temporal operators and $\mathbf{P}$ is a non-temporal formula not involving $\Rightarrow$.

1.2 Statemachines

In RSDS we use a subset of statechart notation called *Structured Reactive System* (*SRS*) statecharts, based on a hierarchical structure of statemachines [13].

A sensor or actuator statemachine is a tuple $\mathbf{A} = (\mathbf{init_A}, \mathbf{States_A}, \mathbf{Trans_A}, \mathbf{Events_A})$ where $\mathbf{init_A} \in \mathbf{States_A}$ is the initial state, $\mathbf{States_A}$ is the non-empty set of states of $\mathbf{A}$, $\mathbf{Events_A}$ the set of events of $\mathbf{A}$, and $\mathbf{Trans_A}$ the set of transitions of $\mathbf{A}$. There are functions

$$\mathbf{source_A} : \mathbf{Trans_A} \rightarrow \mathbf{States_A}$$
$$\mathbf{target_A} : \mathbf{Trans_A} \rightarrow \mathbf{States_A}$$
$$\mathbf{event_A} : \mathbf{Trans_A} \rightarrow \mathbf{Events_A}$$

which give the source, target and triggering event of each transition. We write $\mathbf{tr} : \mathbf{s} \rightarrow_\alpha \mathbf{t}$ to indicate that $\mathbf{tr}$ is a transition with source $\mathbf{s}$, target $\mathbf{t}$ and event α.

A controller statemachine $\mathbf{C}$ has the same structure but additionally has a set of receivers:

$$\mathbf{receivers_{Sys}}(\mathbf{C}) \subseteq \mathbf{Conts} \cup \mathbf{Acts}$$

and generations on its transitions:

$$\mathbf{generations_C} : \mathbf{Trans_C} \rightarrow \mathrm{seq}(\mathbf{Events_{R_1}} \cup \ldots \cup \mathbf{Events_{R_p}})$$

where $\mathbf{receivers_{Sys}}(\mathbf{C}) = \{\mathbf{R}_1, \ldots, \mathbf{R_p}\}$. No member of $\mathbf{Ext_{Sys}}$ can occur in any generation.

An abstraction morphism $\sigma : \mathbf{C} \to \mathbf{A}$, where $\mathbf{Events_C} \subseteq \mathbf{Events_A}$, consists of functions

$$\sigma : \mathbf{States_C} \to \mathbf{States_A}$$
$$\sigma : \mathbf{Trans_C} \to \mathbf{Trans_A}$$

with $\sigma(\mathbf{init_C}) = \mathbf{init_A}$ and such that $\mathbf{tr} : \mathbf{s} \to_\alpha \mathbf{t}$ in $\mathbf{C}$ implies that $\sigma(\mathbf{tr}) : \sigma(\mathbf{s}) \to_\alpha \sigma(\mathbf{t})$ in $\mathbf{A}$.

σ is *adequate* if its two submaps are both surjective: every state $\mathbf{s}$ of $\mathbf{A}$ has some corresponding state of $\mathbf{C}$:

$$\forall \mathbf{s} : \mathbf{States_A} \cdot \exists \mathbf{s}' : \mathbf{States_C} \cdot \sigma(\mathbf{s}') = \mathbf{s}$$

and if every event α which has an abstract transition from some $\mathbf{s}$ also has a corresponding concrete transition from each corresponding state:

$$\begin{array}{l} \forall \mathbf{s} : \mathbf{States_A};\ \mathbf{t} : \mathbf{Trans_A} \cdot \\ \quad \mathbf{event_A}(\mathbf{t}) = \alpha \wedge \mathbf{source_A}(\mathbf{t}) = \mathbf{s} \ \Rightarrow \\ \quad\quad (\forall \mathbf{s}' : \mathbf{States_C} \cdot \sigma(\mathbf{s}') = \mathbf{s} \ \Rightarrow \\ \quad\quad\quad \exists \mathbf{t}' : \mathbf{Trans_C} \cdot \mathbf{event_C}(\mathbf{t}') = \alpha \ \wedge \ \sigma(\mathbf{t}') = \mathbf{t} \ \wedge \ \mathbf{source_C}(\mathbf{t}') = \mathbf{s}') \end{array}$$

This ensures that $\mathbf{C}$ can react to α in every situation that $\mathbf{A}$ can.

A statemachine $\mathbf{Sm}$ is *consistent* if there is at most one transition for each event from each state of the component:

$$\begin{array}{l} \forall \mathbf{tr}, \mathbf{tr}' : \mathbf{Trans_{Sm}} \cdot \\ \quad\quad \mathbf{event_{Sm}}(\mathbf{tr}) = \mathbf{event_{Sm}}(\mathbf{tr}') \wedge \mathbf{source_{Sm}}(\mathbf{tr}) = \mathbf{source_{Sm}}(\mathbf{tr}') \ \Rightarrow \ \mathbf{tr} = \mathbf{tr}' \end{array}$$

Components in a UML or RSDS system are required to be consistent. A component is *complete* if there is a transition for each event from each state of the component:

$$\forall \alpha \in \mathbf{Events_{Sm}};\ \mathbf{s} : \mathbf{States_{Sm}} \cdot \exists \mathbf{tr} : \mathbf{Trans_{Sm}} \cdot \mathbf{source_{Sm}}(\mathbf{tr}) = \mathbf{s} \wedge \mathbf{event_{Sm}}(\mathbf{tr}) = \alpha$$

1.3 Class Diagrams

Class diagrams are used to display the components involved in the system as entities, and their relationships with each other. This is of particular relevance when there may be multiple instances of a component in a system (eg, multiple valves in a chemical plant). A class is represented as a named rectangle, the name being the same as the statemachine for the component. An association is represented as a named line between two distinct classes.

Invariants and controller statemachines can be extended to express properties and behaviour in class diagrams as follows:

- atomic formulae are:
  1. equalities $\mathbf{obj.att} = \mathbf{val}$ (denoting $\mathbf{att}(\mathbf{obj}) = \mathbf{val}$) where $\mathbf{att}$ is the state variable or other attribute of a class $\mathbf{C}$, $\mathbf{obj}$ is an instance of $\mathbf{C}$, and $\mathbf{val}$ a member of the type of $\mathbf{att}$
  2. membership predicates $\mathbf{obj} : \mathbf{C}$ and $(\mathbf{x}, \mathbf{y}) : \mathbf{r}$ where $\mathbf{r}$ is a relationship
- controller transitions $\mathbf{t}(\mathbf{a})$ are indexed by elements $\mathbf{a}$ of the sensors $\mathbf{Sm}$ whose events they respond to. The generations of these transitions can include events $\mathbf{a}.\mathbf{r}.\beta$ (denoting the action symbol $\beta(\mathbf{r}(\mathbf{a}))$) where $\mathbf{r}$ is a relation (or composition of relations) directed from $\mathbf{Sm}$. They can also include events $\mathbf{new_D}(\mathbf{x})$, $\mathbf{kill_D}(\mathbf{x})$, and $\mathbf{link_r}(\mathbf{x}, \mathbf{y})$ and $\mathbf{unlink_r}(\mathbf{x}, \mathbf{y})$ for any relation $\mathbf{r}$ incident with $\mathbf{Sm}$.

The most abstract way of expressing properties about class diagrams is to use a series of membership predicates to identify the type of objects and which relationships they belong to, then use equality predicates to constrain their states:

$$(\mathbf{a}, \mathbf{b}) : \mathbf{r} \ \& \ \mathbf{a.x} = \mathbf{On} \ \Rightarrow \ \mathbf{b.y} = \mathbf{On}$$

in the model of Figure 2(a), for example.

This form is the basis of the decompositions described in Section 5. In more design-oriented models, we can express properties more concisely by identifying a direction to relationships (Figure 2(b)). In this case we can write:

$$\mathbf{a} : \mathbf{X} \ \& \ \mathbf{a.x} = \mathbf{On} \ \Rightarrow \ \mathbf{a.r.y} = \mathbf{On}$$

This form can be regarded as an abbreviation for the expanded form which introduces explicit objects **b** for the elements of **Y** in the relation **r** with **a**. Finally, if there is an aggregation (Figure 2(c)) then the

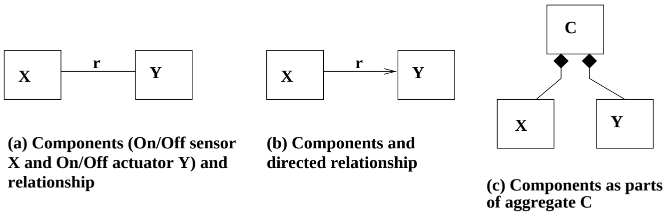

Figure 2: Class Diagram Notations

'parts' of the aggregation can be considered as directly being attributes of the 'whole' entity, so we can write, eg:

$$\mathbf{c} : \mathbf{C} \ \& \ \mathbf{c.x} = \mathbf{On} \ \Rightarrow \ \mathbf{c.y} = \mathbf{On}$$

This form can be re-expressed in the more basic forms by introducing intermediate relationships **cx** from **C** to **X** and **cy** from **C** to **Y**.

Figure 3 shows two example class diagrams, from a railway signalling system [1] and the basic production cell [15]. An example invariant from the signalling system is that if a train is present on the bridge,

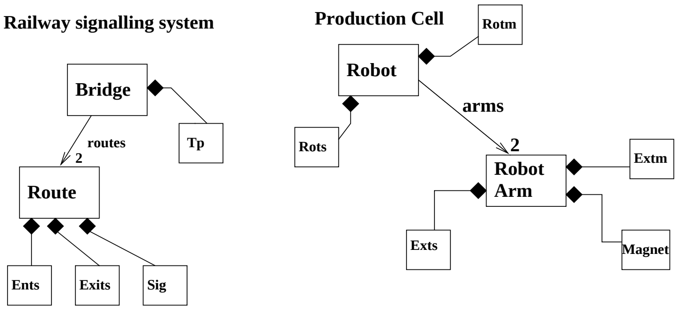

Figure 3: Example Class Diagrams

then the signals for both routes (directions) across it are red:

$$\mathbf{b} : \mathbf{Bridge} \ \& \ \mathbf{b.tp} = \mathbf{On} \ \Rightarrow \ \mathbf{b.routes.sig} = \mathbf{red}$$

Likewise, for the robot system, if the robot is rotating, both its arms must be retracted:

$$\mathbf{r} : \mathbf{Robot} \ \& \ \mathbf{r.rotm} \neq \mathbf{Off} \ \Rightarrow \ \mathbf{r.arms.exts} = \mathbf{ret}$$

2 Semantics of RSDS Specifications

There are two main semantic views of an RSDS system which are of interest for verification and validation:

1. A “coarse grain” semantics which describes “computations” of the model as sequences of steps, each step consists of a single sensor event and a finite number of actuator events, comprising the reaction to this sensor event, or it consists of no events. This corresponds to the specification level machine descriptions in the B method.

   At this level we can reason about global system behaviour such as the liveness property expressed in item 3 of Section 1.1 above.

2. A “fine grain” semantics in which steps consist of a sensor event followed by a sequence of actuator events which are the response to the sensor event – ie, an additional level of granularity is introduced to represent ordering of actuator commands within a reaction cycle. This corresponds to the implementation level descriptions in the B method. Again a step can consist of no events.

   At this level we can reason about constraints local to the reaction to a single event, such as that a lift must be stopped before its doors can be opened.

We first describe these semantics at the single object level and then show how they can be generalised to classes and multiple instances.

2.1 Coarse-grain Semantics (Object Level)

We define the coarse-grain semantics of an RSDS system $\mathbf{Sys} = (\mathbf{Sens}, \mathbf{Invs}, \mathbf{Conts}, \mathbf{Acts})$ in terms of temporal theories in CTL.

The *external events* of **Sys** are the sensor events:

$$\mathbf{Ext_{Sys}} \;=\; \bigcup_{\mathbf{i:1..n}} \mathbf{Events_{S_i}}$$

all other events of **Sys** are *internal*.

For each component statemachine **Sm** a theory $\Gamma_{\mathbf{Sm}}$ is defined. This has a single attribute **sm** ranging over $\mathbf{States_{Sm}}$ and it has action symbols for each element of $\mathbf{Trans_{Sm}} \cup \mathbf{Events_{Sm}}$.

The axioms of $\Gamma_{\mathbf{Sm}}$ are:

1. The state-transition behaviour of **Sm**:

   $$\begin{array}{l} \mathbf{AG}(\mathbf{sm} = \mathbf{s} \;\wedge\; \alpha \;\Rightarrow\; \mathbf{tr}) \\ \mathbf{AG}(\mathbf{tr} \;\Rightarrow\; \alpha) \\ \mathbf{AG}(\mathbf{sm} = \mathbf{s} \;\wedge\; \mathbf{tr} \;\Rightarrow\; \mathbf{AX}(\mathbf{sm} = \mathbf{t})) \end{array}$$

   for each transition **tr** of **Sm** with source **s**, target **t** and trigger event α. (We use the notation $\mathbf{tr} : \mathbf{s} \rightarrow_{\alpha} \mathbf{t}$ in this case).

   In the initial state:

   $$\mathbf{sm} = \mathbf{init_{Sm}}$$

   where $\mathbf{init_{Sm}}$ is the initial state of **Sm**.

2. That at most one transition of **Sm** can occur in a step:

   $$\mathbf{AG}\neg\,(\mathbf{tr} \wedge \mathbf{tr}')$$

   for each pair of distinct transitions of **Sm**. Similarly for events.

3. That a transition can only occur if **Sm** is in its source state:

   $$\mathbf{AG}(\mathbf{tr} \;\Rightarrow\; \mathbf{sm} = \mathbf{s})$$

   where $\mathbf{s} = \mathbf{source_{Sm}}(\mathbf{tr})$.

4. Locality [7]:

$$\mathbf{AG}(\neg\, \mathbf{tr}_1 \wedge \ldots \wedge \neg\, \mathbf{tr_n} \;\wedge\; \mathbf{sm} = \mathbf{s} \;\Rightarrow\; \mathbf{AX}(\mathbf{sm} = \mathbf{s}))$$

for each $\mathbf{s} \in \mathbf{States_{Sm}}$, where the $\mathbf{tr_i}$ are all transition action symbols of $\mathbf{Sm}$.

5. For a controller transition $\mathbf{tr}$ with generations ρ:

$$\mathbf{AG}(\mathbf{tr} \;\Rightarrow\; \rho(1) \wedge \ldots \wedge \rho(\mathbf{p}))$$

where $\mathbf{p}$ is the length $\mathbf{size}(\rho)$ of ρ.

The semantic interpretation of an invariant $\mathbf{I} \in \mathbf{Invs}$ is $\mathbf{AG}(\mathbf{I})$.

The coarse-grain semantics $\Gamma_{\mathbf{Sys}}$ of an RSDS system $\mathbf{Sys}$ is the colimit of all the theories $\Gamma_{\mathbf{Sm}}$ for $\mathbf{Sm} \in \mathbf{Sens} \cup \mathbf{Conts} \cup \mathbf{Acts}$, extended with the following global axioms, and $\mathbf{AG(I)}$ for each invariant $\mathbf{I}$ in $\mathbf{Invs}$.

Global axioms of $\Gamma_{\mathbf{Sys}}$ are:

1. That only one sensor event occurs in each step:

$$\mathbf{AG}\neg\,(\mathbf{tr} \wedge \mathbf{tr}')$$

for each pair of transitions $\mathbf{tr}$ of $\mathbf{Sm}$, $\mathbf{tr}'$ of $\mathbf{Sm}'$ where $\mathbf{Sm}$ and $\mathbf{Sm}'$ are distinct sensor components. Similarly for events of distinct sensor components. We assume that $\mathbf{Events_{Sm}} \cap \mathbf{Events_{Sm'}} = \varnothing$ for distinct components in the model, except that sensors may have events in common with controller components (but not each other) and similarly the sets of action symbols representing transitions of different statemachines are disjoint.

2. A "system locality" principle also states that each actuator event can only occur as a result of a reaction to some sensor event:

$$\mathbf{AG}(\beta \;\Rightarrow\; \alpha_1 \vee \ldots \vee \alpha_{\mathbf{n}})$$

for each β which is an event of some actuator component, and where the $\alpha_{\mathbf{i}}$ are all events of the set of sensor components.

3. A "coverage" axiom states that if a transition is triggered by an internal event (ie, not an event of any sensor), then a transition containing that event in its generations must also have occurred:

$$\mathbf{AG}(\alpha \;\Rightarrow\; \mathbf{tr}_1 \vee \ldots \vee \mathbf{tr_P})$$

for each event $\alpha \in \bigcup_{\mathbf{Sm} \in \mathbf{Conts} \cup \mathbf{Acts}} \mathbf{Events_{Sm}} - \mathbf{Ext_{Sys}}$, where the $\mathbf{tr_i}$ are all the transitions $\mathbf{t}$ in $\bigcup_{\mathbf{Sm} \in \mathbf{Conts}} \mathbf{Trans_{Sm}}$ which have $\alpha \in \mathbf{ran}(\mathbf{generations_{Sm}}(\mathbf{t}))$.

Note that we do not require that events only occur if they are enabled, ie, if their component is currently in a source state of a transition for the event. For event $\alpha \in \mathbf{Events_{Sm}}$, define

$$\mathbf{en}(\alpha) \;\equiv\; \mathbf{sm} = \mathbf{s}_1 \vee \ldots \vee \mathbf{sm} = \mathbf{s_n}$$

where the $\mathbf{s_i}$ are all source states of transitions triggered by α in $\mathbf{Sm}$. Then if $\alpha \wedge \neg\, \mathbf{en}(\alpha)$ holds, no transition can fire and hence the state of the component does not change. Alternative semantics in which the system moves into a non-normal mode if such events occur could also be considered, to represent fault-detection.

2.2 Fine-grain Semantics

In this view we explicitly model the cycles of event detection and reaction, and model the order in which actuator commands are issued. This differs from the coarse-grain semantics in two key aspects:

1. More than one sensor event may occur as input to the system in a single cycle – determining which event to react to first is carried out by a dedicated 'outer level' or system interface module within its implementation, guided by event criticality/hazard analysis [12].

2. Actuator events may occur without sensor events occurring, however they can occur only within a cycle if preceeded by a sensor event within this cycle. Similarly, internal events must be preceeded within a reaction cycle by an external event.

   System locality is dropped, and replaced by axioms concerning the queue of events in the system: ie, that actuator events may only occur if they are at the head of the queue of events waiting to be processed.

Definition A *system state* of an RSDS specification $\mathbf{S}$ is a tuple $\mathbf{x} = (\mathbf{s}_1, \ldots, \mathbf{s}_\mathbf{n})$ of elements $\mathbf{s_i} \in \mathbf{States}_{\mathbf{Sm_i}}$ where $\mathbf{Sm}_1, \ldots, \mathbf{Sm_n}$ are all the sensor and actuator statemachines of the RSDS specification, where $\mathbf{x}$ satisfies all the invariants of $\mathbf{S}$.

System states are the only states in models of the coarse-grain semantics, and these correspond to the *stable states* which are endpoints of reaction cycles in the fine-grain semantics (Figure 4).

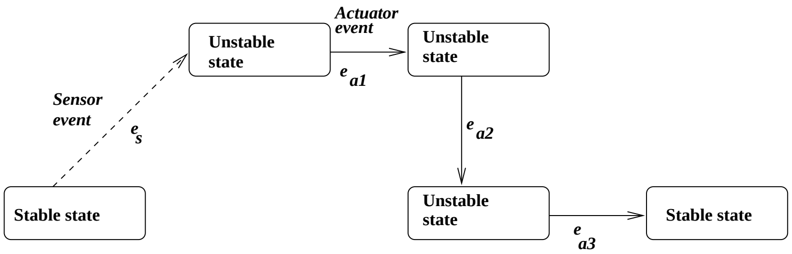

Figure 4: Fine-grain Semantics

A map σ is an abstraction from the fine to the coarse-grain semantics if:

- Each stable state $\mathbf{s}$ in the concrete model $\mathbf{C}$ abstracts to a state $\sigma(\mathbf{s})$ of the abstract model $\mathbf{A}$;

- Each transition $\mathbf{t}'$ for a sensor event in the concrete model abstracts to a corresponding transition $\mathbf{t}$ for this event in the abstract model, such that $\mathbf{source_A}(\mathbf{t}) = \sigma(\mathbf{source_C}(\mathbf{t}'))$ and $\mathbf{target_A}(\mathbf{t})$ is $\sigma(\mathbf{s}')$ where $\mathbf{s}'$ is the stable state of $\mathbf{C}$ which results from the completed reaction to $\mathbf{t}'$ in $\mathbf{C}$.

Such a map is *adequate* if σ is onto for states and transitions, as usual.

2.2.1 Axiomatic Fine-Grain Semantics

The fine-grain sematics of a subsystem **Sys** consisting of sensors $\mathbf{S}_1, \ldots, \mathbf{S_n}$, controllers $\mathbf{C}_1, \ldots, \mathbf{C_m}$ and actuators $\mathbf{A}_1, \ldots, \mathbf{A_p}$ can be expressed in terms of the statemachines of these components together with axioms on a system attribute

$$\mathbf{event\_queue} : \mathrm{seq}(\mathbf{EventToken})$$

which holds a sequence of tokens representing pending events. The axioms of components are the same as in the coarse grain case, as are system axioms, except that coverage of actuator events by sensor events is dropped and replaced by the following axioms on **event_queue**.

EventToken is some set isomorphic to the set $\mathbf{Events_{Sys}}$ of all events of the system:

$$\mathbf{Events_{Sys}} = \bigcup_{\mathbf{i:1..n}} \mathbf{Events_{S_i}} \cup \bigcup_{\mathbf{i:1..m}} \mathbf{Events_{C_i}} \cup \bigcup_{\mathbf{i:1..p}} \mathbf{Events_{A_i}}$$

Let

$$\mathbf{name} : \mathbf{Events_{Sys}} \to \mathbf{EventToken}$$

be this semantic isomorphism.

The event queue is initially empty:

$$\mathbf{event\_queue} = []$$

An external event α can only be responded to if the queue is empty:

$$\mathbf{AG}(\mathbf{t} \;\Rightarrow\; \mathbf{event\_queue} = [])$$

for each transition $\mathbf{t}$ with trigger event some $\alpha \in \mathbf{Ext_{Sys}}$.

When an external event is accepted by the system, the generations of the triggered transition become the new event queue – all these events should be internal:

$$\mathbf{AG}(\mathbf{t} \;\Rightarrow\; \mathbf{AX}(\mathbf{event\_queue} = \mathbf{generations}(\mathbf{t})))$$

where $\mathbf{t}$ is (a sensor or controller) transition triggered by α. If several transitions in different components are triggered by α then some interleaving of their generations becomes the new event queue. Normally this will only be the case if one transition is in some $\mathbf{S_i}$ (and so has no generations) and the other is in an outermost controller, so no non-determinism actually arises.

We assume that each component has at most one transition for an event from each state of the component (consistency).

It is an axiom that an internal event can only be processed if it is at the head of the queue:

$$\mathbf{AG}(\beta \;\Rightarrow\; \mathbf{event\_queue} \neq [] \;\wedge\; \mathbf{name}(\beta) = \mathbf{event\_queue}(1))$$

The head of the queue is removed when the corresponding event is processed, and replaced with some interleaving ρ of the generations of all enabled transitions triggered by this occurrence of β:

$$\forall\, \mathbf{q} \cdot \mathbf{AG}(\beta \wedge \mathbf{q} = \mathbf{event\_queue} \;\Rightarrow\; \mathbf{AX}(\mathbf{event\_queue} = \rho \frown \mathbf{tail}(\mathbf{q})))$$

This models the "run to completion" semantics of statecharts in UML [17].

A stable state is a configuration of **Sys** where $\mathbf{event\_queue} = []$. It can be shown that under certain assumptions, each specification formula φ valid in the coarse-grain semantics of **Sys** will be valid in the fine-grain semantics in the form

$$\mathbf{AG}(\mathbf{event\_queue} = [] \;\Rightarrow\; \varphi^*)$$

where φ^* is a relativisation of φ to the stable states of the concrete model: **AX** is interpreted as "in all next stable states", etc.

Notice that subcontrollers in fine-grain models need not be related in a purely hierarchical manner, as in the example of Figure 5. Events may even be sent to the same component in which the transition generating the event occurs. In the system of Figure 5 the successive values of **event_queue**, during the reaction to α from the initial state, are:

$[]$
$[\beta, \gamma]$
$[\beta_1, \beta_2, \beta_3, \gamma]$
$[\delta_2, \beta_2, \beta_3, \gamma]$
$[\beta_2, \beta_3, \gamma]$
$[\beta_3, \gamma]$
$[\gamma]$
$[\gamma_1, \gamma_2]$
$[\delta_1, \gamma_2]$
$[\gamma_2]$
$[\delta_2]$
$[]$

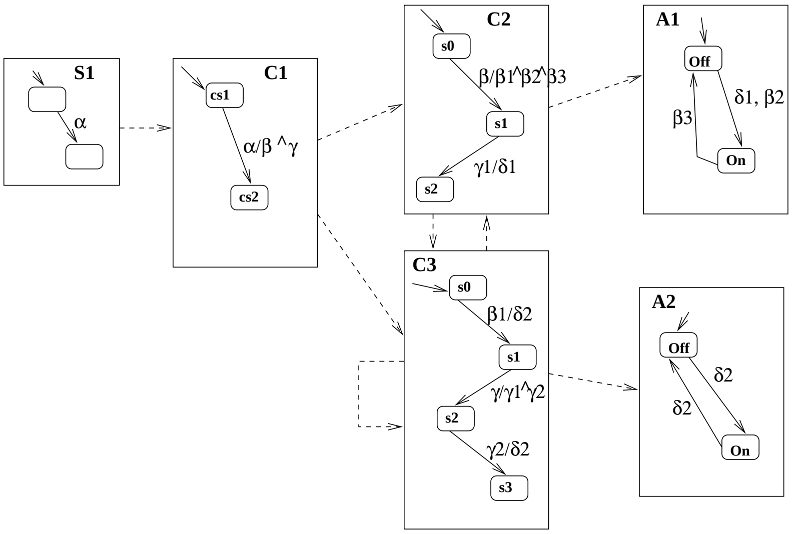

Figure 5: Example of Fine-Grain Semantics

To extend this semantics to the object-oriented version defined in Section 4, **event_queue** is defined as a sequence of pairs $(\alpha_{\mathbf{tok}}, \mathbf{a})$ of tokens for events, and object references. These represent events $\alpha(\mathbf{a})$ on object $\mathbf{a}$.

3 Refinement Transformations on RSDS Specifications

The main RSDS strategies for specification decomposition are horizontal, phase and hierarchical decompositions [11].

3.1 Horizontal Decomposition

Given an abstract RSDS system $\mathbf{T} = (\mathbf{S}, \mathbf{I}, \{\}, \mathbf{A})$, a horizontal decomposition is applicable if $\mathbf{A}$ can be split into two disjoint sets $\mathbf{A}_1$, $\mathbf{A}_2$ with $\mathbf{A}_1 \cup \mathbf{A}_2 = \mathbf{A}$ and if $\mathbf{I}$ can be expressed as the conjunction of two sets of invariants, $\mathbf{I}_1$ over $\mathbf{S}$ and $\mathbf{A}_1$, and $\mathbf{I}_2$ over $\mathbf{S}$ and $\mathbf{A}_2$ respectively. Ie, $\bigwedge \mathbf{I} \equiv \bigwedge \mathbf{I}_1 \wedge \bigwedge \mathbf{I}_2$.

If such a factoring is possible, $\mathbf{T}$ can be split into subsystems $\mathbf{T}_1 = (\mathbf{S}, \mathbf{I}_1, \{\}, \mathbf{A}_1)$ and $\mathbf{T}_2 = (\mathbf{S}, \mathbf{I}_2, \{\}, \mathbf{A}_2)$.

Example If **s1** and **s2** are binary switches and **btl** and **bm** binary actuators, and the system axioms are:

$$\begin{array}{l} \mathrm{s1} = \mathbf{On} \wedge \mathrm{s2} = \mathbf{Off} \;\Rightarrow\; \mathbf{bm} = \mathbf{On} \;\wedge\; \mathbf{btl} = \mathbf{red} \\ \mathrm{s1} = \mathbf{Off} \;\Rightarrow\; \mathbf{btl} = \mathbf{green} \\ \mathrm{s2} = \mathbf{On} \;\Rightarrow\; \mathbf{bm} = \mathbf{Off} \end{array}$$

then this system can be hierarchically decomposed into subsystems

$$(\{\mathrm{s1}, \mathrm{s2}\},\;\; \{\mathrm{s1} = \mathbf{On} \wedge \mathrm{s2} = \mathbf{Off} \;\Rightarrow\; \mathbf{btl} = \mathbf{red},\; \mathrm{s1} = \mathbf{Off} \;\Rightarrow\; \mathbf{btl} = \mathbf{green}\}, \{\}, \{\mathbf{btl}\})$$

and

$$(\{\mathrm{s1}, \mathrm{s2}\},\;\; \{\mathrm{s1} = \mathbf{On} \wedge \mathrm{s2} = \mathbf{Off} \;\Rightarrow\; \mathbf{bm} = \mathbf{On},\; \mathrm{s2} = \mathbf{On} \;\Rightarrow\; \mathbf{bm} = \mathbf{Off}\}, \{\}, \{\mathbf{bm}\})$$

However if we had included an axiom

$$\mathrm{s1} = \mathbf{On} \;\Rightarrow\; \mathbf{EF}(\mathbf{btl} = \mathbf{green} \wedge \mathbf{bm} = \mathbf{On})$$

the system could not be split.

3.2 Phase Decomposition

In this transformation, we select some sensor component **Sen** with more than 1 state, and divide the invariants **I** into sets $\mathbf{I_v}$ for each $\mathbf{v} \in \mathbf{States_{Sen}}$ formed as follows:

$$\mathbf{I_v} \;=\; \{\varphi \in \mathbf{I} \bullet \mathbf{sen} = \mathbf{v} \;\Rightarrow\; \varphi\}$$

The formulae in $\mathbf{I_v}$ will in general simplify to logically equivalent formulae with fewer variables than originally. For example

$$\mathbf{sen} = \mathbf{v} \;\Rightarrow\; (\mathbf{sen} = \mathbf{v}' \Rightarrow \mathbf{a} = \mathbf{aval})$$

reduces to **true** if $\mathbf{v}$ and $\mathbf{v}'$ are distinct states of **Sen**.

The corresponding set of actuators $\mathbf{A_v}$ is the subset of elements **A** which occur in some non-trivial invariant of $\mathbf{I_v}$: a trivial invariant is one that simplifies to **true** by equality axioms $\mathbf{not}(\mathbf{v} = \mathbf{v}')$ where $\mathbf{v}$ and $\mathbf{v}'$ are two distinct elements of **Sen**, and propositional logic.

3.3 Hierarchical Decomposition

As with horizontal decomposition, this is based on the groups of dependent variables in the system. A set of these are selected, with no two groups having a common actuator, and such that all system actuators are in some group. Using heuristics, those sensors which are 'local' to a given actuator, in the sense of co-occuring frequently in invariants with the actuator, are identified. A supervisor controller for the selected groups is generated, which contains all actions that require knowledge of local variables of two or more groups. Different subsystems therefore have no sensor variables in common, unlike the case of horizontal decomposition. They may have knowledge of each others state via 'virtual sensors' which communicate significant changes in combinations of sensor states of other subsystems to a particular subsystem.

3.4 Dynamic Phase Decomposition

This decomposition makes use of the class diagram of a system. Any entities **E** on this diagram which are not sensors or actuators, and which have a many-one association to some class **C** with multiple subclasses can be used as the basis of such a decomposition. The system is split into subcontrollers which manage particular **E** instances, with a different phase subcontroller for each subclass of **C** that the instance may be linked to.

4 Generalised Class Theories of Components

Typically there may be several isomorphic copies of a component in a system, for example each floor in a lift control system will have a lift request button, and each request button will have a corresponding light to indicate if the request is awaiting service. To model this we parameterise the instance theory $\Gamma_{\mathbf{C}}$ of a component **C** by a constant type of object identities @**C** for instances of **C** [4]. The resulting class theory $\Gamma\mathbf{C}_{\mathbf{C}}$ has attributes

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

for each attribute $\mathbf{att} : \mathbf{T}$ of **C**, and action symbol

$$\alpha : @\mathbf{C} \;\to$$

for each action symbol α of **C**. In addition there is a new attribute $\overline{\mathbf{C}} : \mathbb{F}@\mathbf{C}$ representing the set of existing instances of **C**, and actions $\mathbf{new_C} : @\mathbf{C} \;\to$ and $\mathbf{kill_C} : @\mathbf{C} \;\to$ to add and remove elements of @**C** from this set. Each axiom $\mathbf{AG}(\varphi)$ of $\Gamma_{\mathbf{C}}$ becomes an axiom of $\Gamma\mathbf{C}_{\mathbf{C}}$ in its parameterised form $\forall\, \mathbf{a} \cdot \mathbf{AG}(\mathbf{a} \in \overline{\mathbf{C}} \;\Rightarrow\; \varphi(\mathbf{a}))$. Initial state axioms φ become $\mathbf{AG}(\forall\, \mathbf{a} : @\mathbf{C} \cdot \mathbf{new_C}(\mathbf{a}) \Rightarrow \mathbf{AX}(\varphi(\mathbf{a})))$.

In addition there are the *sequentiality* axioms enforcing that only one instance of an event can occur at any time, and that different events of a component cannot co-occur:

1.

$$\mathbf{AG}(\forall\, \mathbf{a}, \mathbf{b} : \overline{\mathbf{C}} \cdot \mathbf{a} \neq \mathbf{b} \;\wedge\; \alpha(\mathbf{a}) \;\Rightarrow\; \neg\, \alpha(\mathbf{b}))$$

for α in $\mathbf{Events_C}$.

2.

$$\mathbf{AG}(\forall\, \mathbf{a}, \mathbf{b} : \overline{\mathbf{C}} \cdot \alpha(\mathbf{a}) \;\Rightarrow\; \neg\, \beta(\mathbf{b}))$$

where α, β are distinct elements of $\mathbf{Events_C}$.

and axioms for **new** and **kill**:

$$\begin{array}{l}
\mathbf{AG}(\forall\, \mathbf{x} : \mathbb{P}(@\mathbf{C}) \cdot \overline{\mathbf{C}} = \mathbf{x} \wedge \mathbf{new_C}(\mathbf{a}) \;\Rightarrow\; \mathbf{AX}(\overline{\mathbf{C}} = \mathbf{x} \cup \{\mathbf{a}\})) \\
\mathbf{AG}(\forall\, \mathbf{x} : \mathbb{P}(@\mathbf{C}) \cdot \overline{\mathbf{C}} = \mathbf{x} \wedge \mathbf{kill_C}(\mathbf{a}) \;\Rightarrow\; \mathbf{AX}(\overline{\mathbf{C}} = \mathbf{x} \setminus \{\mathbf{a}\})) \\
\mathbf{AG}(\forall\, \mathbf{x} : \mathbb{P}(@\mathbf{C}) \cdot \overline{\mathbf{C}} = \mathbf{x} \;\Rightarrow\; \forall\, \mathbf{a} : @\mathbf{C} \cdot \neg\, \mathbf{new_C}(\mathbf{a}) \wedge \neg\, \mathbf{kill_C}(\mathbf{a}) \;\Rightarrow\; \mathbf{AX}(\overline{\mathbf{C}} = \mathbf{x}))
\end{array}$$

If class $\mathbf{D}$ is a subclass of class $\mathbf{C}$ then $@\mathbf{D} \subseteq @\mathbf{C}$ and $\mathbf{AG}(\overline{\mathbf{D}} \subseteq \overline{\mathbf{C}})$.

An association or aggregation from class $\mathbf{C}$ to class $\mathbf{D}$ is modelled as a mathematical relation

$$\mathbf{r} : \overline{\mathbf{C}} \leftrightarrow \overline{\mathbf{D}}$$

ie, a set of pairs $(\mathbf{c}, \mathbf{d})$ of instances of $\mathbf{C}$ and $\mathbf{D}$. $\mathbf{r}$ is defined as an attribute of $\Gamma\mathbf{C_{Sys}}$ and actions $\mathbf{link_r}(\mathbf{c}, \mathbf{d})$ to add and $\mathbf{unlink_r}(\mathbf{c}, \mathbf{d})$ to remove pairs from it are also included in this theory, together with axioms defining their behaviour.

4.1 Generalised Theory of an RSDS System

The theory $\Gamma\mathbf{C_{Sys}}$ of a RSDS system $\mathbf{Sys} = (\{\mathbf{S}_1, \ldots, \mathbf{S_n}\}, \mathbf{Invs}, \{\}, \{\mathbf{A}_1, \ldots, \mathbf{A_m}\})$ is the co-limit of the class theories $\Gamma\mathbf{C_M}$ of its components, together with the global axioms that express that only one instance $\alpha(\mathbf{a})$ of any sensor event may occur in any time step, and that any instance of an actuator event must co-occur with some instance of a sensor event:

Mutual exclusion of sensor events is formally defined as the property

$$\mathbf{AG}(\forall\, \mathbf{x_i} \in \overline{\mathbf{S_i}};\; \mathbf{x_j} \in \overline{\mathbf{S_j}} \cdot \neg\, (\alpha(\mathbf{x_i}) \wedge \beta(\mathbf{x_j})))$$

for each $1 \leq \mathbf{i} \neq \mathbf{j} \leq \mathbf{n}$ and $\alpha \in \mathbf{Events_{S_i}}$, $\beta \in \mathbf{Events_{S_j}}$.

Coverage of actuator events by sensor events is the property

$$\begin{array}{l}
\mathbf{AG}(\forall\, \mathbf{z_i} \in \overline{\mathbf{A_i}} \cdot \alpha(\mathbf{z_i}) \;\Rightarrow \\
\qquad \exists\, \mathbf{x}_1 \in \overline{\mathbf{S}_1} \cdot \bigvee_{\beta \in \mathbf{Events_{S_1}}} \beta(\mathbf{x}_1) \;\vee\; \ldots \;\vee\; \exists\, \mathbf{x_n} \in \overline{\mathbf{S_n}} \cdot \bigvee_{\beta \in \mathbf{Events_{S_n}}} \beta(\mathbf{x_n}))
\end{array}$$

for each $1 \leq \mathbf{i} \leq \mathbf{m}$ and $\alpha \in \mathbf{Events_{A_i}}$.

Coverage of internal events by transitions which generate them is the property

$$\begin{array}{l}
\mathbf{AG}(\forall\, \mathbf{z_i} \in \overline{\mathbf{A_i}} \cdot \alpha(\mathbf{z_i}) \;\Rightarrow \\
\qquad \exists\, \mathbf{x}_1 \in \overline{\mathbf{C}_1} \cdot \mathbf{tr}_{1,1}(\mathbf{x}_1) \;\vee\; \ldots \;\vee\; \exists\, \mathbf{x_r} \in \overline{\mathbf{C_r}} \cdot \mathbf{tr}_{\mathbf{r},\mathbf{n_r}}(\mathbf{x_r}))
\end{array}$$

for each $1 \leq \mathbf{i} \leq \mathbf{m}$ and $\alpha \in \mathbf{Events_{A_i}}$, where the $\mathbf{tr_{j,l}}$ are all of the transitions with α in their generations. Similarly if α is an internal event of some controller.

The first design step is usually to introduce a *controller* component $\mathbf{C}$ intermediate between the sensors and actuators (top of Figure 6). The controller has no attributes apart from $\overline{\mathbf{C}}$ and has action symbols $\mathbf{kill_C}$ and $\mathbf{new_C}$, together with symbols for each event of each of the sensors.

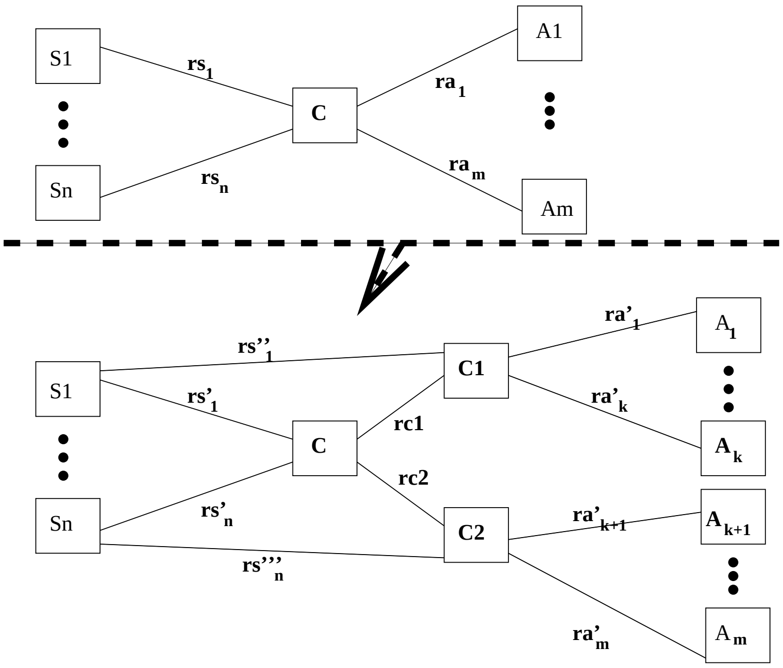

Figure 6: Typical Initial RSDS Design and Horizontal Decomposition

In the configuration of Figure 6, we may require that there is a single controller: $\overline{\mathbf{C}} = \{\mathbf{c}\}$ and that all the sensors and actuators are linked to this:

$$
\begin{aligned}
\mathbf{rs}_1 &= \overline{\mathbf{S}_1} \times \{\mathbf{c}\} \\
&\vdots \\
\mathbf{rs_n} &= \overline{\mathbf{S_n}} \times \{\mathbf{c}\} \\
\mathbf{ra}_1 &= \{\mathbf{c}\} \times \overline{\mathbf{A}_1} \\
&\vdots \\
\mathbf{ra_m} &= \{\mathbf{c}\} \times \overline{\mathbf{A_m}}
\end{aligned}
$$

For horizontal and hierarchical decomposition we normally assume that the system is static throughout its lifetime, ie, that $\mathbf{new_C}$ and $\mathbf{kill_C}$ actions never occur for any component $\mathbf{C}$. For phase decomposition these actions only occur to move a controller object from one subclass of the phase-split controller to another subclass. In dynamic phase decomposition **new** and **kill** may be applied to controller objects but not to sensors or actuators.

5 Proof of Correctness of Design Transformations

Using the above framework we can verify horizontal, phase, hierarchical and dynamic phase decompositions. These decompositions are usually carried out before explicit control algorithms are defined for controllers. We describe horizontal and heirarchical decompositions here. A refinement step from an RSDS system $\mathbf{Sys}_1 = (\mathbf{Sens}_1, \mathbf{Invs}_1, \mathbf{Cons}_1, \mathbf{Acts}_1)$ to a system $\mathbf{Sys}_2 = (\mathbf{Sens}_2, \mathbf{Invs}_2, \mathbf{Cons}_2, \mathbf{Acts}_2)$ is correct if there is a syntactic interpretation σ of $\mathbf{Sys}_1$ in $\mathbf{Sys}_2$ such that $\Gamma\mathbf{C}_{\mathbf{Sys}_2} \vdash \sigma(\varphi)$ for each axiom φ of $\Gamma\mathbf{C}_{\mathbf{Sys}_1}$. We say that $\Gamma\mathbf{C}_{\mathbf{Sys}_2}$ is a *theory extension* of $\Gamma\mathbf{C}_{\mathbf{Sys}_1}$.

5.1 Horizontal Decomposition

A typical general situation is shown in Figure 6, where controller $\mathbf{C}$ is decomposed into independent subcontrollers $\mathbf{C}_1$ and $\mathbf{C}_2$ which share the same set of sensors as $\mathbf{C}$ but have disjoint sets of actuators.

Assume that

$$\forall \mathbf{y} \in \overline{\mathbf{C}} \cdot \exists! \mathbf{y}_1 \in \overline{\mathbf{C}_1} \cdot (\mathbf{y}, \mathbf{y}_1) \in \mathbf{rc}_1$$

and

$$\forall \mathbf{y} \in \overline{\mathbf{C}} \cdot \exists! \mathbf{y}_2 \in \overline{\mathbf{C}_2} \cdot (\mathbf{y}, \mathbf{y}_2) \in \mathbf{rc}_2$$

Each $\mathbf{rs}''_\mathbf{i}$ relation is defined to be $\mathbf{rs}'_\mathbf{i};\ \mathbf{rc}_1$ in the first new submodel $\mathbf{Sys}_1$ and $\mathbf{rs}'''_\mathbf{i}$ is defined to be $\mathbf{rs}'_\mathbf{i};\ \mathbf{rc}_2$ in $\mathbf{Sys}_2$.

$\mathbf{rs}_\mathbf{i}$ in the original model **Sys** is interpreted by $\mathbf{rs}'_\mathbf{i}$ in $\mathbf{Sys}_1$, and $\mathbf{ra}_\mathbf{j}$ is interpreted by $\mathbf{rc}_1;\ \mathbf{ra}'_\mathbf{j}$ (if $1 \leq \mathbf{j} \leq \mathbf{k}$) or by $\mathbf{rc}_2;\ \mathbf{ra}'_\mathbf{j}$ (if $\mathbf{k} < \mathbf{j} \leq \mathbf{m}$). This does yield a theory extension provided that each axiom

$$\begin{array}{l} (\mathbf{x}_1, \mathbf{y}) \in \mathbf{rs}_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{x_n}, \mathbf{y}) \in \mathbf{rs_n} \ \wedge \\ (\mathbf{y}, \mathbf{z}_1) \in \mathbf{ra}_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{y}, \mathbf{z_m}) \in \mathbf{ra_m} \ \Rightarrow \quad \varphi(\mathbf{x}_1, \ldots, \mathbf{x_n}, \mathbf{z}_1, \ldots, \mathbf{z_m}) \end{array}$$

of **Sys** is also an axiom of either the $\mathbf{Sys}_1$ subsystem, in the form

$$\begin{array}{l} (\mathbf{x}_1, \mathbf{y}_1) \in \mathbf{rs}''_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{x_n}, \mathbf{y}_1) \in \mathbf{rs}''_\mathbf{n} \ \wedge \\ (\mathbf{y}_1, \mathbf{z}_1) \in \mathbf{ra}'_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{y}_1, \mathbf{z_k}) \in \mathbf{ra}'_\mathbf{k} \ \Rightarrow \quad \varphi(\mathbf{x}_1, \ldots, \mathbf{x_n}, \mathbf{z}_1, \ldots, \mathbf{z_k}) \end{array}$$

or of the $\mathbf{Sys}_2$ subsystem, in the form

$$\begin{array}{l} (\mathbf{x}_1, \mathbf{y}_2) \in \mathbf{rs}'''_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{x_n}, \mathbf{y}_2) \in \mathbf{rs}'''_\mathbf{n} \ \wedge \\ (\mathbf{y}_2, \mathbf{z_{k+1}}) \in \mathbf{ra}'_\mathbf{k+1} \ \wedge \ \ldots \ \wedge \ (\mathbf{y}_2, \mathbf{z_m}) \in \mathbf{ra}'_\mathbf{m} \ \Rightarrow \quad \varphi(\mathbf{x}_1, \ldots, \mathbf{x_n}, \mathbf{z_{k+1}}, \ldots, \mathbf{z_m}) \end{array}$$

depending on which set of actuators is involved in φ.

This holds because, if φ is of the first form, ie, involves only actuators from $\mathbf{A}_1$ to $\mathbf{A_k}$, then if

$$\begin{array}{l} (\mathbf{x}_1, \mathbf{y}) \in \mathbf{rs}'_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{x_n}, \mathbf{y}) \in \mathbf{rs}'_\mathbf{n} \ \wedge \\ (\mathbf{y}, \mathbf{z}_1) \in \mathbf{rc}_1;\ \mathbf{ra}'_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{y}, \mathbf{z_k}) \in \mathbf{rc}_1;\ \mathbf{ra}'_\mathbf{k} \end{array}$$

then there is a unique element $\mathbf{y}_1$ such that $(\mathbf{y}, \mathbf{y}_1)$ is in $\mathbf{rc}_1$ so for this $\mathbf{y}_1$:

$$(\mathbf{x}_1, \mathbf{y}_1) \in \mathbf{rs}''_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{x_n}, \mathbf{y}_1) \in \mathbf{rs}''_\mathbf{n}$$

and

$$(\mathbf{y}_1, \mathbf{z}_1) \in \mathbf{ra}'_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{y}_1, \mathbf{z_k}) \in \mathbf{ra}'_\mathbf{k}$$

so the result follows. Similarly for $\mathbf{Sys}_2$ axioms. The global invariants of $\mathbf{Sys}_1$ and $\mathbf{Sys}_2$ together prove those of **Sys**: mutual exclusion of all sensor events holds since $\mathbf{Sys}_1$ and $\mathbf{Sys}_2$ have the same set of sensors and hence exclusion axioms as **Sys**. Also each actuator of **Sys** is either in $\mathbf{Sys}_1$ and so its events are covered by some sensor event (of $\mathbf{Sys}_1$ and hence **Sys**) or in $\mathbf{Sys}_2$ (likewise).

5.2 Phase Decomposition

Figure 7 shows the typical situation in this case. Assume that **Sen** is the sensor on whose states we are going to split the controller into phases. For example, assume $\mathbf{States_{Sen}} = \{\mathbf{v}_1, \mathbf{v}_2\}$. Define

$$\begin{array}{lcl} \overline{\mathbf{C1}} & = & \{\mathbf{y} : \overline{\mathbf{C}} \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{rsen} \ \Rightarrow \ \mathbf{sen}(\mathbf{x}) = \mathbf{v}_1\} \\ \overline{\mathbf{C2}} & = & \{\mathbf{y} : \overline{\mathbf{C}} \mid (\mathbf{x}, \mathbf{y}) \in \mathbf{rsen} \ \Rightarrow \ \mathbf{sen}(\mathbf{x}) = \mathbf{v}_2\} \end{array}$$

rsen must be such that these two sets are a partition of $\overline{\mathbf{C}}$, this is the case if **rsen** is surjective and $\mathbf{rsen}^{-1}$ is a function, for example, if $\overline{\mathbf{Sen}}$ has a unique element. When a **Sen** element changes state to $\mathbf{v}_1$, the system must be reconfigured so that all the sensors and actuators are connected to an element of **C1** instead of an element of **C2**, etc.

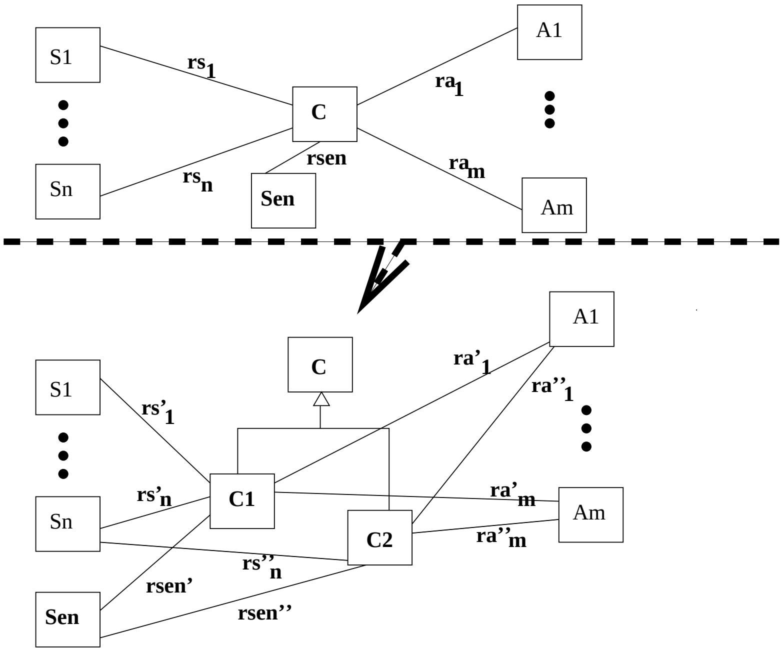

Figure 7: Phase Decomposition

The system $\mathbf{Sys}_1$ comprising $\mathbf{C}1$ and the sensors and actuators must satisfy

$$\begin{aligned}&(\mathbf{x},\mathbf{y}) \in \mathbf{rsen}' \ \wedge \ (\mathbf{x}_1,\mathbf{y}) \in \mathbf{rs}'_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{x_n},\mathbf{y}) \in \mathbf{rs}'_\mathbf{n} \ \wedge \\ &(\mathbf{y},\mathbf{z}_1) \in \mathbf{ra}'_1 \ \wedge \ \ldots \ \wedge \ (\mathbf{y},\mathbf{z_m}) \in \mathbf{ra}'_\mathbf{m} \\ &\qquad\qquad \Rightarrow \ (\mathbf{sen}(\mathbf{x}) = \mathbf{v}_1 \ \Rightarrow \ \varphi(\mathbf{x},\mathbf{x}_1,\ldots,\mathbf{x_n},\mathbf{z}_1,\ldots,\mathbf{z_m}))\end{aligned}$$

for each invariant φ of the original system $\mathbf{Sys}$. Similarly for $\mathbf{Sys}_2$.

The interpretation of $\mathbf{rsen}$ is $\mathbf{rsen}' \ \cup \ \mathbf{rsen}''$ and likewise for all other relations and $\overline{\mathbf{C}}$, which is interpreted as $\overline{\mathbf{C}1} \cup \overline{\mathbf{C}2}$.

5.3 Hierarchical Decomposition

Figure 8 shows a typical situation in this case, where a system $\mathbf{Sys}$ is split into two subsystems, and a new 'virtual sensor' component $\mathbf{Vs}$ is introduced as an interface between one system $\mathbf{Sys}_1$ consisting of $\mathbf{S}1$, $\mathbf{S}2$, $\mathbf{C}1$ and $\mathbf{A}2$, and a subordinate system $\mathbf{Sys}_2$ consisting of $\mathbf{S}0$, $\mathbf{Vs}$, $\mathbf{C}2$ and $\mathbf{A}1$. We will give the proof for this case, similar cases can be handled in the same way provided the assumptions identified below are valid.

The role of the controller $\mathbf{C}1$ is to maintain some invariant between the states of $\mathbf{Vs}$ and the sensors which it conceals from $\mathbf{C}2$ (in addition to carrying out its own control function for actuators it directly controls). For example $\mathbf{Vs}$ elements could express a "conjunction" of the states of these sensors:

$$\mathbf{vs}(\mathbf{x}) = \mathbf{On} \ \Leftrightarrow \ \mathrm{s}1(\mathbf{x}1) = \mathbf{On} \ \wedge \ \mathrm{s}2(\mathbf{x}2) = \mathbf{On}$$

when $\mathbf{x}$, $\mathbf{x}1$ and $\mathbf{x}2$ are connected to the same controller $\mathbf{c}1 \in \overline{\mathbf{C}1}$.

So assume there is a function

$$\mathbf{f} : \mathbf{States}_{\mathbf{S}1} \times \mathbf{States}_{\mathbf{S}2} \rightarrow \mathbf{States}_{\mathbf{Vs}}$$

such that

$$(\mathbf{x}1,\mathbf{y}) \in \mathbf{rs}1' \ \wedge \ (\mathbf{x}2,\mathbf{y}) \in \mathbf{rs}2' \ \wedge \ (\mathbf{y},\mathbf{x}) \in \mathbf{rva} \ \Rightarrow \ \mathbf{vs}(\mathbf{x}) = \mathbf{f}(\mathbf{s}1(\mathbf{x}1), \mathbf{s}2(\mathbf{x}2))$$

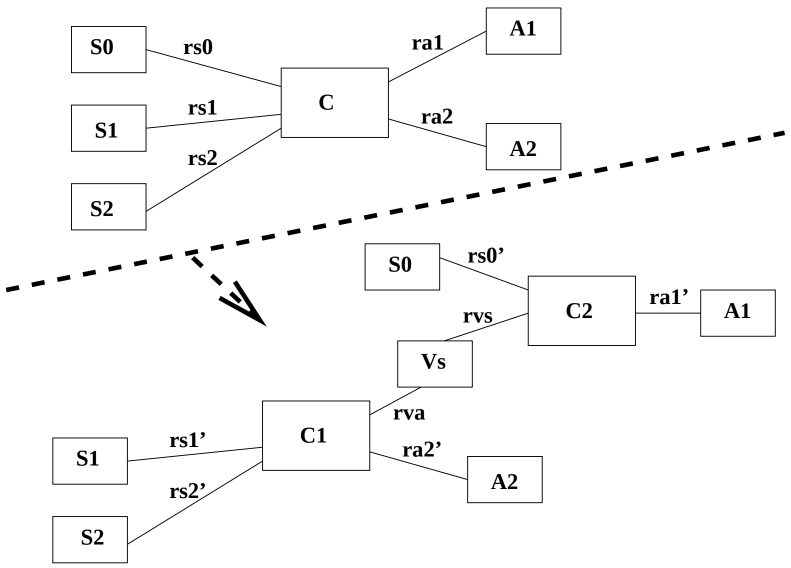

Figure 8: Hierarchical Decomposition

$\mathbf{C}2$ must maintain an invariant $\varphi'(\mathbf{x}, \mathbf{x}0, \mathbf{z}1)$ for each invariant $\varphi(\mathbf{x}0, \mathbf{x}1, \mathbf{x}2, \mathbf{z}1)$ of $\mathbf{Sys}$ which relates any of the sensors $\mathbf{S}0$, $\mathbf{S}1$, $\mathbf{S}2$ to the actuator $\mathbf{A}1$, where $\mathbf{z}1 \in \overline{\mathbf{A1}}$ and φ' is such that

$$\mathbf{vs}(\mathbf{x}) = \mathbf{f}(\mathrm{s1}(\mathbf{x}1), \mathrm{s2}(\mathbf{x}2)) \;\wedge\; \varphi'(\mathbf{x}, \mathbf{x}0, \mathbf{z}1) \;\Rightarrow\; \varphi(\mathbf{x}0, \mathbf{x}1, \mathbf{x}2, \mathbf{z}1)$$

$\mathbf{C}1$ maintains those invariants of $\mathbf{Sys}$ which relate any of $\mathbf{S}1$ and $\mathbf{S}2$ to the actuator $\mathbf{A}2$. There should be no other invariants in $\mathbf{Sys}$ (ie, no invariant relates $\mathbf{A}2$ to $\mathbf{S}0$).

The interpretation of $\overline{\mathbf{C}}$ is the set

$$\{(\mathbf{y}1, \mathbf{x}, \mathbf{y}2) \mid \mathbf{y}1 \in \overline{\mathbf{C1}} \wedge \mathbf{x} \in \overline{\mathbf{Vs}} \wedge \mathbf{y}2 \in \overline{\mathbf{C2}} \wedge (\mathbf{y}1, \mathbf{x}) \in \mathbf{rva} \wedge (\mathbf{x}, \mathbf{y}2) \in \mathbf{rvs}\}$$

The interpretation $\sigma(\mathbf{rs}0)$ of $\mathbf{rs}0$ is

$$\{(\mathbf{x}0, (\mathbf{y}1, \mathbf{x}, \mathbf{y}2)) \mid (\mathbf{x}0, \mathbf{y}2) \in \mathbf{rs}0' \wedge (\mathbf{y}1, \mathbf{x}) \in \mathbf{rva} \wedge (\mathbf{x}, \mathbf{y}2) \in \mathbf{rvs}\}$$

The interpretation $\sigma(\mathbf{rs}1)$ of $\mathbf{rs}1$ is

$$\{(\mathbf{x}1, (\mathbf{y}1, \mathbf{x}, \mathbf{y}2)) \mid (\mathbf{x}1, \mathbf{y}1) \in \mathbf{rs}1' \wedge (\mathbf{y}1, \mathbf{x}) \in \mathbf{rva} \wedge (\mathbf{x}, \mathbf{y}2) \in \mathbf{rvs}\}$$

Similarly for $\mathbf{rs}2$. The interpretation of $\mathbf{ra}1$ is

$$\{((\mathbf{y}1, \mathbf{x}, \mathbf{y}2), \mathbf{z}1) \mid (\mathbf{y}2, \mathbf{z}1) \in \mathbf{ra}1' \wedge (\mathbf{y}1, \mathbf{x}) \in \mathbf{rva} \wedge (\mathbf{x}, \mathbf{y}2) \in \mathbf{rvs}\}$$

The interpretation of $\mathbf{ra}2$ is

$$\{((\mathbf{y}1, \mathbf{x}, \mathbf{y}2), \mathbf{z}2) \mid (\mathbf{y}1, \mathbf{z}2) \in \mathbf{ra}2' \wedge (\mathbf{y}1, \mathbf{x}) \in \mathbf{rva} \wedge (\mathbf{x}, \mathbf{y}2) \in \mathbf{rvs}\}$$

Then $\mathbf{Sys}$ is correctly implemented by the combination of $\mathbf{Sys}_1$ and $\mathbf{Sys}_2$ since if

$$\begin{array}{l}(\mathbf{x}0, (\mathbf{y}1, \mathbf{x}, \mathbf{y}2)) \in \sigma(\mathbf{rs}0) \;\wedge\; (\mathbf{x}1, (\mathbf{y}1, \mathbf{x}, \mathbf{y}2)) \in \sigma(\mathbf{rs}1) \;\wedge \\ (\mathbf{x}2, (\mathbf{y}1, \mathbf{x}, \mathbf{y}2)) \in \sigma(\mathbf{rs}2) \;\wedge\; ((\mathbf{y}1, \mathbf{x}, \mathbf{y}2), \mathbf{z}1) \in \sigma(\mathbf{ra}1) \;\wedge \\ ((\mathbf{y}1, \mathbf{x}, \mathbf{y}2), \mathbf{z}2) \in \sigma(\mathbf{ra}2)\end{array}$$

then

$$\begin{array}{l}(\mathbf{x}0, \mathbf{y}2) \in \mathbf{rs}0' \wedge (\mathbf{x}1, \mathbf{y}1) \in \mathbf{rs}1' \wedge (\mathbf{x}2, \mathbf{y}1) \in \mathbf{rs}2' \;\wedge \\ (\mathbf{y}1, \mathbf{x}) \in \mathbf{rva} \wedge (\mathbf{x}, \mathbf{y}2) \in \mathbf{rvs} \;\wedge \\ (\mathbf{y}2, \mathbf{z}1) \in \mathbf{ra}1' \wedge (\mathbf{y}1, \mathbf{z}2) \in \mathbf{ra}2'\end{array}$$

But then $\mathbf{vs(x)} = \mathbf{f(s1(x1), s2(x2))}$ and $\varphi'(\mathbf{x}, \mathbf{x0}, \mathbf{z1})$ hold, so $\varphi(\mathbf{x0}, \mathbf{x1}, \mathbf{x2}, \mathbf{z1})$ holds. Similarly each invariant involving **A**2 holds since **C**1 ensures these.

Coverage of actuator events of **A**1 and **A**2 by sensor events is valid in the refined model since each **A**1 event is covered by a **S**0 event or **Vs** event (by system locality of $\mathbf{Sys}_2$), and each **Vs** event is covered by a **S**1 event or **S**2 event (system locality of $\mathbf{Sys}_1$). Each **A**2 event is covered by a **S**1 or **S**2 event by system locality of **Sys**1.

For sensor exclusion to hold, we need to require that each sensor event of **S**1 or **S**2 leads to an event of **Vs** in the same step. Events of **S**1 and **S**2 exclude each other by sensor exclusion for $\mathbf{Sys}_1$, and events of **S**0 and **Vs** exclude each other by sensor exclusion for $\mathbf{Sys}_2$. Therefore since an event of **S**1 or **S**2 would produce a **Vs** event, **S**0 events also exclude those of **S**1 and **S**2.

This assumption is not unreasonable – in the case that **Vs** expresses the conjunction of **S**1 and **S**2 for example, there would be invariants

$$
\begin{array}{l}
\mathrm{s1}\mathbf{on}(\mathrm{x1}) \wedge \mathrm{s2}(\mathrm{x2}) = \mathbf{Off} \;\Rightarrow\; \mathbf{goOff}(\mathrm{x}) \\
\mathrm{s1}\mathbf{on}(\mathrm{x1}) \wedge \mathrm{s2}(\mathrm{x2}) = \mathbf{On} \;\Rightarrow\; \mathbf{goOn}(\mathrm{x}) \\
\mathrm{s1}\mathbf{off}(\mathrm{x1}) \;\Rightarrow\; \mathbf{goOff}(\mathrm{x}) \\
\mathrm{s2}\mathbf{on}(\mathrm{x2}) \wedge \mathrm{s1}(\mathrm{x1}) = \mathbf{Off} \;\Rightarrow\; \mathbf{goOff}(\mathrm{x}) \\
\mathrm{s2}\mathbf{on}(\mathrm{x2}) \wedge \mathrm{s1}(\mathrm{x1}) = \mathbf{On} \;\Rightarrow\; \mathbf{goOn}(\mathrm{x}) \\
\mathrm{s2}\mathbf{off}(\mathrm{x2}) \;\Rightarrow\; \mathbf{goOff}(\mathrm{x})
\end{array}
$$

resulting in a **Vs** event occurrence for each event occurrence of either **S**1 or **S**2.

5.4 Dynamic Phase Decomposition

In the case of horizontal and hierarchical decompositions, the abstract and concrete versions of a system had similar relations. In particular, any addition of a sensor, controller pair $(\mathbf{x}, \mathbf{y})$ to an abstract relation requires a corresponding addition to its interpreting relation in the concrete system, and likewise for controller-actuator relations.

In contrast, dynamic phase decomposition may introduce a complex set of dynamic relationships between controllers, sensors and actuators. For example, in a basic version of the production cell case study, we could define a controller which responds to a **s2on** event (blank reaches end of feedbelt) by switching off the belt motor, and requesting a robot arm to move out to collect the blank if it is free (Figure 9). Blanks are represented as an entity with a lifecycle which passes through different states: on the feedbelt,

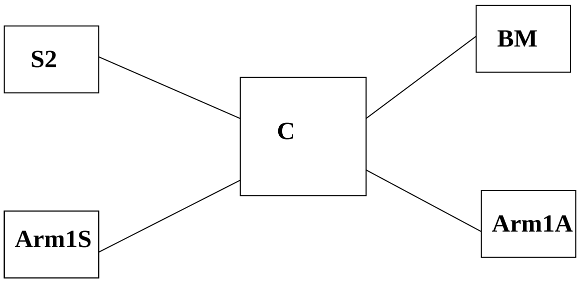

Figure 9: Abstract System Structure for Basic Production Cell

at end of feedbelt, on arm 1, etc, so can be used as the basis of a phase decomposition.

A controller is associated with each blank in the system, the connections between this controller and the sensors and actuators change as the blank moves through the system. Each controller in the **OnFeedbelt** and **AtFeedbeltEnd** subclasses of **Blank** is attached to a **S**2 instance and each **OnFeedbelt** controller is attached to the **BM** (beltmotor) instance of the system, for example, and an instance in **AtFeedbeltEnd** is connected to the **Arm1S** sensor and **Arm1A** actuator. Figure 10 shows the refined system structure.

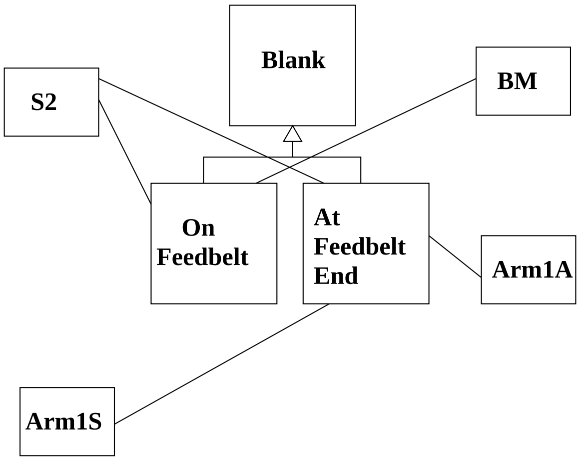

Figure 10: Refined System Structure for Basic Production Cell

6 Extension to Full UML Statecharts

In general UML state models [17] generated actions may either be synchronous or asynchronous. The former must be executed to completion before their caller can progress to generate the next action, whilst the latter immediately release the caller to progress to the next action.

We can model this by adding an attribute

$$\mathbf{event\_set} : \mathbb{F}(\mathbf{EventToken})$$

which holds the current set of asynchronous events that have been generated but not yet consumed.

For example, if a transition for event α generates events β; γ and β triggers a transition with generations β_1; β_2 with β_2 asynchronous, and γ triggers a transition with generations γ_1; γ_2, then the change of the event queue and event set with each step after α is accepted are as follows:

event_queue	**event_set**
$[\beta, \gamma]$	$\{\}$
$[\beta_1, \beta_2, \gamma]$	$\{\}$
$[\gamma]$	$\{\beta_2\}$
$[\gamma_1, \gamma_2]$	$\{\beta_2\}$
$[\gamma_2]$	$\{\beta_2\}$
$[\gamma_2]$	$\{\}$
$[]$	$\{\}$

A system can select either the event at the head of the queue, or any event in the event set to respond to at any step:

$$\mathbf{AG}(\alpha \;\Rightarrow\; \mathbf{name}(\alpha) \in \mathbf{event\_set} \lor (\mathbf{event\_queue} \neq [] \land \mathbf{name}(\alpha) = \mathbf{event\_queue}(1)))$$

The first element in **event_queue** must always be a synchronous event. This means that whenever an event is removed from the head of the queue $\mathbf{q}$, and replaced by a sequence ρ, we must remove all asynchronous events from the front of $\rho \frown \mathbf{q}$ and add them to **event_set**:

$$\mathbf{AG}(\mathbf{tr} \land \mathbf{event\_queue} \neq [] \land \mathbf{name}(\alpha) = \mathbf{event\_queue}(1) \;\Rightarrow\; \mathbf{AX}(\mathbf{event\_queue} = \mathbf{q}' \land \mathbf{event\_set} = \mathbf{s}'))$$

where **tr** is (the unique transition) triggered by α in the current step, $\mathbf{q}$ is the value of the event queue at the start of the step, and $\mathbf{s}$ the value of the event set at the start of the step, and ρ the sequence of tokens of events generated by **tr**. $\mathbf{s}' = \mathbf{s} \cup \mathrm{ran}(\mathbf{a})$ and $\rho \frown \mathbf{tail}(\mathbf{q}) \;=\; \mathbf{a} \frown \mathbf{q}'$ where $\mathbf{a}$ is the maximal sequence of asynchronous events at the front of $\rho \frown \mathbf{tail}(\mathbf{q})$.

Likewise, if an event from the event set is selected, we have:

$$\mathbf{AG}(\mathbf{tr} \wedge \mathbf{name}(\alpha) \in \mathbf{event\_set} \;\Rightarrow\; \mathbf{AX}(\mathbf{event\_queue} = \mathbf{q}' \wedge \mathbf{event\_set} = \mathbf{s}'))$$

where **tr** is (the unique transition) triggered by α in the current step, **q** is the value of the event queue at the start of the step, and **s** the value of the event set at the start of the step, and ρ is the sequence of tokens of events generated by **tr**. $\mathbf{s}' = \mathbf{s} - \{\mathbf{name}(\alpha)\} \cup \mathrm{ran}(\mathbf{a})$ and $\rho \frown \mathbf{q} \;=\; \mathbf{a} \frown \mathbf{q}'$ where **a** is the maximal sequence of asynchronous events at the front of $\rho \frown \mathbf{q}$.

7 Verification of RSDS Models

RSDS models may be verified by translation to B AMN and use of the B Toolkit [3] to check that required invariants are maintained by the synthesised control algorithms. This should always be the case. However temporal properties cannot be checked using B, and may not be ensured by the synthesis process. For such properties we can use model checkers such as SMV [5]. In the following sections we define translations from the coarse and fine-grain models of RSDS into SMV to support such verification.

7.1 Translation of Coarse-grain Semantics to SMV

The general form of the SMV modules generated using the coarse-grain semantics of an RSDS specification is as follows. For a system consisting of sensors $\mathbf{S}_1, \ldots, \mathbf{S_n}$ we construct a controller state machine **C** whose states are all tuples $(\mathbf{x}_1, \ldots, \mathbf{x_n})$ of states $\mathbf{x}_1 \in \mathbf{States}_{\mathbf{S}_1}, \ldots, \mathbf{x_n} \in \mathbf{States}_{\mathbf{S_n}}$ which satisfy the static system and environment invariants. Let $\mathbf{c}_1, \ldots, \mathbf{c_m}$ be an enumeration of these tuples. The transitions of **C** are the transitions of the AND-composition of $\mathbf{S}_1, \ldots, \mathbf{S_n}$. From the system invariants we derive the actuator settings which must hold in each state of **C** (the invariants of the RSDS specification are incomplete if there is not such a unique setting for each state of **C**).

The SMV module for **C** is then generated from its statemachine:

```
MODULE Controller
VAR
  global_state: { c1, ..., cm };
  con_event: { e1, ..., er };
DEFINE
  CT1 := con_event = e1 & global_state = c1;
  ...
ASSIGN
  init(global_state) := c_init;

  next(global_state) :=
    case
      CT1: c2;
      ...
      1: global_state;
    esac;
```

The **CT** represent all transitions of **C**, eg, the transition for $\mathbf{e1} : \mathbf{c1} \rightarrow \mathbf{c2}$ in the above schema.

Sensors are represented by modules

```
MODULE S1(C)
VAR
  s1: { ss1, ..., ssp };
DEFINE
  s1T0 := C.con_event = s1on & s1 = ss1;
  ...
ASSIGN
  init(s1) := s1_init;
```

```
  next(s1) :=
    case
      s1T0: st1;
      ...
      1: s1;
    esac;
```

Each controller event which is also an event of $\mathbf{S}_1$ is used in a definition for the LHS of a case in the **next** statement. The corresponding RHS is the projection of the resulting global state onto $\mathbf{S}_1$.

For actuators the same pattern is followed, except that the **next** statement uses the actuator settings which results from each global transition.

Finally there is a **main** module which combines the separate component modules and links them together:

```
MODULE main
VAR
  C: Controller;
  Ms1: S1(C);
  ..
SPEC AG(AG(Env) -> T1)
...
SPEC AG(AG(Env) -> Tr)
```

Env is the conjunction of environmental assumptions. The $\mathbf{T_i}$ are all temporal system invariants. The SPEC clauses of **main** must evaluate to true for the specification to be correct.

Figure 11 shows how the RSDS system structure maps to a corresponding structure of SMV modules.

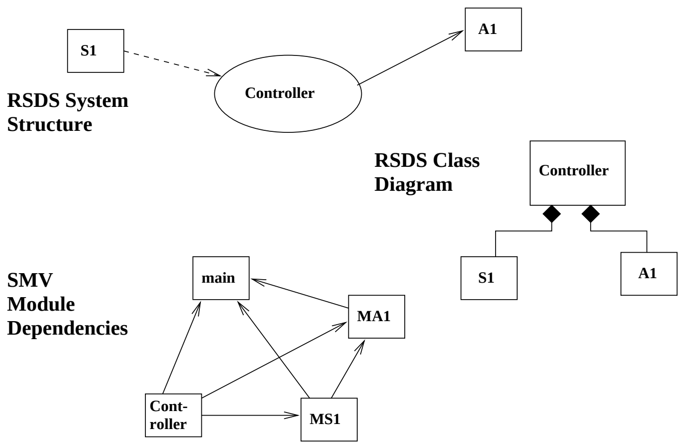

Figure 11: RSDS and SMV module structures

7.2 Translation of Fine-grain Semantics to SMV

For the fine-grain models the translation of sensors and controller is similar to the coarse-grain case, however **event_queue** is represented as an additional variable, in a **Queue** module, and the guard representing $\mathbf{Q}.\mathbf{event\_queue} = []$ is added to each controller and sensor transition. For subcontrollers and actuators, transition definitions for internal events β include guards $\mathbf{Q}.\mathbf{event\_queue} \neq [] \ \wedge \ \beta\mathbf{\_tok} = \mathbf{Q}.\mathbf{event\_queue}(1)$. Details are in [14].

Conclusions

We have shown that a useful subset of UML notation can be formally defined in a coherent manner, so that statechart and class diagram models have semantic representations in the same logical language. We have shown applications of this semantics in supporting temporal verification of reactive systems. A large number of case studies using RSDS have been carried out and tool support for RSDS is under development.

References

[1] K. Androutsopoulos, K. Lano, *Reactive System Specification and Verification in RSDS*, AVOCS '01, Oxford University, May 2001.

[2] B. Berard et al, *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer-Verlag, 1999.

[3] The B Toolkit, B Core UK Ltd, http://www.b-core.co.uk/.

[4] J C Bicarregui, K C Lano, T S E Maibaum, *Objects, Associations and Subsystems: a hierarchical approach to encapsulation*, ECOOP 97, LNCS, 1997.

[5] J R Burch, E M Clarke, K L McMillan, D L Dill, J Hwang, *Symbolic Model Checking:* 10^{20} *States and Beyond*, Proceedings of the Fifth Annual Symposium on Logic in Computer Science, 1990.

[6] E. Clarke, W. Heinle, *Modular Translation of Statecharts to SMV*, CMU report, August 2000.

[7] J Fiadeiro and T Maibaum. Describing, Structuring and Implementing Objects, in de Bakker *et al.*, **Foundations of Object Oriented languages**, LNCS 489, Springer-Verlag, 1991.

[8] D. Gries, *The Science of Programming*, Springer-Verlag, 1981.

[9] S. Goldsack and J. Kramer, *Invariants in the Application-oriented Specification of Control Systems*, Automatica, Vol. 18, No. 2, pp. 71 – 76, 1982.

[10] S. Goldsack, K. Lano, *Annealing and Data Decomposition in* VDM^{++}, ACM SIGPLAN Notices, Vol. 31, No. 4, April 1996.

[11] K. Lano, K. Androutsopoulos, D. Clark, *Structuring and Design of Reactive Systems using RSDS and B*, FASE 2000, LNCS, Springer-Verlag, 2000.

[12] K. Lano, K. Androutsopolous, *Structuring Reactive System Specifications in B AMN*, ICFEM 2000, IEEE Computer Society Press, 2000.

[13] K. Lano, D. Clark, K. Androutsopoulos, P. Kan, *Invariant-based Synthesis of Fault-tolerant Systems*, FTRTFT 2000.

[14] K. Lano, D. Clark, K. Androutsopoulos, P. Kan, *Semantic Foundations of RSDS*, King's College London, 2001.

[15] C. Lewerentz, T. Lindner (eds.), *Formal Development of Reactive Systems*, LNCS Vol. 891, Springer-Verlag, 1995.

[16] SHOLIS case study, Praxis, 2000.

[17] UML Version 1.3 Semantics, at `http://www.rational.com/uml`.