Variations on the Semantics of Graphical Models for Reactive Systems

Charles ANDRÉ, Jean-Paul RIGAULT

I3S Laboratory,
University of Nice Sophia Antipolis / CNRS,
BP 121, Sophia Antipolis cedex, 06903, FRANCE,
andre@unice.fr, jpr@essi.fr

Abstract—**The goal of this paper is to study concepts underlying graphical instant-based models. They are useful for reactive system modeling. They should be formal models (mathematical semantics). Some are able to express in an unambiguous and concise way complex reactive behaviors. The expressiveness may be an impediment to the readability and the intuitive understanding. We analyze the influence of some primitives on the computation (compilation or interpretation) of the behavior.**

Keywords—**Reactive system, semantics, state-transition system, visual modeling, synchronous programming.**

I. Introduction

Control-dominated systems are frequent in real time applications, especially in embedded ones (automobile, air and space, process control, integrated manufacturing, robotics...). They are reactive[1], which implies to deal with concurrency, communication, and pre-emption. They have to satisfy stringent constraints (correctness, response time), thus they have better be deterministic. Finally they may be life or mission critical and thus, they should be submitted to formal verification and validation. As a consequence, they require formal semantics.

The reactive nature and its consequences are the major reasons why the behavior of these systems is delicate to handle. A way of reducing the behavioral complexity is to have them evolve through successive phases, called *instants*, and the corresponding models are qualified as *instant-based*. Indeed, during an instant, the time seems to be suspended (the external events are frozen). The so-called synchronous models[2] are among these instant-based schemes.

The reactive systems are *discrete-event* systems. They are mostly event-driven, which means that they perform little processing on their own and that their behavior can be represented as a sequence of reactions to stimuli.

A classical way of representing sequential evolutions is to recurse to state-transition models. However the simplest forms of automata are not convenient to cope with the complexity of modern applications. Representing the behavior of such systems implies hierarchical description, support of concurrency and synchronization, and communication between the different parts of the system. In our approach, communication and synchronization are unified under the concept of *signals*. The stimuli that provoke reactions are associated with emission and reception of signals. Signals are also the actors of preemption, making it possible to suspend (to freeze) or to abort the behavior of some parts of the system.

A major problem with (concurrent) reactive system is to control their internal state changes, the duration of their transitions, and the information available during the transition itself.

We use a simplifying and yet fruitful hypothesis: the system evolves only during discrete phases (instant), with strictly defined start and end marks. During one phase, only the external events which were present at the beginning of the phase and the internal events that occurred as a consequence of the first ones are considered. The phase ends either when some stability (fixed-point) has been achieved or when some external "clock" decides that it is over.

This paper focuses on instant-based and state-based models, and variations on their semantics. In particular, we explore different ways of computing the fixed point. In the next section we introduce the key notions of signal and instant. The third section deals with the syntax of extended state-transition models while the fourth one is devoted to semantic issues.

II. Signals and Instants

A. Signals

Signal is our unique abstraction for handling communication and synchronization. Emitting or receiving a signal meets with the classical notion of *event*. A signal has a *presence status*: present ($+$), absent ($-$), or unknown ($\perp$). It may convey a *value* of a given type. If the signal can be multiply emitted within one instant, a combination function is also required. Finally, a signal has a *scope*: external (input or output), or local (bi-directional).

B. Instants

An instant, an atomic reaction phase in our model, is characterized by a Begin Of Instant (BoI) and an End Of Instant (EoI). BoI and EoI are in monotonic order and stricly alternating (instants cannot intersect). The input events are frozen at BoI; the output events are available at EoI. Between two instants (more specifically, between EoI_i and BoI_{i+1}) nothing happens. Instants define a *logical* time and give a clear meaning to simultaneity (everything which happens during the same instant), and to past and future. "Strict future" designates the next instant(s).

During an instant, the various (concurrent) parts of the system can produce and absorb signals. A signal (external as well as local) is broadcast: its target is not explicitly designated and it is available for any process. Broadcast contrasts with point-to-point communication where the target is well-identified, and the information is restricted to the communication partners. Broadcast improves structural modularity: no static structural dependency is implied between the sender and the receiver. Another advantage is the consistent perception of a signal throughout the system.

III. State-Transition models

A. Flat model

The simplest state-transition models are flat models, made of states and transitions. When a transition leaving the *current state* is “firable” (i.e., its triggering event has occurred), the transition is taken and the target of the transition becomes the new current state.

Well-known examples are Moore and Mealy machines. With the former, the outputs depend only on the current state whereas, with the latter, the outputs depend on the current state and the current inputs. Synchronous models like SyncCharts [3] support both output associated with states and outputs associated with transitions.

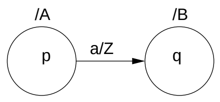

Fig. 1. State/transition.

Even with theses simple flat models, several synchronous interpretations are possible. Consider a transition between state p and q (Fig 1). Let p be the current state. Table I represents the behavior during the change of state initiated by the occurrence of a at instant k.

At instant k, a traditional sequential machine (a mixed Moore/Mealy model) computes its next state q, which will be reached at the next instant. As the current state is still p, the generated output is A (Moore behavior) and Z is also emitted while the transition is taken (Mealy behavior).

TABLE I
Change of state.

instant input	k-1	k a	k+1
Sequential machine	p A	p A, Z	q B
Synchronous model	p	q	q
$\longrightarrow$	A	A, Z, B	B
$\circ\!\!\longrightarrow$	A	Z, B	B
$\longrightarrow\!\!\circ$	A	A, Z	B
$\circ\!\!\longrightarrow\!\!\circ$	A	Z	B

A synchronous model has a different behavior: the change of state takes place at instant k, thus the current state becomes q immediately. In any case, signal Z is emitted while the transition is taken. Other emitted signals depend on the kind of transition (explained in Sec. III-D). Below are possible kinds of transitions, not necessarily available in all synchronous models.

- *weak abortion transition* ($\longrightarrow$): state p is “weakly” exited, i.e., it finishes its current reaction (emitting A). At the same instant q starts a reaction (emitting B).
- *strong abortion transition* ($\circ\!\!\longrightarrow$): the reaction of state p is aborted (A not emitted). q starts a reaction (emitting B).
- *weak abortion with suspension of the target* ($\longrightarrow\!\!\circ$): state p is “weakly” exited and emits A. The reaction of the new current state q is suspended (B not emitted).
- *strong abortion with suspension of the target* ($\circ\!\!\longrightarrow\!\!\circ$): p is strongly exited, so that A is not emitted, neither is B (the new current state q being suspended).

B. Hierarchy and Concurrency

In order to master the complexity of reactive systems, models demand a form of structural decomposition. There are two constructs for coping with structure: refinement (hierarchy) and concurrent composition.

This section addresses the structural point of view, through a particular syntax (namely, SyncCharts [3]). The semantic aspects are the topic of the next section. The SyncCharts representation is based on states, which can be further decomposed according to the previously mentioned constructs.

B.1 Structure of SyncCharts

With a syncChart is associated a unique MacroStar called the *root*. A *MacroStar* is composed of at least one Constellation. A *Constellation* is a set of Stars that defines a state-transition graph. A *Star* has a Body and outgoing arcs of several kinds (strong abort, weak abort, normal termination) called *Beams*. The *Body* is either a BasicStar or a MacroStar (recursive structure). Thus, a syncChart may also be represented as a bipartite tree, which alternates macrostars and constellations. Basic stars are the leaves of the tree.

As it is classical for state-transition systems, Beams have labels composed of three optional fields: a *trigger*, a *guard*, and an *effect*.

The structure is restricted by constraints that define well-formed SyncCharts. For instance, each Constellation is a connected graph and has at least one *initial Star*.

Note that according to the previous construction rules, interlevel transitions, as found in Statecharts [1], are forbidden. This improves modularity and eases composition.

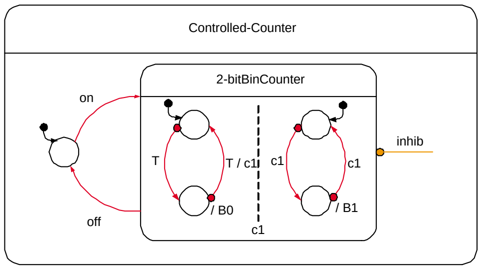

Fig. 2. Exemple of syncChart.

Fig. 2 presents an example of syncChart. The root is the macrostar named `Controlled-Counter`. It is itself composed of only one contellation made of a basic star (the initial state) and another macrostar (`2-bitBinCounter`). The latter comprises two constellations, each one containing basic stars only.

Other features are represented on the figure: A star may be “suspendable”, which is denoted by a suspension arc (as for the star whose body is the `2-bitBinCounter` macrostar); Local signals may be declared within a macrostar (signal `c1`).

Some features are not represented: A star may be *final*, denoted by a double-line border; Optional instantaneous actions (**`entry`** and **`exit`**) are associated with stars; Guards are Boolean expressions between square brackets.

B.2 Status of a SyncChart

The activity status of a syncChart is structurally and recursively defined. A basic star is either *Active* or *Idle*. A constellation is Active if and only if one of its component stars is Active. A macrostar is Active if and only if all its component constellations are Active.

C. Communication and Concurrency

As previously mentioned (Sec. II-A) communication is supported by signals only. Signal scopes are defined at the MacroStar level. A MacroStar or a BasicStar may have input, output, and local signals. The respective sets (`inputs`, `outputs`, `locals`) are disjoint. They must satisfy the *signal consistency rules*: Let m be a MacroStar and m' be a MacroStar or a BasicStar directly contained in m:

$$\mathtt{m}'.\mathtt{inputs} \subseteq \mathtt{m}.\mathtt{inputs} \cup \mathtt{m}.\mathtt{locals}$$
$$\mathtt{m}'.\mathtt{outputs} \subseteq \mathtt{m}.\mathtt{outputs} \cup \mathtt{m}.\mathtt{locals}$$
$$\forall \sigma \in \mathtt{m}'.\mathtt{inputs} \cup \mathtt{m}'.\mathtt{outputs} :$$
$$\text{type of } \sigma \text{ in } m' = \text{type of } \sigma \text{ in } m$$

At the uppermost level of the hierarchy, for a syncChart sc, $\mathtt{sc.root.inputs} \subseteq \mathtt{sc.inputs}$, and $\mathtt{sc.root.outputs} \subseteq \mathtt{sc.outputs}$. Note that the environment has only to know about the syncChart's input and output sets.

Communication implies concurrency. The concurrent structural elements are the constellations. The communication is possible only through the signals that are in the scope of the immediately enclosing MacroStar.

D. Preemptions

Preemption [4] is the ability for a star to forbid the execution of another star, either definitely (abortion) or temporarily (suspension). In synchronous models, preemption is an efficient and usual mechanism for synchronization, not necessarily bound to exceptional behaviors.

Abortion is the standard way of leaving a star. An abortion is qualified as *strong* when the "killed" star is forbidden to execute any reaction when aborted, whereas with *weak* abortion the star is allowed to terminate its current reaction before being exited. The *normal termination* is also a form of weak abortion, due to inner termination of the star's body. `saBeams` (`waBeams`, `ntBeam`, respectively) denotes the possibly empty ordered set of strong abortion beams (weak abortion beams, normal termination beam, respectively). `ntBeam` is a singleton set.

Suspending (or freezing) a star activity is made possible by a suspension beam (member of the optional singleton set named `suspension`). Whenever the trigger of the suspension beam evaluates to true, the star is suspended.
Remark: when several triggers are simultaneously asserted, the behavior is kept fully deterministic thanks to the strict priority associated with each beam of a star (ordered sets of beams).

IV. SEMANTICS

A. Generality

For the sake of simplicity, this paper focuses on *pure models* (i.e., models with neither valued signal, nor variable). The fixed point computation of the next state and of each signal status, which is central to the synchronous semantics, is well-addressed by pure models. Taking account of values is relevant of more classical semantics. Note that, in pure models, transitions are triggered but not guarded.

Usually the semantics of instant-based models is given as the set of possible stimuli/reaction sequence pairs (input sequences / output sequences). This set is given implicitly by an acceptation procedure or by an inductive construction mechanism.

B. Reactivity, Determinism

A system is said to be *reactive* when, given an initial state, for any input sequence there exists at least one output sequence. A system is said to be *deterministic* when there exists at most one output sequence for each input sequence. A syncChart which is both reactive and deterministic is said to be *logically correct*.

Even if syntactically correct, a (well-formed) syncChart, which is not logically correct must be rejected.

C. Communication

Since reactions in SYNCCHARTS may result from instantaneous cooperation of several subsystems, signals, which support communication, are the cornerstone in the SYNCCHARTS semantics.

With each reaction is associated a set of signals E called the *context*. E is partitioned into two blocks: E^+ and E^- (i.e., $E = E^+ \cup E^-$ and $E^+ \cap E^- = \emptyset$). E^+ is the set of the signals certainly present at the current instant, E^- the set of the signals certainly absent. During an instant, a signal must be either present or absent. When computing a reaction, only input signals, which are imposed by the environment, have a definite status; The presence status of all other signals must be determined. Like in the Esterel language[5], we assume that a non-input signal is present if and only if it is emitted during the instant.

D. Boolean Models

Giving a syncChart, the problem is to compute, for any reachable state and for any input, the next state and the presence status of all the signals. Since our model is deterministic, this is equivalent to finding two functions: a next-state function δ and an output function ω.

Insofar as only logically correct solutions are acceptable, a Boolean characterization of the solution is possible. The presence of signals and the activity of states are considered as Booleans.

D.1 Mealy machines

In subsection III-A Moore and Mealy machines were mentioned as flat models. The behavior of a (deterministic) Mealy machine is characterized by two systems of Boolean equations: $X' = \delta(X, I)$ and $O = \omega(X, I)$, where X, I, O, and X' are Boolean vectors.

With this model, the output sequence corresponding to a given input sequence is computed by induction.

D.2 Synchronous Point of View

The same approach can be used to implicitly define "synchronous" behavior. Let ${}^{\backprime}X$ denote the previous state (i.e., the state before the reaction). The "swiftness" of synchronous models might be captured by the Boolean equation systems: $X = \delta({}^{\backprime}X, I)$ and $O = \omega({}^{\backprime}X, I)$.

This is not sufficient to express the possible influence of output signals on the reaction. Taking account of outputs for computing the next state and the actual outputs leads to two different models:

Delayed effect: In this approach, an emitted signal never causes changes during the current instant but can be used during the next reaction. This avoids unstable situations. The equations become $X = \delta({}^{\backprime}X, I, {}^{\backprime}O)$ and $O = \omega({}^{\backprime}X, I, {}^{\backprime}O)$. The arguments passed to the functions that compute X and O are given and fixed at each instant. They are the previous state ${}^{\backprime}X$, the input signals I imposed by the environment for the whole reaction, and the previous output signals ${}^{\backprime}O$. This is the point of view adopted in the STATEMATE semantics of Statecharts [6]

Immediate effect: Another solution is to take account of output signals (and local signals) at the very instant they are emitted. Boolean vectors are linked by equations: $X = \delta(X, I, O, L)$, $L = \lambda(X, I, O, L)$, and $O = \omega(X, I, O, L)$ where L stands for local signal status.

Vectors X, L, and O being in both sides, we have to solve a fixed point problem, given I and ${}^{\backprime}X$. The existence of the fixed point solution is not guaranteed. Even if a (logically correct) solution exists, it may be counter-intuitive because it may violate causality. This point is studied in the subsection on "constructive semantics". Beforehand, we make a digression on the usefulness of immediate reactions.

Examples of immediate reaction: Immediate reactions are useful to prevent from undesirable transient states. A first example is given with macrostar `2-bitBinCounter` in Fig.2. In this model of a synchronous binary counter, the immediate reaction to local signal `c1` makes it possible to move from state 01 to state 10 without passing through state 00.

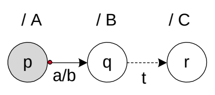

Fig. 3. Immediate reactions.

Immediate reactions may also be explicit. Usually when entering a star, the triggers of the outgoing beams are not checked: only (strictly) future occurrences are considered. In SYNCCHARTS, the immediate consideration of a trigger can be enforced by the $\sharp$ symbol (read "immediate") prefixing a trigger. Immediate reaction combined with preemption results in different behaviors, as illustrated by Fig.3. Table II gives the behavior at the instant when a occurs, according to the type of preemption for star q. Note that, in this example, the immediate strong abortion makes star q to be bypassed.

E. *Constructive Semantics*

E.1 Principle

As previously mentioned (Sec. IV-B) a logically correct solution may lead to counter-intuitive solution (i.e., a solution that defeats causality). To address this problem, instead of dealing with Boolean values, we recurse to a Scott Boolean domain $B_{\perp} = \{\perp, -, +\}$ where $\perp < +$ and $\perp < -$.

TABLE II
IMMEDIATE REACTIONS.

transition t	new state	emitted signals
$\xrightarrow{b}$	$\{q\}$	b, B
$\xrightarrow{\#b}$	$\{r\}$	b, B, C
$\circ\!\xrightarrow{\#b}$	$\{r\}$	b, C

The technique adopted for determining the context E is to propagate facts and build the solution incrementally, if one exists. The computation relies on monotonic functions. The order relation is defined by

$$E \leq E' \iff E^{+} \subseteq E'^{+} \wedge E^{-} \subseteq E'^{-}$$

Let σ^{+} (resp., σ^{-}) denotes the fact that signal σ is present (resp., absent) at the current instant. $\sigma^{\perp}$ means that the presence status of σ is not (yet) known. When computing a reaction all non-input signals are first set to the unknown presence status $\perp$. According to propagated facts, they are then progressively set to either $+$ or $-$, and therefore they enrich the context E.

An obvious consistency rule is that:

$$\forall \sigma : (\sigma \in E^{+} \Leftrightarrow \sigma^{+}) \wedge (\sigma \in E^{-} \Leftrightarrow \sigma^{-})$$

This approach is the one advocated by G. Berry[7], [8] for the Esterel language. He named it the *constructive semantics* in reference to the fact that values are computed by explicit proofs, not by a "trial and error" procedure.

Signals are combined in so-called *signal expressions* used, for instance, in triggers. A signal expression is an expression similar to a Boolean expression, made of signals, operators ('not ', 'or ', 'and '), and parentheses.

Given a context E, a signal expression evaluates in $B_{\perp}$: For $\Phi ::= \sigma$ (for a signal σ) $|$ not ϕ $|$ ϕ or ψ $|$ ϕ and ψ. A signal expression Φ is evaluated as follows:

$$\begin{array}{rcl}
\text{eval}(\sigma, E) & = & + \text{ if } \sigma \in \mathrm{E}^{+} \\
 & & - \text{ if } \sigma \in \mathrm{E}^{-} \\
 & & \perp, \text{ otherwise} \\
\text{eval}(\text{not } \phi, \mathsf{E}) & = & \text{not eval } (\phi, \mathsf{E}) \\
\text{eval}(\phi \text{ or } \psi, \mathsf{E}) & = & \text{eval } (\phi, E) \text{ or eval } (\psi, \mathsf{E})
\end{array}$$

where operators not and or are defined in the Table below, and ϕ and $\psi =$ not $\big((\text{not } \phi) \text{ or } (\text{not } \psi)\big)$.

	not
$\perp$	$\perp$
-	+
+	-

or	$\perp$	-	+
$\perp$	$\perp$	$\perp$	+
-	$\perp$	-	+
+	+	+	+

E.2 Computation of a Reaction

The computation of a reaction is a complex process. In this paper, we choose to explain this computation by a dynamic process: a syncChart is seen as a collection of *reactive objects*[9]. A logical thread is associated with each constellation, and all the threads cooperate to find the fixed point solution. We use the adjective "logical" because the threads are conceptual: there are used to explain the semantics.

The threads execute concurrently. A thread suspends its execution whenever it needs a signal whose presence status is unknown. Other (still running) threads may emit the awaited signal, so that the suspended thread can resume. The process stops when all the threads have terminated their reaction or are suspended. What is done in the latter case depends on the chosen semantics. Some semantic variants will be described below. Beforehand, we explain how logical threads are managed.

Recursive Execution: At the beginning of an instant, the `react()` method of the syncChart is called. This method makes recursive calls to the `react()` method of its components, till stability or deadlock is reached. `react()` methods return either `TERM`, or `FSTOP`, or `STOP`. These values are ordered: `TERM` $<$ `FSTOP` $<$ `STOP`.

```
SyncChart::react()
   read inputs
   for all s in outputs do
     s.reset()
   done
   root.react()
```

The `reset()` method, when applied to a signal, sets its presence status to $\perp$.

```
MacroStar::react()
   for all s in locals do
     s.reset()
   done
   parallel for all c in constels do
     r[k] = c.react()
   done
   return Max(r)
```

The reaction of a macrostar may fork several logical threads (one for each child constellation).

```
Constellation::react()
   if curState==null then
     curState = initialStar
   endif
   while (r=curStar.react())==TERM do
     curStar = nextStar
   done
   return r
```

The `react()` method of a constellation calls the `react()` method of its current active star. Note that several stars can be passed through, in a strictly sequential order, during the one instant.

For the `react()` method of a star, see Fig. 4. Note that the behavior is different at the first instant and at the following instants. The `kill()` method makes recursive depth-first kills: stars enter their `IDLE` state, and constellations forget their current star.

The heart of the computation is the `testP` (test presence) function that evaluates a signal expression. This function is executed by a thread that waits until the signal expression evaluates to either + or -.

```
boolean testP(expr,E)
   wait (r=eval(expr,E)) != ⊥
   return (r == +)
```

This function is used in trigger evaluations (capsules with gray background in Fig.4). Given an ordered set of beams `B` and a context `E`, `testBeams` returns the first firable beam (i.e., the first beam whose trigger evaluates to `true`), if any, or `null`, otherwise.

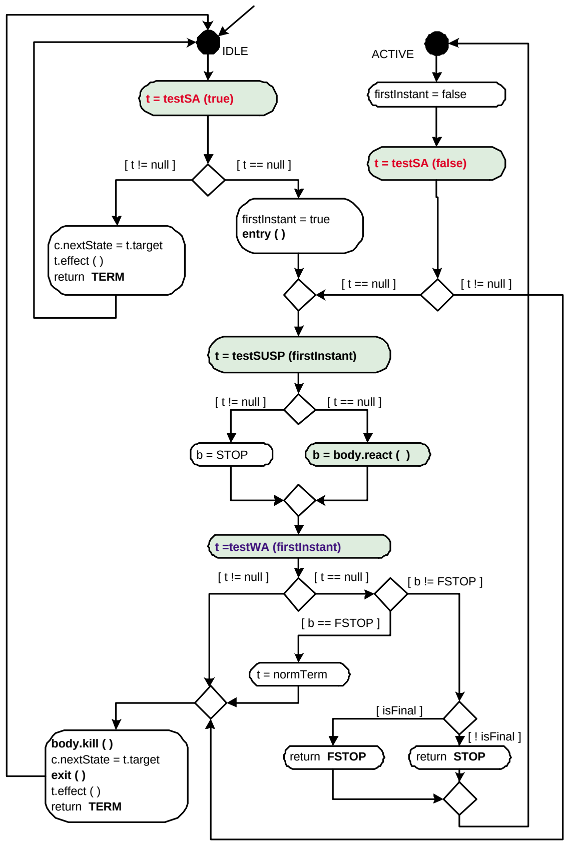

Fig. 4. Star Reaction.

```
Beam testBeams(B,E)
   for (b:Beam=B'first to B'last) do
     if testP(b.trigger,E) then
       return b
     end if
   end for
   return null
```

Function `testSA()` (`testWA()`, `testSUSP()`, `testNT()`, respectively) used in `Star :: react()` is a special case of `testBeams`, where parameter `B` is the set of the strong abortion (weak abortion, suspension, normal termination, respectively) beams.

There are different ways to resume execution of suspended threads when executing `testP()`. This leads to different semantics described in what follows.

E.3 Propagating Positive Facts Only

In this approach, a thread checking a trigger is allowed to proceed only if `testP` method returns `true` (a positive fact). Thus, only positive facts are propagated during the instant, making it possible to chain several instantaneous actions. When all the threads are suspended, signal emission is no longer possible and there is no way to resume thread executions. All signals still with unknown presence status are set to absent. The end of the reaction is enforced. Only at the next instant will all the pending threads resume and execute the continuation associated with the outcome of `testP()`.

This interpretation is easy to understand and compute. Since only positive facts (certainties) are propagated, there is no risk of causality violation. Unfortunately this semantics severely restricts the use of strong abortion, suspension, and priority.

E.4 Use of a Potential Function

The idea is now to propagate negative facts as well as positive ones. A negative fact may be the certainty that a given signal cannot be emitted during a reaction. To know that a signal shall not be emitted seems to be relevant of clairvoyance. In fact, we construct a monotonic decreasing set of potentially emitted signals, called the *potential*. Any signal not in this set will certainly not be emitted, provided that this set is *correct* (i.e., all signals possibly emitted *are in* the potential). When all the threads are terminated or suspended, the context is enriched with negative facts, so that some threads possibly resume. When the fixed point is reached, if any signal is still in the unknown presence status, then the syncChart is rejected as non-constructive.

The potential is a superset of the signals effectively emitted during the reaction. Overestimated potentials are easier to compute but they slow the convergence down and may restrict the class of accepted (i.e., semantically correct) syncCharts. Various potential functions for the Esterel language have been analyzed by F. Boussinot[10].

SYNCCHARTS used in the Esterel-Studio environment [11] adhere to a strong semantics, fully compatible with the one adopted for the Esterel language; There is no restriction on the use of the various forms of preemption. A drawback of this expressiveness is that a novice user may design a syncChart with causality cycles not easy to understand and correct.

E.5 Other Possible Relaxations

If we adopt a SYNCCHARTS semantics like the one presented in Sec.IV-E.3, we have no causality problem but most of the expressiveness of the model is lost. On the other hand, the behavior of standard SYNCCHARTS is not always easy to understand and debug. There is room between these two solutions for "simpler" SYNCCHARTS. The main issue when simplifying a model is to know what is gained and what is lost. Possible criteria are: *legibility* (number and complexity of primitive graphical elements), *understandability* (simplicity of underlying concepts and composition rules), *efficiency* (cost for computing reactions), and *expressiveness* of the model for some application domain.

Fig.4 may help in choosing the simplifications. For instance, if you decide to remove strong abortion and use only the weak form, all the parts related to `testSA()` are erased in the graph. Forbidding explicit immediate triggers is another possible variant. In this case, the upper left loop in the graph can be deleted since not trigger can be satisfied when entering a star. Tailoring a potential function is another way to improve efficiency.

V. CONCLUSION

This paper demonstrates how to describe complex reactive systems in a modular way owing to the notion of instant combined with states. Graphical notations expressively support the corresponding model. The presentation aspect is highly useful. It is not sufficient, though, to cope with the stringent requirements of the application domains of reactive systems. Mathematical foundation is needed. Fortunately, the instant-based model lends itself to simple semantics, yet realistic and expressive. The so-called *synchronous languages* have pioneered this approach and introduced the concept of pure synchrony as their semantic basis. Models like SYNCCHARTS stricly follow this line.

However, the pure synchrony hypothesis sometimes makes the task of the user difficult; moreover it not necessary, nor suitable for all applications. We have unveiled several tracks to relax this hypothesis without jeopardizing the semantic soundness. This is delicate work since it is not always easy to characterize the class of correct models that result from weakening pure synchrony. For instance, several attempts have been made to improve the efficiency of the Esterel compiler: Edwards[12] only considers a subset of acceptable programs; the SAXO-RT[13] compiler generates even more efficient C code but the class of accepted programs is not clearly characterized.

Much work has still to be done in order to find the relaxing assumptions suitable for coping with particular kinds of application. Tools are required to assess the impact of an assumption. Of course, they must comply with the current software practice. In particular, we have undertaken the integration of such tools, the graphical notation, and their semantic variants into the UML[14].

REFERENCES

[1] D. Harel and A. Pnueli, "On the development of reactive systems in logic and models of concurrent systems," *NATO ASI Series, K.R Apt Ed., Springer-Verlag*, vol. 13, pp. 477–498, 1985.

[2] N. Halbwachs, *Synchronous Programming of Reactive Systems*. Amsterdam: Kluwer Academic Publishers, 1993.

[3] C. André, "Representation and analysis of reactive behaviors: A synchronous approach," in *Computational Engineering in Systems Applications (CESA)*, (Lille (F)), pp. 19–29, IEEE-SMC, July 1996.

[4] G. Berry, "Preemption in concurrent systems," *Proc FSTTCS, Lecture notes in Computer Science*, vol. 761, pp. 72–93, 1992.

[5] F. Boussinot and R. De Simone, "The ESTEREL language," *Proceeding of the IEEE*, vol. 79, pp. 1293–1304, September 1991.

[6] D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," *ACM Trans. Soft. Eng. Method.*, vol. 5, pp. 477–498, October 1996.

[7] G. Berry, *The Constructive Semantics of pure Esterel*. Sophia Antipolis (F): not yet published, available on the web, `www.inria.fr/equipes/meije/esterel`, 1996.

[8] G. Berry, "The foundations of Esterel," in *Proof, Language and Interaction: Essays in Honour of Robin Milner* (C. S. G. Plotkin and M. Tofte, eds.), MIT Press, 2000.

[9] F. Boussinot, G. Doumenc, and J.-B. Stefani, "Reactive objects," *Ann. Telecommunication*, vol. 51, no. 9–10, pp. 459–473, 1996.

[10] F. Boussinot, "Sugarcubes implementation of causality," Tech. Rep. 3487, INRIA, September 1998.

[11] Esterel Technologies, Guyancourt (F), *Esterel Studio, V3.1*, September 2001. Reference Manual.

[12] S. A. Edwards, "An ESTEREL compiler for large control-dominated systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 2, pp. 169–183, 2002.

[13] E. Closse, M. Poize, J. Pulou, P. Venier, and D. Weil, "SAXO-RT: Efficient compilation of ESTEREL for real-time embedded systems," (Bucarest (Romania)), IWACT, 2001.

[14] C. André, M.-A. Peraldi-Frati, and J.-P. Rigault, "Integrating the Synchronous Paradigm into UML: Application to Control-Dominated Systems," in *UML* $\ll$ 2002 $\gg$, (Dresden (D)), Springer-Verlag, October 2002.