Multi-Viewpoint State Machines for Rich Component Models[1]

Albert Benveniste and Benoît Caillaud
IRISA / INRIA, Rennes, France

Roberto Passerone
PARADES GEIE, Rome, Italy
University of Trento, Trento, Italy

November 18, 2008

[1]This research has been developed in the framework of the European IP-SPEEDS project number 033471.

Contents

Chapter 1

Multi-Viewpoint State Machines for Rich Component Models[1]

1.1 Introduction and requirements

This chapter presents the modeling effort that sustains the work related to the IP-SPEEDS Heterogeneous Rich Component (HRC) metamodel, its associated multiple viewpoint contract formalism and the underlying mathematical model of machines supporting such contracts. We put the emphasis on combining different viewpoints and providing a simple and elegant notion of parallel composition.

The motivations behind this work are the drastic organizational changes that several industrial sectors involving complex embedded systems have experienced — aerospace and automotive being typical examples. Initially organized around large, vertically integrated companies supporting most of the design in house, these sectors were restructured in the 80's because of the emergence of sizeable competitive suppliers. Original equipment manufacturers (OEM) performed system design and integration by importing entire subsystems from suppliers. This, however, shifted a significant portion of the value to the suppliers, and eventually contributed to late errors that caused delays and excessive additional cost during the system integration phase. In the past decade, these industrial sectors went through a profound reorganization in an attempt by OEMs to recover value from the supply chain, by focusing on those parts of the design at the core of their competitive advantage. The rest of the system was instead centered around standard platforms that could be developed and shared by otherwise competitors. Examples of this trend are AUTOSAR in the automotive

[1]This research has been developed in the framework of the European IP-SPEEDS project number 033471.

industry [10], and Integrated Modular Avionics (IMA) in aerospace [7]. This new organization requires extensive virtual prototyping and design space exploration, where component or subsystem specification and integration occur at different phases of the design, including at the early ones [19].

Component based development has emerged as the technology of choice to address the challenges that result from this paradigm shift. Our objective is to develop a component-based model that is tailored to the specific requirement of system development with a highly distributed OEM/supplier chain. This raises the novel issue of dividing and distributing responsibilities between the different actors of the OEM/supplier chain. The OEM wants to define and know precisely what a given supplier is responsible for. Since components or subsystems interact, this implies that the responsibility for each entity in the area of interaction must be precisely assigned to a given supplier, and must remain unaffected by others. Thus, each supplier is assigned a design task in the form of a goal, which we call *guarantee* or *promise*, that involves only entities for which the supplier is responsible. Other entities entering the sub-system for design are not under the responsibility of this supplier. Nonetheless, they may be subject to constraints assigned to the other suppliers, that can therefore be offered to this supplier as *assumptions*. Assumptions are under the responsibility of other actors of the OEM/supplier chain but can be used by this supplier to simplify the task of achieving its own promises. This mechanism of assumptions and promises is structured into *contracts* [9], which form the essence of distributed system development involving complex OEM/supplier chains.

In addition to contracts, supporting an effective concurrent system development requires the correct modeling of both interfaces and open systems, as well as the ability to talk about partial designs and the use of abstraction mechanisms. This is especially true in the context of safety critical embedded systems. In this case, the need for high quality, zero-defect software calls for techniques in which component specification and integration is supported by clean mathematics that encompass both static and *dynamic* semantics — this means that the behavior of components and their composition, and not just their port and type interface, must be mathematically defined. Furthermore, system design includes various aspects — functional, timeliness, safety and fault tolerance, etc. — involving different teams with different skills using heterogeneous techniques and tools. We call each of these different aspects a *viewpoint* of the component or of the system. Our technology of contracts is based on a mathematical foundation consisting of a model of system that is rich enough to support the different viewpoints of system design, and at the same time clean and simple enough to allow for the development of mathematically sound techniques. We build on these foundations to construct a more descriptive state-based model, called the Heterogeneous Rich Component (HRC) model, that describes the relationships between the parts of a component in an executable fashion. It is the objective of this chapter to present this higher level model. Nonetheless, we also provide a quick overview of the contract model it is intended to support — readers interested in details regarding this contract framework are referred to [5, 6].

Our notion of contract builds on similar formalisms developed in related fields. For example, a contract-based specification was applied by Meyer in the context of the programming language Eiffel [17]. In his work, Meyer uses *preconditions* and *postconditions* as state predicates for the methods of a class, and *invariants* for the class itself. Similar ideas were already present in seminal work by Dijkstra [12] and Lamport [16] on *weakest preconditions* and *predicate transformers* for sequential and concurrent programs, and in more recent work by Back and von Wright, who introduce contracts [4] in the *refinement calculus* [3]. In this formalism, processes are described with guarded commands operating on shared variables. This formalism is best suited to reason about discrete, untimed process behavior.

More recently, De Alfaro and Henzinger have proposed Interface Automata as a way of expressing constraints on the environment in the case of synchronous models [11]. The authors have also extended the approach to other kinds of behaviors, including resources and asynchronous behaviors [8, 15]. Our contribution here consists in developing a particular formalism for hybrid continuous-time and discrete state machines where composition is naturally expressed as intersection. We show how to translate our model to the more traditional hybrid automata model [14]. In addition, we identify specialized categories of automata for the cases that do not need the full generality of the model, and introduce probabilities as a way of representing failures.

The chapter is structured as follows. We will first review the concepts of component and contract from a semantic point of view in Section 1.2. We then describe the Extended State Machine model in Section 1.3 and compare it to a more traditional hybrid model in Section 1.4. The syntax and the expressive power used for expressions in the transitions of the state-based model is reviewed in Section 1.5, followed, in Section 1.6, by the specialization of the model into different categories to support alternative viewpoints. Several examples complement the formalism throughout the chapter.

1.2 Components and Contracts

Our model is based on the concept of *component*. A component is a hierarchical entity that represents a unit of design. Components are connected together to form a system by sharing and agreeing on the values of certain ports and variables. A component may include both *implementations* and *contracts*. An implementation M is an instantiation of a component and consists of a set P of ports and variables (in the following, for simplicity, we will refer only to ports) and of a set of behaviors, or runs, also denoted by M, which assign a history of "values" to ports. Because implementations and contracts may refer to different viewpoints, as we shall see, we refer to the components in our model as *heterogeneous rich components* (HRC).

We build the notion of a contract for a component as a pair of assertions, which express its assumptions and promises. An assertion E is a property that may or may not be satisfied by a behavior. Thus, assertions can again

be modeled as a set of behaviors over ports, precisely as the set of behaviors that satisfy it. An implementation M satisfies an assertion E whenever they are defined over the same set of ports and all the behaviors of M satisfy the assertion, i.e., when $M \subseteq E$.

A contract is an assertion on the behaviors of a component (the promise) subject to certain assumptions. We therefore represent a contract C as a pair (A, G), where A corresponds to the assumption, and G to the promise. An implementation of a component satisfies a contract whenever it satisfies its promise, subject to the assumption. Formally, $M \cap A \subseteq G$, where M and C have the same ports. We write $M \models C$ when M satisfies a contract C. There exists a unique maximal (by behavior containment) implementation satisfying a contract C, namely $M_C = G \cup \neg A$. One can interpret M_C as the implication $A \Rightarrow G$. Clearly, $M \models (A, G)$ if and only if $M \models (A, M_C)$, if and only if $M \subseteq M_C$. Because of this property, we can restrict our attention to contracts of the form $C = (A, M_C)$, which we say are in *canonical form*, without losing expressiveness. The operation of computing the canonical form, i.e., replacing G with $G \cup \neg A$, is well defined, since the maximal implementation is unique, and it is idempotent. Working with canonical forms simplifies the definition of our operators and relations, and provides a unique representation for equivalent contracts.

The combination of contracts associated to different components can be obtained through the operation of parallel composition. If $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ are contracts (possibly over different sets of ports), the composite must satisfy the guarantees of both, implying an operation of intersection. The situation is more subtle for assumptions. Suppose first that the two contracts have disjoint sets of ports. Intuitively, the assumptions of the composite should be simply the conjunction of the assumptions of each contract, since the environment should satisfy all the assumptions. In general, however, part of the assumptions A_1 will be already satisfied by composing C_1 with C_2, acting as a partial environment for C_1. Therefore, G_2 can contribute to relaxing the assumptions A_1. And vice-versa. The assumption and the promise of the composite contract $C = (A, G)$ can therefore be computed as follows:

$$A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2), \tag{1.1}$$

$$G = G_1 \cap G_2, \tag{1.2}$$

which is consistent with similar definitions in other contexts [11, 13, 18]. C_1 and C_2 may have different ports. In that case, we must extend the behaviors to a common set of ports before applying (1.1) and (1.2). This can be achieved by an operation of inverse projection. Projection, or elimination, in contracts requires handling assumptions and promises differently, in order to preserve their semantics. For a contract $C = (A, G)$ and a port p, the *elimination of p in C* is given by

$$[C]_p = (\forall p\, A,\ \exists p\, G) \tag{1.3}$$

where A and G are seen as predicates. Elimination trivially extends to finite sets of ports, denoted by $[C]_P$, where P is the considered set of ports. For

inverse elimination in parallel composition, the set of ports P to be considered is the union of the ports P_1 and P_2 of the individual contracts.

Parallel composition can be used to construct complex contracts out of simpler ones, and to combine contracts of different components. Despite having to be satisfied simultaneously, however, multiple viewpoints *associated to the same component* do not generally compose by parallel composition. We would like, instead, to compute the conjunction $\sqcap$ of the contracts, so that if $M \models C_f \sqcap C_t$, then $M \models C_f$ and $M \models C_t$. This can best be achieved by first defining a partial order on contracts, which formalizes a notion of substitutability, or refinement. We say that $C = (A, G)$ *dominates* $C' = (A', G')$, written $C \preceq C'$, if and only if $A \supseteq A'$ and $G \subseteq G'$, and the contracts have the same ports. Dominance amounts to relaxing assumptions and reinforcing promises, therefore strengthening the contract. Clearly, if $M \models C$ and $C \preceq C'$, then $M \models C'$.

Given the ordering of contracts, we can compute greatest lower bounds and least upper bounds, which correspond to taking the conjunction and disjunction of contracts, respectively. For contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ (in canonical form), we have

$$C_1 \sqcap C_2 = (A_1 \cup A_2, G_1 \cap G_2), \tag{1.4}$$

$$C_1 \sqcup C_2 = (A_1 \cap A_2, G_1 \cup G_2). \tag{1.5}$$

The resulting contracts are in canonical form. Conjunction of contracts amounts to taking the union of the assumptions, as required, and can therefore be used to compute the overall contract for a component starting from the contracts related to multiple viewpoints. The following example illustrates the need for two different composition operators.

Example 1 (viewpoint synchronization) We discuss here an example of viewpoint synchronization. Assume two contracts $C_i, i = 1, 2$ modeling two different viewpoints attached to a same rich component $\mathbf{C}$. For example, let $C_1 = (A_1, G_1)$ be a viewpoint in the functional category and $C_2 = (A_2, G_2)$ be a viewpoint of the timed category.

Assumption A_1 specifies allowed data pattern for the environment, whereas A_2 sets timing requirements for it. Since contracts are in canonical forms, the promise G_1 itself says that, if the environment offers the due data pattern, then a certain behavioural property can be guaranteed. Similarly, G_2 says that, if the environment meets the timing requirements, then outputs will be scheduled as wanted and deadlines will be met. Thus, both $G_i, i = 1, 2$ are implications.

The greatest lower bound $C_1 \sqcap C_2$ can accept environments that satisfy either the functional assumptions, or the timing assumptions, or both. The promise of $C_1 \sqcap C_2$ is the conjunction of the two implications: if the environment offers the due data pattern, then a certain behavioural property can be guaranteed, or, if the environment meets the timing requirements, then outputs will be scheduled as wanted and deadlines will be met, or, if both the environment offers the due data pattern and the environment meets the timing requirements, then both a

certain behavioural property can be guaranteed and outputs will be scheduled as wanted and deadlines will be met.

To have a closer look at the problem, assume first that the two viewpoints are orthogonal or unrelated, meaning that the first viewpoint, which belongs to the functional category, does not depend on dates, while the second viewpoint does not depend on the functional behaviour (e.g., we have a dataflow network of computations that is fixed regardless of any value at any port). Let these two respective viewpoints state as follows:

- if the environment alternates the values $\mathtt{T}, \mathtt{F}, \mathtt{T}, \ldots$ on port b, then the value carried by port x of component $\mathbf{C}$ never exceeds 5.
- if the environment provides at least one data per second on port b, then component $\mathbf{C}$ can issue at least one data every two seconds on port x.

These two viewpoints relate to the same rich component. Still, having the two contracts $(A_i, G_i), i = \mathit{funct}, \mathit{timed}$ for $\mathbf{C}$ should mean that: if the environment satisfies the functional assumption, then $\mathbf{C}$ satisfies the functional guarantees. Also, if the environment satisfies the timing assumption, then $\mathbf{C}$ satisfies the timing guarantees. Figure 1.1 illustrates the greatest lower bound of the viewpoints belonging to two different categories, and compares it with their parallel composition, introduced in Section 1.2. For this case, the correct definition for viewpoint synchronization is the greatest lower bound.

The four diagrams on the top are the truth tables of the functional category C_f and its assumption A_f and promise G_f, and similarly for the timed category C_t. Note that these two contracts are in canonical form. In the middle, we show the same contracts lifted to the same set of variables b, d_b, x, d_x, combining function and timing. On the bottom, the two tables on the left-hand side are the truth tables of the greatest lower bound $C_f \sqcap C_t$. For comparison, we show on the right-hand side the truth tables of the parallel composition $C_1 \parallel C_2$, revealing that the assumption is too restrictive and not the one expected.

So far we discussed the case of non interacting viewpoints. But in general, viewpoints may interact as explained in the following variation of the same example. Assume that the viewpoints (the first one belonging to the functional category, while the other one belongs to the timed category) interact as follows:

- if the environment alternate the values $\mathtt{T}, \mathtt{F}, \mathtt{T}, \ldots$ on port b, then the value carried by port x of $\mathbf{C}$ never exceeds 5; if x outputs the value 0, then an exception is raised and a handling task T is executed;
- if the environment provides at least one data per second on port b, then $\mathbf{C}$ can issue at least one data every two seconds on port x; when executed, task T takes 5 seconds for its execution.

For this case, the activation port α_T of task T is an output port of the functional view, and an input port of the timed view. This activation port is boolean; it is output every time the component is activated and is true when an exception

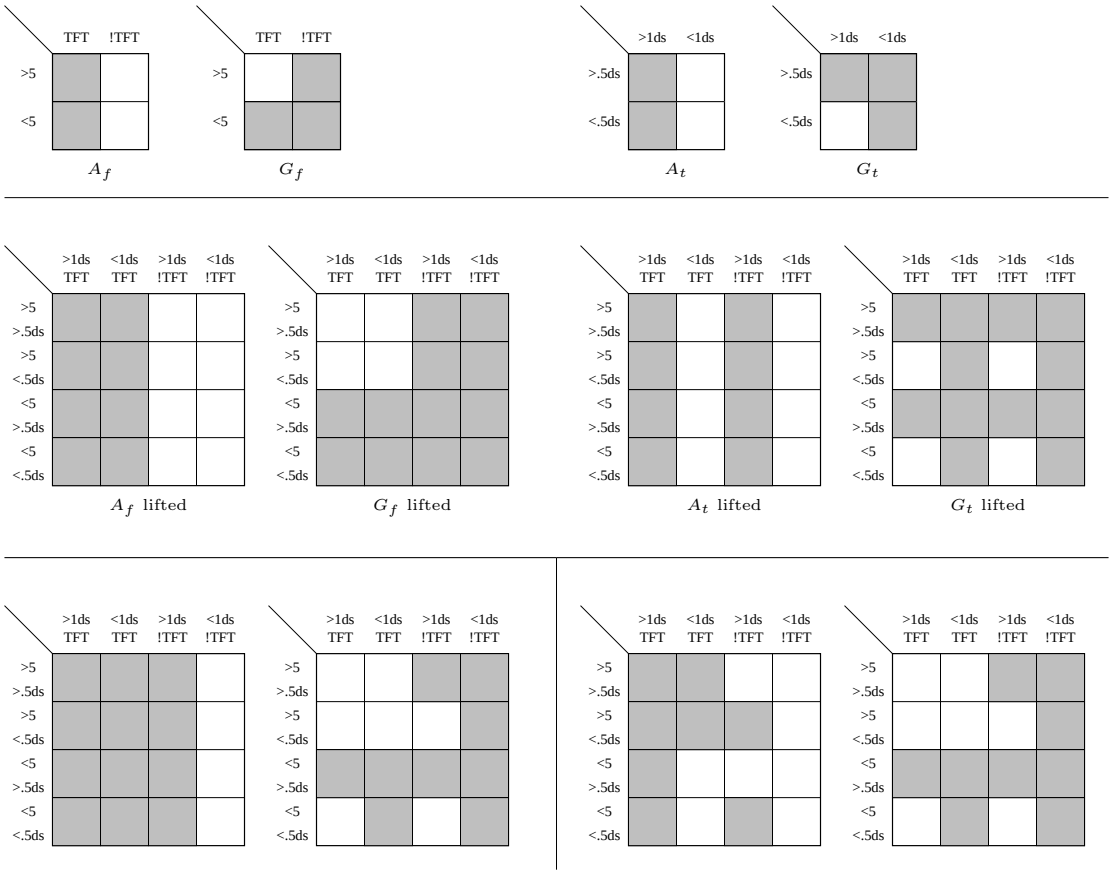

Figure 1.1: Truth tables for the synchronization of categories.

is raised. Then, the timed viewpoint will involve α_T and d_{α_T} as inputs, and will output the date d_T of completion of the task T according to the following formula: $d_T = (d_{\alpha_T} + 5)$ when $(\alpha_T = \texttt{T})$. Note that d_{α_T} has no meaning when $\alpha_T = \texttt{F}$.

Here we had an example of connecting an output of a functional viewpoint to an input of a timed viewpoint. Note that the converse can also occur. Figure 1.2 illustrates the possible interaction architectures for a synchronization viewpoint. ◇

Discussion. So far we have defined contracts and implementations in terms of abstract assertions, i.e., sets of runs. In the next sections, we describe in more precise terms the mathematical nature of these abstract assertions.

To provide intuition for our design choices, we start by comparing two alternative views of system runs, illustrated in Figure 1.3. In the classical approach, shown on the left in the figure, transitions take no time; time and continuous dy-

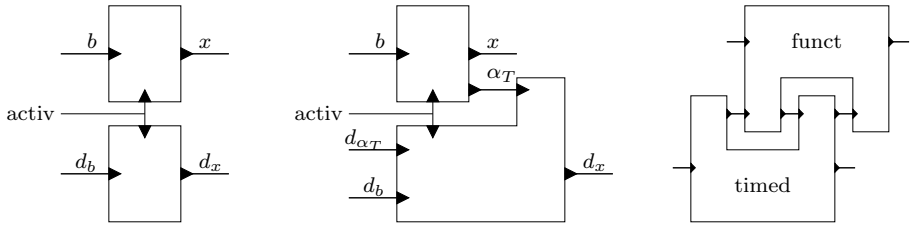

Figure 1.2: Illustrating the synchronization of viewpoints.

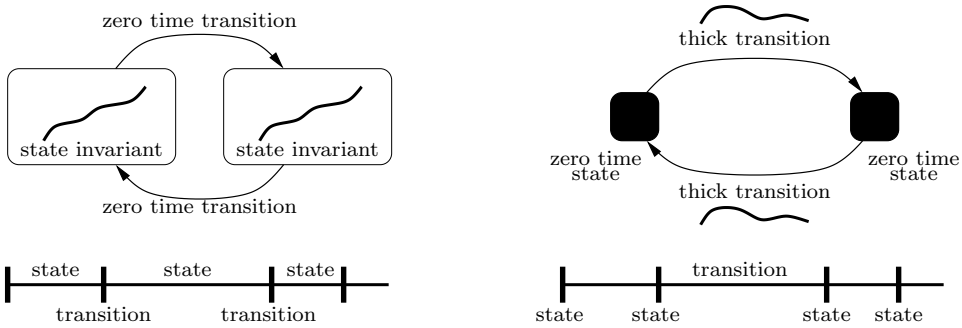

Figure 1.3: Runs. Left: classical approach. Right: alternative approach. State invariants on the left, or thick transitions on the right, involve the progress of time and continuous dynamics such as differential equations.

namics progress within states; they are specified by state invariants and guarded. The alternative approach is dual: states are snapshot valuations of variables and take no time; time and continuous dynamics progress within "thick" transitions that are guarded.

The two approaches have advantages and disadvantages. The classical approach is preferred for abstractions based on regions, which are valid for certain classes of models. The alternative approach makes it much easier to deal with composition and is able to capture open systems, as we shall see. Clearly, the two approaches are dual and can be exchanged without harm.

We shall develop the two approaches and relate them throughout this chapter.

1.3 Extended State Machines

Extended State Machines (ESM) follow the second approach illustrated in Figure 1.3. They are our preferred model, because of the simplicity of its associated parallel composition.

1.3.1 Variables and Ports, Events and Interactions, Continuous Dynamics

Interaction between viewpoints and components is achieved by synchronizing events and variables involved in both discrete and continuous dynamics. Synchronization events take place on ports. Dynamic creation or deletion of ports or variables is not supported by the model.

Values are taken in a unique domain D that encompasses all usual data types (booleans, enumerated types, integers, real numbers, etc.). We shall distinguish a subdomain $D_c \subset D$ in which variables involved in continuous evolutions take their values; D_c collects all needed Euclidean spaces to deal with differential equations or inclusions. Other type consistency issues are dealt with in the static semantics definition of HRC and are disregarded in the sequel.

We are given a finite set $\mathcal{V}$ of *variables*; the set of variables is partitioned into two subsets $\mathcal{V} = \mathcal{V}_d \uplus \mathcal{V}_c$: the variables belonging to $\mathcal{V}_d$ are used exclusively in the discrete evolutions, and those belonging to $\mathcal{V}_c$ can be used in both continuous and discrete evolutions. *States* correspond to the assignment of a value to each variable: $s : \mathcal{V} \to D$.

A finite set of *ports* $\mathcal{P}$ is then considered. *Events* correspond to the assignment of a value to a port; therefore an event is a pair $(p, d) \in \mathcal{P} \times D$. *Interactions,* also called *labels* in the sequel, are sets of events. The only restriction is that a given port may yield at most one event in an interaction. Hence interactions are partial mappings $\lambda : \mathcal{P} \rightharpoonup D$. The set of all interactions is denoted by Λ $(= \mathcal{P} \rightharpoonup D)$. The empty interaction $\varepsilon_{\mathcal{P}}$ over ports $\mathcal{P}$ is the unique mapping $\varepsilon_{\mathcal{P}} : \mathcal{P} \rightharpoonup D$ that is undefined for any $p \in \mathcal{P}$.

Regarding continuous dynamics we restrict ourselves to the case where a unique global physical time is available, denoted generically by the symbols t or τ and called the *universal time*. Other time bases can be used, but need to be related to this universal time as part of the assertion specification. Investigating the consequences of relaxing this restriction is part of our future work.

Similarly, for $V_c \subseteq \mathcal{V}_c$, the *domain of continuous evolutions on V_c,* denoted by $\mathcal{C}(V_c)$, is the set of all functions

$$\mathcal{C}(V_c) \quad =_{\text{def}} \quad \{\, \varphi \mid \varphi : \mathbb{R}_+ \rightharpoonup V_c \to D_c \,\} \tag{1.6}$$

such that (we write $\varphi(t, v)$ instead of $\varphi(t)(v)$):

1. $\text{dom}(\varphi) = [0, t|$ for some $t > 0$, where symbol $|$ denotes either] or); call t the *duration* of φ and denote it generically by t_φ.

2. For every $v \in V_c$, $\tau \to \varphi(\tau, v)$ is smooth enough (typically at least differentiable on $(0, t)$) and possesses a left limit $\mathit{Exit}(\varphi) \in D^{V_c}$ defined by

$$\mathit{Exit}(\varphi, v) \quad =_{\text{def}} \quad \lim_{\tau \nearrow t} \varphi(\tau, v) \tag{1.7}$$

Each $\varphi \in \mathcal{C}(V_c)$ can be decomposed, for all $t \in (0, t_\varphi)$, as the concatenation

$$\begin{array}{rcll} \varphi & = & \varphi_1 \bullet \varphi_2, \text{ where} & \\ \varphi_1(\tau) & = & \varphi(\tau) \text{ for } 0 \leq \tau < t, & \mathrm{dom}(\varphi_1) = [0, t) \\ \varphi_2(\tau) & = & \varphi(t+\tau) \text{ for } 0 \leq \tau < t_\varphi - t, & \mathrm{dom}(\varphi_2) = [0, t_\varphi - t) \end{array} \tag{1.8}$$

We denote these two evolutions by $\varphi_{<t}$ and $\varphi_{\geq t}$, respectively. We thus have the decomposition

$$\varphi \quad = \quad \varphi_{<t} \bullet \varphi_{\geq t} \tag{1.9}$$

1.3.2 ESM Definition

Having defined variables, ports, labels and interactions, it is possible to introduce extended state machines, as a syntactic means of defining assertions in HRC components.

Definition 1.3.1 (ESM) *An* extended state machine *is a tuple with the following components:*

$$\begin{array}{rcl} E & = & (V, P, \rho, \delta, I, F)\,, \textit{ where:} \\ P \subseteq \mathcal{P} & , & V = V_d \uplus V_c, V_d \subseteq \mathcal{V}_d, V_c \subseteq \mathcal{V}_c \\ S & =_{\mathrm{def}} & D^V \textit{ is the set of } \text{states}, \textit{ projecting to} \\ S_d & =_{\mathrm{def}} & D^{V_d} \textit{ the set of } \text{discrete states}, \textit{ and} \\ S_c & =_{\mathrm{def}} & D^{V_c} \textit{ the set of } \text{continuous states}, \\ \rho & \subseteq & S \times \Lambda \times S, \textit{ where } \Lambda =_{\mathrm{def}} (P \rightharpoonup D), \textit{ is the } \text{discrete transition relation} \\ \delta & \subseteq & S \times \mathcal{C}(V_c) \times S \textit{ is the } \text{continuous transition relation} \\ I & \subseteq & S \textit{ is the set of } \text{initial states}. \\ \textit{and } F & \subseteq & S \textit{ is the set of } \text{final states}. \end{array}$$

where we require that δ does not modify discrete states:

$$\forall (s, \varphi, s') \in \delta, \forall v \in V_d \quad \Rightarrow \quad s'(v) = s(v). \tag{1.10}$$

For convenience, we shall denote the disjoint union of sets of ports and variables by $W =_{\mathrm{def}} P \uplus V$.

Runs. The runs recognized by an ESM are arbitrary finite interleavings of discrete and continuous evolutions, separated by snapshot states:

$$\sigma \quad =_{\mathrm{def}} \quad s_0, w_1, s_1, w_2, s_2, \ldots, s_{k-1}, w_k, s_k, \ldots \tag{1.11}$$

where

$$\begin{array}{rcl} s_0 & \in & I \\ \forall k > 0 & : & \left\{ \begin{array}{rcl} \text{either } w_k & = & (s_{k-1}, \lambda_k, s_k) \in \rho \\ \text{or } w_k & = & (s_{k-1}, \varphi_k, s_k) \in \delta \end{array} \right. \end{array}$$

Infinite runs are captured by considering their finite approximations. *Accepted runs* are finite runs ending in F. To capture nonterminating computations, just take $F = S$. In run σ, time progresses as follows: discrete transitions take no time and continuous evolutions are concatenated. Formally:

- State s_k is reached at time $\sum_{i=1}^{k} t_{w_i}$, where t_w denotes the duration of w; by convention, t_w is equal to t_φ (the duration of φ) if $w = (s, \varphi, s')$, and is equal to zero if $w = (s, \lambda, s')$.

- At time t, the number of transitions completed is $\max\{k \mid \sum_{i=1}^{k} t_{w_i} \leq t\}$.

Projection. For $W = P \oplus V$ a set of ports and variables, ρ a discrete transition relation defined over W, δ a continuous transition relation defined over W, and $W' \subseteq W$, let $\mathbf{proj}_{W,W'}(\rho)$ and $\mathbf{proj}_{W,W'}(\delta)$ respectively denote the projections of ρ and δ over W', obtained by existential elimination of ports or variables not belonging to W'. The results are discrete and continuous transition relations defined over W', respectively. Corresponding inverse projections are denoted by $\mathbf{proj}^{-1}_{W,W'}(\dots)$.

Product. The composition of ESM is by intersection; interaction can occur via both variables and ports:

$$
\begin{array}{rcl}
E_1 \times E_2 & = & (V, P, \rho, \delta, I, F)\,, \text{ where:} \\
V_d & = & V_{d,1} \cup V_{d,2} \quad \text{discrete variables can be shared} \\
V_c & = & V_{c,1} \cup V_{c,2} \quad \text{continuous variables can be shared} \\
P & = & P_1 \cup P_2 \quad \text{ports can be shared} \\
\rho & =_{\text{def}} & \mathbf{proj}^{-1}_{W,W_1}(\rho_1) \cap \mathbf{proj}^{-1}_{W,W_2}(\rho_2) \\
\delta & =_{\text{def}} & \mathbf{proj}^{-1}_{W,W_1}(\delta_1) \cap \mathbf{proj}^{-1}_{W,W_2}(\delta_2) \\
I & =_{\text{def}} & \mathbf{proj}^{-1}_{W,W_1}(I_1) \cap \mathbf{proj}^{-1}_{W,W_2}(I_2) \\
F & =_{\text{def}} & \mathbf{proj}^{-1}_{W,W_1}(F_1) \cap \mathbf{proj}^{-1}_{W,W_2}(F_2)
\end{array}
$$

where we recall that $W = P \uplus V$. ESMs synchronize on discrete transitions thanks to shared ports and variables. Continuous evolutions synchronize only via shared variables. If $W = W_1 = W_2$, then $\rho = \rho_1 \cap \rho_2$ and $\delta = \delta_1 \cap \delta_2$, whence the name of "composition by intersection". When specialized to continuous dynamics made of differential equations, this boils down to systems of differential equations like in undergraduate mathematics.

Our interpretation of runs with snapshot states and thick transitions (see Figure 1.3) is instrumental in allowing for the above simple and elegant definition of parallel composition "by intersection". With thick states and zero-time transitions, it is more difficult to define composition, because synchronization takes place both on states and transitions.

Union or Disjunction. The union of two sets of runs can be obtained from two ESMs by taking the union of their variables, and by adding a distinguished variable $\# \in \mathcal{V}_c$ that indicates the particular state space in which we are operating ($\# = 0$ for the first ESM, $\# = 1$ for the second). Then, we simply take the union of the transition relations after inverse projection. Formally, for i indexing the set of components involved in the considered union, let

$$\begin{array}{lll} \rho\,|^V_{\#=i} & =_{\text{def}} & \{(s,\lambda,s') \in S \times \Lambda \times S \mid s(\#) = i \text{ and } s'(\#) = i\} \\ \delta\,|^V_{\#=i} & =_{\text{def}} & \{(s,\varphi,s') \in S \times \mathcal{C}(\mathcal{V}_c) \times S \mid s(\#) = i \text{ and } s'(\#) = i\} \end{array}$$

be the transition relation that is true everywhere variable $\#$ is evaluated to i. Then,

$$\begin{array}{rcl} E_1 \cup E_2 & = & (V, P, \rho, \delta, I, F) \\ V_d & = & V_{d,1} \cup V_{d,2} \uplus \{\#\} \\ V_c & = & V_{c,1} \cup V_{c,2} \uplus \{\#\} \\ P & = & P_1 \cup P_2 \\ \rho & =_{\text{def}} & \left(\mathbf{proj}^{-1}_{W,W_1}(\rho_1) \cap \rho|^V_{\#=1}\right) \cup \left(\mathbf{proj}^{-1}_{W,W_2}(\rho_2) \cap \rho|^V_{\#=2}\right) \\ \delta & =_{\text{def}} & \left(\mathbf{proj}^{-1}_{W,W_1}(\delta_1) \cap \rho|^V_{\#=1}\right) \cup \left(\mathbf{proj}^{-1}_{W,W_2}(\delta_2) \cap \rho|^V_{\#=2}\right) \\ I & =_{\text{def}} & \left\{s \in S \mid s_{|W_1} \in I_1 \wedge s(\#) = 1\right\} \cup \left\{s \in S \mid s_{|W_2} \in I_2 \wedge s(\#) = 2\right\} \\ F & =_{\text{def}} & \left\{s \in S \mid s_{|W_1} \in F_1 \wedge s(\#) = 1\right\} \cup \left\{s \in S \mid s_{|W_2} \in F_2 \wedge s(\#) = 2\right\} \end{array}$$

Inputs and Outputs. Whenever needed we can distinguish inputs and outputs, which we also call *uncontrolled* and *controlled* ports. In this paragraph we define the corresponding algebra. Ports and variables are partitionned into inputs and outputs:

$$\begin{array}{rcl} P & = & P^I \uplus P^O \\ V & = & V^I \uplus V^O \end{array}$$

Regarding products, the set of ports of a product is again the union of the set of ports of each component. However, outputs cannot be shared.[2] That is, the product of two ESMs E_1 and E_2 is defined if and only if

$$\begin{array}{rcl} P_1^O \cap P_2^O & = & \emptyset \\ V_1^O \cap V_2^O & = & \emptyset \end{array} \tag{1.12}$$

In that case,

$$\begin{array}{rcl} P^I & = & (P_1^I \cup P_2^I) - (P_1^O \cup P_2^O), \\ P^O & = & P_1^O \cup P_2^O, \end{array} \tag{1.13}$$

with the corresponding rules for variables.

[2] we could allow sharing of outputs, and declare a failure whenever two components set a different value on an output port.

Receptiveness. For E an ESM, and $P' \subseteq P, V' \subseteq V$ a subset of its ports and variables, E is said to be (P', V')-*receptive* if and only if for all runs σ' restricted to ports and variables belonging to (P', V'), there exists a run in σ of E such that σ' and σ coincide over $P' \uplus V'$.

Receptiveness is a semantic concept. It is often implicitly meant that an ESM should be receptive with respect to its inputs. However, the example in Figure 1.4 shows that receptiveness is generally not preserved by composition, even when Condition (1.12) is satisfied and Rule (1.13) is used for the composition. This example aims at modeling an electric circuit with two components (Figure 1.4, left-hand side) , a resistor R and a voltage sensitive switch that is opened when $v < 1$ and has resistance R' when $v \geq 1$. The ESM for resistor R (Figure 1.4, right-hand side) inputs voltage u and current i and outputs voltage v. The switch ESM inputs voltage v and outputs current i. Each ESM is receptive: $v = u - Ri$ is the output of the first ESM for avery value of u and i. The second ESM outputs $i = v/R'$ when $v \geq 1$ and $i = 0$ otherwise. The composition of these two ESMs has u as only input and v and i as outputs. The system of equations admits a solution when $u < 1$, in wich case $v = u$ and $i = 0$, and when $u \geq 1 + R/R'$, in which case $v = R'/(R + R')u$ and $i = u/(R + R')$. However, it has no solution when $u \in [1; 1 + R/R')$. Clearly, the composition of the two ESMs is not receptive.

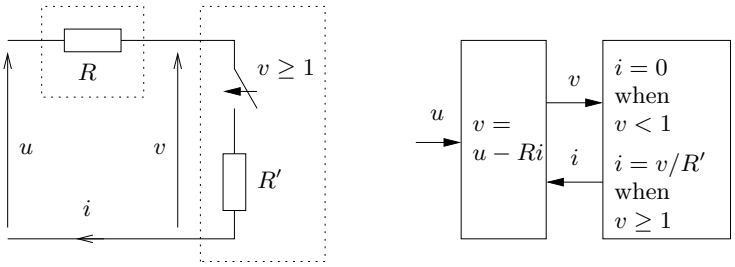

Figure 1.4: Non-receptive composition of two receptive ESMs. Left: electric circuit with two components. Right: modeling of the circuit with two receptive ESMs.

Openness. The ability to handle open systems is an important feature of extended state machines. This can be achieved by requiring that the following conditions hold for discrete and continuous transitions:

$$\{(s, \varepsilon_P, s) \mid s \in S\} \quad \subseteq \quad \rho \tag{1.14}$$

$$\left.\begin{array}{r}(s, \varphi, s') \in \delta \\ t < t_\varphi\end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l}(s, \varphi_{<t}, \mathit{Exit}(\varphi_{<t})) \in \delta \\ (\mathit{Exit}(\varphi_{<t}), \varphi_{\geq t}, s') \in \delta\end{array}\right. \tag{1.15}$$

where, by abuse of notation, we extend $\mathit{Exit}(\varphi_{<t})$ to the set of discrete variables by copying the value they had in state S.

Condition (1.14) on discrete evolutions is the usual stuttering invariance condition for discrete transition systems. It requires that it is always possible,

for an ESM, to perform a discrete stuttering transition that emits no event and leaves states unchanged. This leaves room for other components to perform discrete transitions.

Condition (1.15) on continuous evolutions expresses that it is always possible to interrupt a continuous evolution and resume it immediately. The reason for doing this is to allow for other components to perform a discrete transition (which takes no time) in the middle of a continuous evolution of the considered component.

Observe that conditions for openness imply that any finite run can be extended to an infinite one; whence our definition for accepted runs.

Locations or macro-states. Certain aggregations of states are useful for use in actual syntax. For example, hybrid systems *locations* contain the continuous evolutions. Also, *macro-states* are considered when glueing states together. Locations or macro-states are obtained in our framework by

1. selecting a subset $V'_d \subset V_d$ of discrete variables, and
2. grouping together all states s having the same valuation for all $w \in V'_d$.

For example, one could have one particular discrete variable $w \in V_d$, of enumerated type, that indexes the locations; in this case we would take $V'_d = \{w\}$. Note that the description of the dynamics still requires the discrete and continuous transitions as above. This is further elaborated on in Section 1.4.

1.4 HRC State Machines

In this section we introduce the model that corresponds to the first (classical) approach illustrated in Figure 1.3. Its interest is that it more closely fits the type of model in use when considering Timed Automata [1] or their generalization Hybrid Automata [14]. We call this model *HRC State machines*. Then we show how to translate between HRC State machines and ESMs, thus providing a way to switch to the best framework depending on the considered activity (analysis or composition). To simplify our presentation, we consider only flat HRC State Machines that do not include hierarchical or-states such as in Statecharts. Extension to hierarchical or-states raises no difficulty.

Inspired by the definition of Hybrid Automata in Henzinger [14], we define:

Definition 1.4.1 (HRC State Machine) *A* HRC State Machine *is a tuple*

$$\mathcal{H} \quad = \quad (V, P; \mathbf{G}, \mathit{init}, \mathit{inv}, \mathit{flow}, \mathit{final}; \mathit{trans}) \tag{1.16}$$

where:

- $V = V_d \uplus V_c$ *is a finite set of* variables*, decomposed into* discrete *and* continuous *variables; set* $S = D^V$*, where* D *is the domain of values;*
- P *is a finite set of* ports*;*

- $\mathbf{G}$ *is a finite directed multigraph* $\mathbf{G} = (\mathbf{L}, \mathbf{E})$*, where* $\mathbf{L}$ *is a finite set of* locations *and* $\mathbf{E}$ *is a finite set of* switches*;*
- *Four vertex labeling functions* $init, inv, flow$*, and* $final$*, that assign to each location* $\ell \in \mathbf{L}$ *four predicates;* $init(\ell), inv(\ell)$*, and* $final(\ell)$ *are expressions of boolean type over* V*, and* $flow(\ell) \subseteq \mathcal{C}(V_c)$*, see (1.6);*
- *An edge labeling function trans that assigns to each switch* $\mathbf{e} \in \mathbf{E}$ *a relation* $trans(\mathbf{e}) \subseteq S \times \Lambda \times S$*, where* $\Lambda =_{\rm def} (P \rightharpoonup D)$*.*

HRC State Machine $\mathcal{H}$ can be re-expressed as the following equivalent ESM (in that they possess identical sets of runs):

$$E_{\mathcal{H}} \quad = \quad (V \uplus \{loc\}, P, \rho, \delta, I, F)\,,$$

where:

- V is as in (1.16) and loc is an additional *location variable* taking values in the finite set $\mathbf{L}$; a value for loc is therefore a location ℓ; the corresponding set of states is the set of all possible configurations of the tuple (V, loc); such states are generically written as pairs (s, ℓ).
- P is as in (1.16).
- The discrete transition relation ρ is defined as follows:

  $$\big((s,\ell),\lambda,(s',\ell')\big) \quad \in \quad \rho$$

  if and only if there exists a switch $\mathbf{e}$ with source ℓ and target ℓ' and such that $(s, \lambda, s') \in trans(\mathbf{e})$.
- The continuous transition relation δ is defined as follows:

  $$\big((s,\ell),\varphi,(s',\ell')\big) \quad \in \quad \delta$$

  if and only if $\ell' = \ell$ and continuous evolution φ satisfies both predicates $inv(\ell)$ and $flow(\ell)$.
- $(s_0, \ell_0) \in I$ if and only if $inv(\ell_0)(s_0) = \mathtt{T}$ and $init(\ell_0)(s_0) = \mathtt{T}$.
- $(s_f, \ell_f) \in F$ if and only if $inv(\ell_f)(s_f) = \mathtt{T}$ and $final(\ell_f)(s_f) = \mathtt{T}$.

Conversely, let $E = (V, P, \rho, \delta, I, F)$ be an ESM in which a subset $loc \subset V_d$ of discrete variables has been distinguished. Then, E can be represented as the following HRC State Machine:

$$\mathcal{H}_E \quad = \quad (W, P; \mathbf{G}, init, inv, flow, final; trans) \tag{1.17}$$

where $W =_{\rm def} V - loc$ and:

- $\mathbf{G} = (\mathbf{L}, \mathbf{E})$, where $\mathbf{L} = D^{loc}$ and $\mathbf{e} = (\ell, \ell') \in \mathbf{E}$ if and only if there exists an event $\lambda \in \Lambda$ of E, such that $(\ell, \lambda, \ell') \in \mathbf{proj}_{V,loc}(\rho)$.

- For $\mathbf{e} = (\ell, \ell') \in \mathbf{E}$, $(s, \lambda, s') \in \mathit{trans}(\mathbf{e})$ if and only if $((s, \ell), \lambda, (s', \ell')) \in \rho$.
- For $\ell \in \mathbf{L}$, $\mathit{inv}(\ell)$ is satisfied by state s if and only if $((s, \ell), \lambda, (s', \ell')) \in \rho$, for some event λ, some switch $\mathbf{e} = (\ell, \ell') \in \mathbf{E}$, and some state $s' \in D^W$.
- Since, by (1.10), continuous transition relation δ does not modify discrete states, it does not modify locations. Therefore, if $(s, \varphi, s') \in \delta$, then $s(\mathit{loc}) = s'(\mathit{loc})$, we denote it by ℓ; then $\mathit{flow}(\ell)$ is the set of $\varphi \in \mathcal{C}(V_c)$ such that there exists a pair of states (s, s') with $\ell = s(\mathit{loc}) = s'(\mathit{loc})$ and $(s, \varphi, s') \in \delta$.
- $\mathit{init}(\ell)$ is satisfied by state $s \in D^W$ if and only if the pair (ℓ, s) belongs to I.
- $\mathit{final}(\ell)$ is satisfied by state $s \in D^W$ if and only if the pair (ℓ, s) belongs to F.

The following are natural questions: how does $\mathcal{H}_{E_\mathcal{H}}$ relate to $\mathcal{H}$? and how does $E_{\mathcal{H}_E}$ relate to E? These are not strictly identical but "almost" so. More precisely:

- $\mathcal{H}_{E_\mathcal{H}}$ is identical to $\mathcal{H}$.
- $E_{\mathcal{H}_E}$ identifies with E in which the subset $\mathit{loc} \subseteq V_d$ of discrete variables has been replaced by a single variable whose domain is the product of the domains of variables belonging to loc.

Having the translation of HRC State Machines into ESMs allows them to inherit from the various operators associated with ESMs. In particular

$$\mathcal{H}_1 \times \mathcal{H}_2 \quad = \quad \mathcal{H}_{E_{\mathcal{H}_1} \times E_{\mathcal{H}_2}}$$

where, in defining $\mathcal{H}_{E_{\mathcal{H}_1} \times E_{\mathcal{H}_2}}$, we take $\mathit{loc} = \mathit{loc}_1 \uplus \mathit{loc}_2$. This is an indirect definition for the product — it can also be used to define other operators on HRC State Machines. It involves the (somehow complex) process of translating HRC State Machines to ESMs and vice-versa. But one should remember that defining the product directly on HRC State Machines is complicated as well. Our technique has the advantage of highlighting the very nature of product, namely by intersection.

1.5 Mathematical syntax for the labeling functions of HRC State Machines

In this section we refine the definition of the labeling functions occurring in Definition 1.4.1 of HRC State Machines. Location or vertex labeling functions *init*, *inv*, *final*, and *flow* are specified by using expressions we introduce now. Switch or edge labeling function *trans* will be specified via a pair (guard, action), where the guard is composed of a predicate over locations and variables, and a

set of triggering events on ports; the action consists in assigning the next state following the transition. Guards and actions will also be specified by means of expressions.

1.5.1 Expressions and differential expressions

We consider two distinct copies $\mathcal{V}$ and $\mathcal{V}'$ of the set of all variables, where each $V' \in \mathcal{V}'$ is the primed version of $V \in \mathcal{V}$.

Expressions. We assume a family *Expr* of *expressions* over unprimed variables, primed variables, and ports. Thus all (partial) functions we introduce below are expressed in terms of *Expr*. Expressions are generically denoted by the symbol **E**. Whenever needed, we shall define subfamilies $Expr' \subset Expr$. This mechanism will be useful when we need to make the mathematical syntax of special families of expressions precise.

Expressions over ports. In particular, we shall distinguish $Expr_{\text{pure}}$, the family of *expressions over ports of type* **pure** (carrying no value) which involve the special operator *present* and the three combinators $\vee, \wedge, \ominus$:

$$\begin{array}{rlll} present(p) & \text{is true iff} & p \text{ occurs} \\ p_1 \vee p_2 & \text{occurs iff} & p_1 \text{ occurs or } p_2 \text{ occurs} \\ p_1 \wedge p_2 & \text{occurs iff} & p_1 \text{ occurs and } p_2 \text{ occurs} \\ p_1 \ominus p_2 & \text{occurs iff} & p_1 \text{ occurs but } p_2 \text{ does not occur} \end{array} \tag{1.18}$$

where the expression "p occurs" means that p is given a value in the considered transition, see the last bullet in Definition 1.4.1.

Differential expressions. Let

$$Expr_{|\mathcal{V}_c} \subset Expr$$

be the subfamily of expressions involving only variables belonging to $\mathcal{V}_c$. Let $Expr_{\text{cont}}$ be the set of *differential expressions,* recursively defined as:

$$\begin{array}{rcl} Expr_{\text{cont}} & \supseteq & Expr_{|\mathcal{V}_c} \\ \forall \mathbf{E} \in Expr_{\text{cont}} & \Rightarrow & \dfrac{d}{dt}(\mathbf{E}) \in Expr_{\text{cont}} \end{array} \tag{1.19}$$

where $\frac{d}{dt}(\mathbf{E})$ denotes the time derivative $\frac{d\mathbf{E}}{dt}$ of the continuous evolution of the valuation of **E**. Thus, expressions such as $\mathbf{E} \in C$, where $\mathbf{E} \in Expr_{\text{cont}}$ and C is a subset of D_c, specify *differential inclusions* [2]. Continuous evolutions defined in (1.6) are specified with the following syntax:

$$\mathbf{E} \in C \quad \text{where} \quad \mathbf{E} \in Expr_{\text{cont}} \text{ and } C \subseteq D_c$$

For $\mathbf{E} \in Expr_{\text{cont}}$, let $Exit(\mathbf{E})$ be the left limit of the valuation of **E** at the maximal instant t of its domain, see (1.7).

1.5.2 Invariants

An *invariant* is the association to a location of a pair $(inv, flow)$, see Definition 1.4.1. Invariants are generically denoted by symbol ι (the greek letter "iota"). Invariant *inv* is expressed by means ot expressions, whereas invariant *flow* uses differential expressions.

1.5.3 Mathematical syntax for transition relation *trans*

Referring to the last bullet of Definition 1.4.1, the switch labeling function *trans* is specified as a pair (γ, α) of a guard γ and an action α so that

$$\begin{array}{c}(s, \lambda, s') \in \mathit{trans} \\ \text{iff} \\ (s, \lambda) \models \gamma \text{ (the guard)} \ \wedge \ s' \in \alpha(s, \lambda) \text{ (the action)}\end{array}$$

The pair (γ, α) must be such that

$$\mathrm{dom}(\alpha) \quad \supseteq \quad \left\{ (s, \lambda) \in \overline{S} \;\middle|\; (s, \lambda) \models \gamma \right\},$$

where $\overline{S}$, guards γ, and actions α, are defined next.

Guards. Guards consist of a predicate over (previous) states, and a set of triggering events on ports. We group the two by introducing the notion of *extended states*, which consist of states augmented with valuations of ports. Formally (see Definition 1.4.1):

$$\overline{S} \quad =_{\mathrm{def}} \quad D^V \uplus \Lambda$$

A *guard* is a predicate over extended states:

$$\gamma : \overline{S} \to \{\mathtt{F}, \mathtt{T}\}$$

We say that an extended state (s, l) satisfies γ, written $(s, l) \models \gamma$, if $\gamma(s) = \mathtt{T}$. Guards can be defined as boolean valued expressions involving (unprimed) state variables and ports. Expressions over ports introduced in (1.18) are important for guards, in order to be able to refer to the presence/absence of certain ports in a given transition.

Actions. An *action* is a partial nondeterministic function over extended states:

$$\alpha \ : \ \overline{S} \ \rightharpoonup \ \wp(S')$$

where $\wp$ denotes power set. Actions assign values to primed variables, nondeterministically. It is allowed for actions to be nondeterministic in order to support partial designs.

Whereas guards generally need to use $Expr_{\mathrm{pure}}$ over ports, this is not needed for actions. Thus, the action language can be "classical", in that it does not

need to involve $Expr_{\text{pure}}$ over ports, i.e., the presence/absence of selected ports in defining the considered action. Specifying this is the role of the guard, whereas the action that follows can be restricted to refer only to values carried by ports that are known to be present in the considered transition. Whenever needed, auxiliary ports of the form $p = p_1 \vee p_2$ or $p' = p_1 \ominus p_2$ can be introduced for that purpose, when defining the guard.

1.5.4 Products in terms of guards and actions

We return now to our formalism of ESM, where products are naturally defined. The above mathematical syntax for HRC State Machines induces a corresponding mathematical syntax for ESMs. Accordingly, the product of two ESMs $E = E_1 \times E_2$ is refined as follows:

$$\begin{array}{rrcl} \text{invariants:} & \iota & = & \iota_1 \wedge \iota_2 \\ \text{guards:} & \gamma & = & \gamma_1 \wedge \gamma_2 \\ \text{actions:} & \alpha & = & \mathbf{proj}^{-1}_{W,W_1}(\alpha_1) \cap \mathbf{proj}^{-1}_{W,W_2}(\alpha_2) \end{array} \tag{1.20}$$

This formula has several interesting special cases:

- If $\gamma_i, i = 1, 2$ involves only ports of type "pure", then $\gamma_1 \wedge \gamma_2$ in (1.20) expresses that the two ESMs must synchronize on their shared ports.
- If $\iota_i, i = 1, 2$ involves only flows, then $\iota_1 \wedge \iota_2$ in (1.20) denotes the system consisting of the continuous evolutions for the two ESMs.
- If $\gamma_i, i = 1, 2$ involves only ports x, y, z, where y is shared, and have the form:
$$\begin{array}{rcl} \gamma_1 & : & y = f(x) \\ \gamma_2 & : & z = g(y) \end{array}$$
then $\gamma_1 \wedge \gamma_2$ in (1.20) denotes the conjunction of $y = f(x)$ and $z = g(y)$. This case captures the composition mechanism of dataflow formalisms — thus the composition mechanism of dataflow formalisms is supported by guards, not by actions. Note that the dependency of z on x through y is immediate, i.e., involves no logical delay.
- If $\gamma_i, i = 1, 2$ have the form
$$\begin{array}{rcl} \gamma_1 & : & y = f(x) \\ \gamma_2 & : & z = g(v_y) \end{array}$$
where y is a port and v_y is a state variable storing the value of y at previous transition:
$$\alpha_2 \quad : \quad v'_y := y$$
then $\gamma_1 \wedge \gamma_2$ introduces a *logical delay* in the composition of the two systems.

Thus, we see here a simple syntactic condition to ensure the existence of a logical delay from input ports to output ports while composing two ESMs.

1.6 Categories as specialization of HRC State Machines

We now specialize our model of HRC State Machine into several *categories* of assertions, or *viewpoints*, generically denoted by the symbol Γ. This is achieved by

1. restricting the subset of ports and variables that characterize a category; formally, we define subsets $\mathcal{P}_\Gamma \subseteq \mathcal{P}$ and $\mathcal{V}_\Gamma \subseteq \mathcal{V}$;

2. specializing how the two transition relations ρ and δ restrict to these ports and variables.

We do not need to define the synchronization of different assertions/viewpoints, as this is just a particular case of product of HRC State Machine. In fact, our HRC State Machine model has built-in cross-category heterogeneity. In the next subsections we define basic categories considered within HRC.

Semantic atoms. For categories other than "discrete", we also provide the *semantic atoms*, i.e., the minimal set of building blocks that are sufficient for building any model belonging to the considered viewpoint. Semantic atoms must be combined with a suitable model that belongs to the discrete viewpoint. They will be defined in terms of the mathematical syntax of Section 1.5. (The paragraphs on atoms can be skipped for a first reading.)

1.6.1 Discrete functional category

In a pure discrete HRC State Machine the continuous dynamics is trivial. Allowed ports and variables for this category are:

$$\begin{aligned} \mathcal{P}_\Gamma &= \mathcal{P} \\ \mathcal{V}_\Gamma &= \mathcal{V}_d \\ \mathit{flow} &= \mathbf{Triv} \end{aligned}$$

Since $V_d = \emptyset$, continuous evolutions $\varphi : \mathbb{R}_+ \rightharpoonup \emptyset \to \emptyset$ are all trivial: they just let time progress until their duration t_φ has elapsed and perform nothing else. Call **Triv** the set of all trivial continuous evolutions — note that these are entirely parameterized by their duration. Composing with **Triv** has no effect for continuous evolutions.

1.6.2 Timed category

In a timed viewpoint, only clocks are considered in combination with enumerated state variables for the discrete part:

$$\begin{array}{rcl} \mathcal{P}_\Gamma & = & \mathcal{P} \\ \mathcal{V}_\Gamma & = & \mathcal{V} \\ S_d & : & \text{finite set} \\ \forall \ell \in \mathbf{L}, \varphi \models \mathit{flow}(\ell) & \Rightarrow & \dfrac{d\varphi}{dt} \equiv 1 \text{ (corresponds to the clocks)} \end{array}$$

Semantic atoms. Atoms for timed systems are simply *timers* with their activation guard. Thus timers are HRC State Machine having two variables: the clock c (a continuous variable) and a trigger b_c, a discrete variable of boolean type. In addition, a continuous guard γ_c is provided as a constraint of the form $c \in C$, where C is some subset of the positive real line (typically, $c \leq c_{\max}$, for some threshold $c_{\max}$). Clock c is active whenever $\gamma_c \wedge [b_c = \mathtt{T}]$.

A timed system will be obtained by composing clocks with a discrete HRC State Machine providing the b_c's as outputs, and taking the exit values of the clocks as inputs. The use of this category in expressing contracts is illustrated by the following example.

Example 2 (timing pattern, Figure 1.5) Consider the timing pattern on the left hand side of Figure 1.5. It aims at specifying a timed communication medium. Its intended (informal) meaning is that, whenever the delay between the two events s_b and t_b is less than τ_b, then it is guaranteed that the delay between the two events s_a and t_a is less than τ_a. On the right hand side of this figure, we show two assertions: A, and $\neg G$. The pair (A, G) constitutes contract C. Ports of C are: s_a, s_b, t_b, t_a, e. Among these ports, s_a and t_b are uncontrollable. The two clocks h_a and h_b are local variables of the contract; they satisfy the dynamics $\frac{dh}{dt} = 1$. Assertion A emits e whenever the desired pattern is completed with the due timing constraint on the pair s_b, t_b. Assertion G ensures that, whenever e is received, then the timing constraint on the pair s_a, t_a is satisfied. This contract is not in canonical form. To make it in canonical form, simply replace $\neg G$ by the product $A \times \neg G$. $\diamond$

1.6.3 Hybrid category

The hybrid category simply corresponds to the general case of HRC State Machines.

Semantic atoms. Atoms for hybrid systems are *differential inclusions* with their guard. Differential inclusions are HRC State Machines having two sets of variables: a set $X = \{X_1, \ldots, X_n\}$ of continuous variables and the trigger b_X, a discrete variable of boolean type. In addition, a continuous guard γ_X

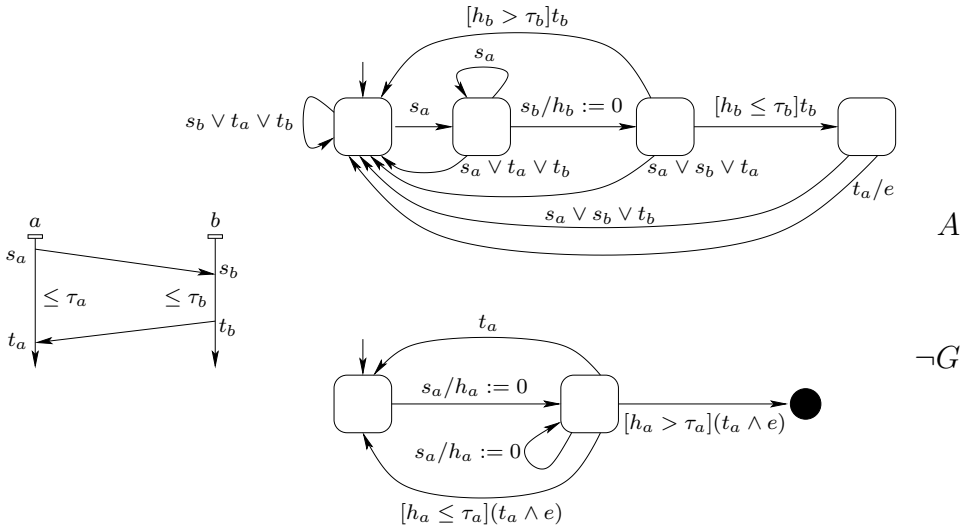

Figure 1.5: Assumption/Promise. Left: represented informally as a timing pattern. Right: represented as the contract $C = (A, G)$ (the black circle is an accepting state).

is provided as a constraint of the form $\exp_c(X_1, \ldots, X_n) \in C$, where $\exp_c$ is some differential expression with values in $\mathbb{R}^p$ and C is some subset of $\mathbb{R}^p$. The differential inclusion is active whenever $\gamma_X \wedge b_X$ holds.

A hybrid system is obtained by composing clocks with a discrete HRC State Machine providing the b_X's as outputs, and taking the exit values of the differential inclusions as inputs. Figure 1.6 gives such a decomposition for a variant of the electric circuit presented in Figure 1.4. The switch is modeled with three hybrid atoms: one for each state of the switch (opened and closed) and one for controlling the two former atoms. Consider the hybrid atom $j = v/R'$. When clock b is true, variable j is controlled by this atom, otherwise it is not constrained by this atom.

1.6.4 Safety or Probabilistic category

Probabilistic ESMs specify what is considered random in a given ESM. Such a framework is useful when dealing with reliability models in which reliability properties interact with functional properties. For example, the risk for a component to fail may become zero in certain operating modes. In this category we provide means to specify such systems in a flexible yet simple way. More precisely, we assume that randomness will apply only to a specified subset $\mathbf{p}$ of ports. To be consistent with our approach, $\mathbf{p}$ must consist of ports that make the considered ESM receptive. The idea is that the environment will be the source of randomness for these ports. An element of the safety category thus consists of the following:

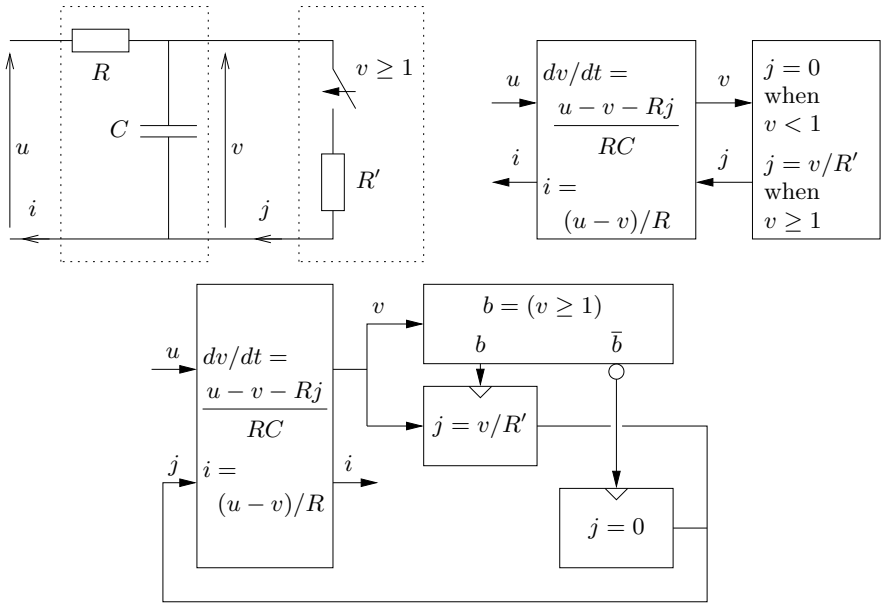

Figure 1.6: Use of clocked hybrid atoms. Top-left: Electric circuit. Top-right: Modeling of the circuit as a composition of ESMs. Bottom: Composite state machine with clocks to control hybrid atoms.

- An HRC State Machine $\mathcal{H}$ with set of ports P.
- A subset $\mathbf{p} \subseteq P$ of *probabilistic ports,* such that $\mathcal{H}$ is $\mathbf{p}$-receptive, see Section 1.3.2 the definition of receptiveness.
- For each $p \in \mathbf{p}$, an *activation port* $a_p \in P$ of pure type. Each event received on port a_p triggers the emission of an event on port p with a value drawn at random from some distribution μ_p. The different random trials are independent between different probabilistic ports.

Probabilistic ports are categorized into *time-related* and *value-related.* If port p is time-related, then μ_p is a distribution on $\mathbb{R}_+$ or $\mathbb{N}_+$ and the value emitted by p is interpreted as a timing delay (e.g., for use in modeling the occurrence of failures). The probabilistic semantics is straightforward. Since $\mathcal{H}$ is $\mathbf{p}$-receptive:

1. one can draw at random the entire random sequence for each probabilistic port p (it need not be an independent sequence, it can be Markov, or even general);
2. these random sequences are stored for feeding the probabilistic ports of $\mathcal{H}$;
3. each probabilistic sequence of data is then offered to $\mathcal{H}$ when activation port a_p requests these.

Comments. Note that this is still compatible with nondeterminism. And other ways of modeling failure generation can be considered. For some applications, failures can be state-dependent. If there are only finitely many such dependencies, then just provide one random source per different possible failure, and select the right one in a state-dependent way. If correlation between failures must be covered, this can be generally achieved by generating appropriate joint distributions by transforming joint distributions for independent random variables. Of course, all these tricks have a cost, and it will be the role of the use cases to check feasibility of this simple and pragmatic approach.

Semantic atoms. Semantic atoms for the safety category consist of an HRC State Machine $\mathcal{H}_p$ having one probabilistic port p, the associated activation port a_p, associated distribution μ_p, and no variable.

Composing probabilistic ESMs. For $i = 1, 2$, let $\mathcal{P}_i = (\mathcal{H}_i, P_i, \mathbf{p}_i, (\mu_p^i)_{p\in\mathbf{p}_i})$ be two probabilistic ESMs. Their parallel composition $\mathcal{P}_1 \parallel \mathcal{P}_2$ is defined only if

$$\mathbf{p}_1 \text{ and } \mathbf{p}_2 \text{ are two disjoint sets of uncontrolled ports in } \mathcal{H}_1 \parallel \mathcal{H}_2. \tag{1.21}$$

Then,

$$\mathcal{P}_1 \parallel \mathcal{P}_2 = (\mathcal{H}, P, \mathbf{p}, (\mu_p)_{p\in\mathbf{p}}) \text{ where } \begin{cases} \mathcal{H} &=& \mathcal{H}_1 \parallel \mathcal{H}_2 \\ \mathbf{p} &=& \mathbf{p}_1 \uplus \mathbf{p}_2 \\ \mu_p &=& \mu_p^i, \text{ where } i \text{ is such that } p \in \mathbf{p}_i \end{cases}$$

Comments regarding Condition (1.21) and a technique of wrappers. The reason for Condition (1.21) is to keep composition simple for probabilistic systems. If this condition does not hold, then indirect coupling between the probabilities may occur, due to constraints resulting from taking the product $\mathcal{H}_1 \parallel \mathcal{H}_2$. Condition (1.21) allows us to capture failure models, as well as random timing models for input signals.

The consequences of Condition (1.21) regarding compositionality are, however, non trivial, as the following example shows. Consider a situation where we have a component having a port x which is either a source of failure, or is subject to failure propagation from another component. In the first case, the model of this component should look like $\mathcal{P} = (\mathcal{H}, P, \mathbf{p}, \mu)$, where $\mathbf{p} = \{x\}$ and port x is uncontrolled. The second case, on the other hand, may be obtained by composing $\mathcal{P}$ with another ESM in which x is an output and therefore controlled. This is ruled out by our Condition (1.21), however. Thus, it seems that this definition prevents us from capturing the above natural situation.

However, a simple mechanism of wrappers solves the problem as we explain next. Isolate the non probabilistic part $\mathcal{H}$ of our probabilistic ESM $\mathcal{P} = (\mathcal{H}, \mathbf{p}, \mu)$. Next, wrap $\mathcal{H}$ with the following small probabilistic ESM $\mathcal{P}_x$, which has one controlled port x and three uncontrolled ports: x_{source}, x_{herit}, and an additional port c taking values in the set $\{source, herit\}$. The only

probabilistic port of $\mathcal{P}_x$ is x_{source}, we equip it with the original probability distribution μ. There is no assumption for ESM $\mathcal{P}_x$, and its guarantee is the following assertion

$$E =_{\text{def}} \quad x = x_{source} \text{ if } c = source \text{ else } x = x_{herit} \text{ if } c = herit,$$

which specifies a selector. Wrapping our original ESM in this way prepares it for the desired parallel composition in a valid way.

This is illustrated in Figure 1.7. In this figure, thick triangles denote prob-

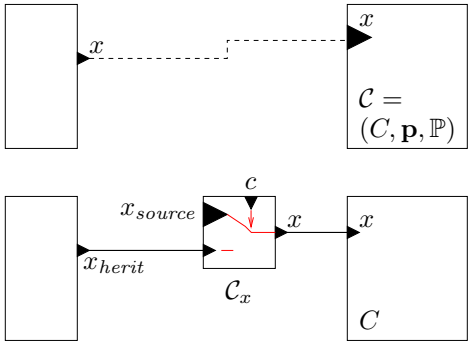

Figure 1.7: Illustrating the wrapper mechanism.

abilistic ports. The incorrect composition is shown at the top; it gives rise to a mismatch between thick and thin triangles. The corrected version, with its wrapper $\mathcal{P}_x$, is shown at the bottom. Probabilistic ESM $\mathcal{P}_x$ has one probabilistic port x_{source} with probability μ, and one uncontrolled port x_{herit}; uncontrolled boolean port c selects which input is propagated to the wrapped ESM $\mathcal{H}$. The design can be prepared for composition by this mechanism of wrapping. Wrapping must be performed manually, however.

1.6.5 Illustrating multi-viewpoint composition

Our approach aims at supporting component based development of heterogeneous embedded systems with multiple viewpoints, both functional and non-functional. The following simple example illustrates this for the case of functional, timed, and safety viewpoints. The overall system architecture is shown on Figure 1.8. It consists of a simple controller that can let the underlying plant "start", "stop", or "work" (signals r, s, and w). The controller is subject to "failure" **f** of fail/stop type. The underlying plant has limited capacity and thus the controller should not accumulate in excess of w messages during a certain period. This is ensured by the supervisor. The supervisor monitors the flow of w's. When they get too frequent, an "overloaded" message **o** is sent to the controller, which reduces the controller's pace. When appropriate, the human operator can decide to switch the controller back to its nominal mode, by sending the "cleaned" message **c** to the pair controller/supervisor.

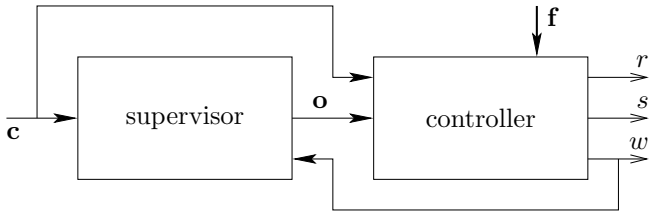

Figure 1.8: The overall system architecture.

This system involves three viewpoints: functional, Quality of Service (QoS) of timed nature, and safety. In designing her system, the designer may follow three different methodologies. She may consider each of the two components with its three viewpoints, implement each of them and then compose the result. Alternatively, she may perform a first design by ignoring the safety viewpoint. The safety aspect is then added in a second stage. Finally, she may consider all contracts for all components in a flat manner. The semantics of our framework has been designed to yield consistent results when following these trhee methods. For more details on this aspect, we refer the interested reader to [5].

The different contracts. Figure 1.9 depicts the set of contracts associated to the controller. For each contract, we show its assumption (top) and promise (bottom). The third contract has trivial, empty assumption. Assumptions are specified as observers, meaning that the corresponding diagrams define the negation of these assumptions. In these diagrams, the circles filled in black denote accepting states.

The first contract C_{funct} describes the functional aspect under the no failure assumption: the controller is activated by commands r ("run") and s ("stop"), and it can let the controlled system (not shown) work, by performing action w. This contract holds in absence of a failure — shown by its assumption.

Contract C_{QoS} indicates that, under the no failure assumption, there exist two modes: nominal and degraded. Event **o** (for "getting overloaded") is not controlled by this component; in turn, when in overloaded mode, the human operator (not shown) can decide to perform "cleaning", corresponding to input event **c** to the system. This contract holds in absence of a failure — shown by its assumption. Contract C_{QoS} relates to timing. When in nominal mode, the controller performs its task (whose termination is abstracted with the action w) in at most τ_n milli-seconds. When in degraded mode, the controller performs its task in at least τ_d milli-seconds, with $\tau_d > \tau_n$.

Finally, contract C_{safety} specifies the safety aspect, which under no assumption states that a fault can occur at any time.

Figure 1.10 depicts the QoS contract for the supervisor in charge of avoiding system collapse by turning it to degraded mode. The assumption is trivial since the supervisor is not subject to failure. The promise is specified in terms of a hybrid automaton of the timed category. This hybrid automaton uses a timer x bound to physical time, thus satisfying the differential equation $\dot{x} = 1$ (x

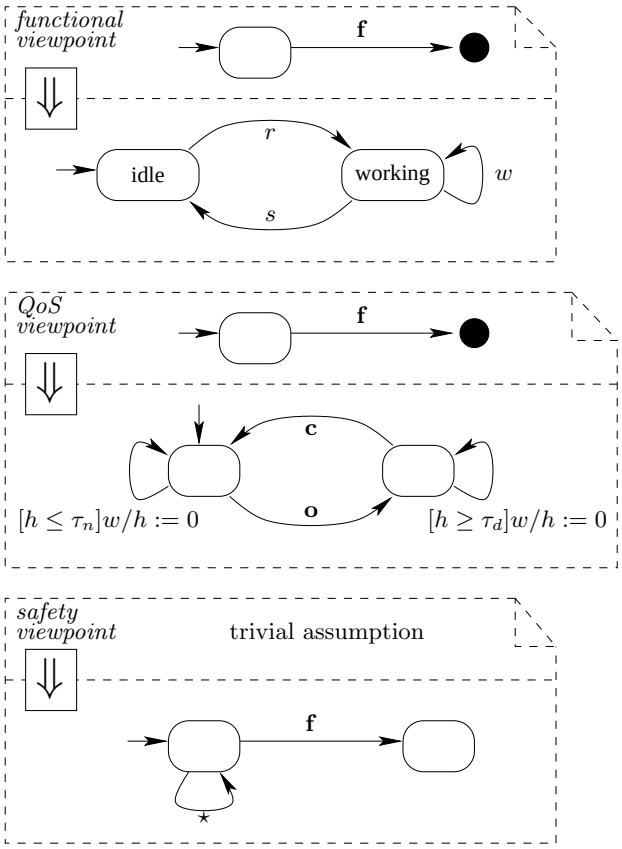

Figure 1.9: The three contracts C_{funct}, C_{QoS}, and C_{safety} specifying the three viewpoints of the controller. The assumption is put on top of the promise and both are separated by the implication symbol $\boxed{\Downarrow}$.

increases with constant speed 1). The behavior of this timer is depicted on the second diagram. When action w occurs too frequently in the long range, timer x starts decreasing and eventually reaches zero, which causes the emission of message **o** and switches the mode to "overloaded", where latency is at least τ_d. At some point, the cleaning message **c** is input by the operator, which resets the timer to 0 and brings the system back to its nominal mode.

1.7 Conclusion

We have briefly presented a framework for multiple viewpoint contracts. This framework is supported by the Heterogeneous Rich Component (HRC) metamodel, for which we have presented an underlying mathematical model of machine. We have emphasized how to support the combination of different view-

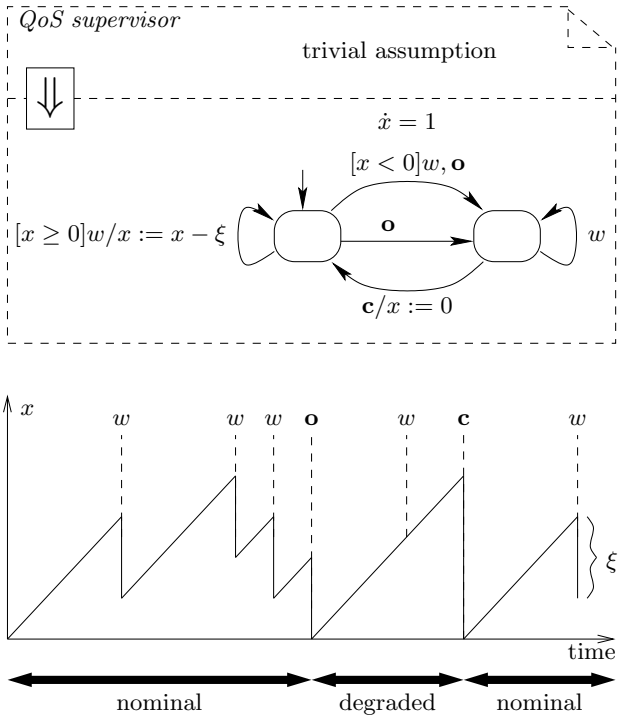

Figure 1.10: Contract C_s of the supervisor and its behaviour.

points and have provided a simple and elegant notion of parallel composition.

In order to support partial designs, we have favored a constrained, non functional, style for our model. Also, we have considered that our systems are open, i.e., are subject to further combination with other, yet to be defined, subsystems. Our mathematical model is stratified in that it is progressively refined by detailing more and more its mathematical objects — from abstract transitions to combinations of guards and assignments.

One important feature of this model is that it has two equally important (and equivalent) versions. In the first version, states are snapshots whereas transitions are "thick" — transitions support continuous progress and invariants. For this version, parallel composition is by intersection, which is particularly simple and elegant. In the second version, transitions are snapshots whereas states are "thick" — states support continuous progress and invariants. This second version conforms to region based models of systems, which are preferred by model checking tools. We have shown how the two versions can be intertranslated. Since the notions of state and transition are in fact interchanged between the two versions, it was essential not to constrain the way systems can interact. We have thus chosen to support both common state variables and common ports as vehicles for interaction.

Finally, we have characterized *categories*, i.e., subclasses of systems focusing

on a particular aspect or viewpoint. One particular category required specific attention in dealing with parallel composition, namely the probabilistic one.

The resulting mathematical model is the basis for a precise behavioral semantics for the HRC metamodel and provides a precise semantic for component composition, an often neglected issue in the design of frameworks for component based design.

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Jean-Paul Aubin and Arrigo Cellina. *Differential Inclusions, Set-Valued Maps And Viability Theory*, volume 264 of *Grundl. der Math. Wiss.* Springer, 1984.

[3] R.-J. Back and J. von Wright. *Refinement Calculus: A systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.

[4] R.-J. Back and J. von Wright. Contracts, games, and refinement. *Information and communication*, 156:25–45, 2000.

[5] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects (FMCO07)*, Revised Lectures, Lecture Notes in Computer Science, Amsterdam, The Netherlands, October 24–26 2007.

[6] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. A generic model of contracts for embedded systems. Rapport de recherche 6214, Institut National de Recherche en Informatique et en Automatique, June 2007.

[7] Henning Butz. The Airbus approach to open Integrated Modular Avionics (IMA): technology, functions, industrial processes and future development road map. In *International Workshop on Aircraft System Technologies, Hamburg*, March 2007.

[8] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Marielle Stoelinga. Resource interfaces. In *Proceedings of the Third Annual Conference on Embedded Software (EMSOFT03)*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.

[9] Werner Damm. Controlling speculative design processes using rich component models. In *Fifth International Conference on Application of Concur-*

rency to System Design (ACSD 2005), pages 118–119, St. Malo, France, June 6–9 2005.

[10] Werner Damm. Embedded system development for automotive applications: trends and challenges. In *Proceedings of the* 6^{th} *ACM & IEEE International conference on Embedded software (EMSOFT06)*, Seoul, Korea, October 22–25 2006.

[11] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.

[12] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, August 1975.

[13] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[14] Thomas A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.

[15] Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Permissive interfaces. In *Proceedings of the* 13^{th} *Annual Symposium on Foundations of Software Engineering (FSE05)*, pages 31–40. ACM Press, 2005.

[16] L. Lamport. win and sin: Predicate transformers for concurrency. *ACM Transactions on Programming Languages and Systems*, 12(3):396–428, July 1990.

[17] Bertrand Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, October 1992.

[18] Radu Negulescu. Process spaces. In *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[19] Alberto Sangiovanni-Vincentelli. Reasoning about the trends and challenges of system level design. *Proc. of the IEEE*, 95(3):467–506, 2007.