On Resource-Sensitive Timed Component Connectors [⋆]

Sun Meng and Farhad Arbab

CWI, Kruislaan 413, Amsterdam, The Netherlands
{Meng.Sun,Farhad.Arbab}@cwi.nl

Abstract. In this paper we introduce a formal model for reasoning about resource sensitive timed component connectors. We extended the constraint automata model, which is used as the semantic model for the exogenous channel-based coordination language Reo, through integrating both resource and time information. This model allows to specify both the interactions that take time to be performed and timeouts. Moreover, the model reflects resource issues, such as bandwidth or allocated memory, that may affect the time needed for interactions when specifying the timed behavior of connectors. The time duration that an interaction takes is represented by a function on the available resources. In addition to the formalism, we also discuss compositional reasoning and present two notions of simulation to relate different connectors from functional and resource-sensitive temporal perspectives respectively.

Keywords: Coordination, Constraint Automata, Resource-Sensitive Timed Constraint Automata, Simulation

1 Introduction

One important challenge of the software engineering field is the so called Service Oriented Computing (SOC) [11]. In SOC, applications are developed by coordinating the behaviour of autonomous services distributed over an overlay network. Coordination models and languages provide a formalization of the “glue code” that interconnects the services and organizes the mutual interactions between them in a distributed processing environment, and are extremely important to the success of SOC. Several coordination models have been proposed in the literature. For example, Reo [3, 5] offers a powerful glue language for implementation of coordinating component connectors based on a calculus of mobile channels. However, most of them were concerned only with functional aspects of the connectors. This means that nothing was said about Quality of Services [12], e.g., the duration of the interaction. As a consequence, only functional properties of coordination could be investigated.

Coordination of services requires service consumers to discover service providers that satisfy both given functional and non-functional requirements, including costs and QoS requirements such as time that a service takes to perform a certain action. Timing constraints are always required to be satisfied in different service oriented applications and the time consumed during the execution of a service falls into one of the following two categories:

[⋆] The work reported in this paper is supported by a grant from the GLANCE funding program of the Dutch National Organization for Scientific Research (NWO), through project CooPer (600.643.000.05N12).

– The service consumes time while it performs actions. The time may depend on the availability of resources. For example, the time for downloading a file from a server depends on the bandwidth of the network.
– The time passes while the system waits for a reaction from the environment. In particular, the service can change the system's state if an interaction is not received before a certain amount of time. For example, the connection to some server (like internet banking) might be disconnected if it does not receive any requirement for a long time due to the security reason.

In this paper we consider the temporal issues of Reo which allow the specifier to define how the behavior of channels and component interfaces can be affected by both categories of temporal aspects. Although there are a plethora of timed extensions of classical models [1, 16, 17], most of them only specify one temporal aspects: Time is either associated with actions or associated with delays/timeouts. We present a formalism based on constraint automata, allowing us to take into account both temporal issues considered before, and specify in a natural way both aspects of temporal properties for connectors. Furthermore, a new contribution in this paper is that resources may influence the timing property of the behavior. Therefore the execution of an interaction may take different time values if the available resources are different.

The choice of Reo as the coordination language (and therefore constraint automata as its operational semantic model) is motivated by the fact that (1) it allows exogenous coordination and arbitrary user defined primitives, and (2) it is unique among other models in allowing arbitrary combination of synchrony and asynchrony in coordination protocols. This, for instance, facilitates multi-party transactions through Reo's inherent propagation of synchrony and exclusion constraints.

We also propose a formal simulation relation allowing to systematically compare connectors given by resource sensitive timed constraint automata. The notion of simulation of ordinary constraint automata has already been studied in the literature [2, 4, 5]. However, to the best of our knowledge, none of them take resource issues into account. Here, we propose new techniques specifically devoted to resource sensitive timed connectors. Regarding functional simulation we have to consider not only language inclusion as discussed in [5], but also the possible timeouts.

Taking resources and time into account, the simulation relation in our model can be used to check the standard refinement pattern: Having a certain requirement in mind, it is often quite easy to depict a resource sensitive timed constraint automaton A that describes the allowed behavior. In this sense, A can serve as specification for a Reo circuit that is to be designed. A Reo circuit G is viewed to be correct (w.r.t. specification A) iff the resource sensitive timed constraint automaton A_G for G does not show any behavior that is forbidden by the specification, where both functional behavior and temporal behavior are considered.

The rest of the paper is organized as follows: In Section 2 we recall the basic concepts of ordinary constraint automata. Resource sensitive timed constraint automata are introduced in Section 3. In Section 4 we present the simulation relation in our model and provide the congruence result with respect to the composition operators. In Section 5 we consider related research work. We conclude in Section 6 with a brief discussion of some further work.

2 Constraint Automata

Constraint automata (CA) were introduced in [5] as a formalism to capture the operational semantics of channel-based component connectors in Reo. This section summarizes the basis concepts of constraint automata. Constraint automata are variants of labelled transition systems where transitions are augmented with pairs N, g rather than action labels. The states of a constraint automata stand for the network configurations, e.g., the contents of the buffers for FIFO channels. The transition labels N, g can be viewed as sets of I/O-operations that will be performed in parallel. More precisely, N is a set of nodes in the network where data-flow is observed simultaneously, and g is a boolean condition on the observed data items. Transitions going out of a state s represent the possible data-flow in the corresponding configuration and its effect on the configuration.

CA use a finite set $\mathscr{N}$ of nodes. The nodes can play the role of input and output ports of components and connectors, but they can appear outside the interfaces of components as intermediate stations of the network where several channels are glued together and the transmission of data items can be observed. In the sequel, we assume a finite and non-empty set $Data$ consisting of data items that can be transferred through channels. A data assignment denotes a function $\delta : N \to Data$ where $\emptyset \neq N \subseteq \mathscr{N}$. We write δ_A for the data item assigned to node $A \in N$ under δ and $DA(N)$ for the set of all data assignments for node-set N. CA use a symbolic representation of data assignments by data constraints which mean propositional formulas built from the atoms $d_A = d_B$, $d_A \in P$ or $d_A = d$ where A, B are nodes, d_A and d_B are symbols for the observed data item at node A and $d \in Data$, $P \subseteq Data$ (and the standard boolean connectors $\wedge$, $\vee$, $\neg$, etc.). For a node set N, $DC(N)$ denotes the set of data constraints that only refer to the terms d_A for $A \in N$.

Definition 1. *A constraint automaton over the data domain $Data$ is a tuple $\mathscr{A} = (S, S_0, \mathscr{N}, \longrightarrow)$ where S is a set of states, also called configurations, $S_0 \subseteq S$ is the set of its initial state, $\mathscr{N}$ is a finite set of nodes, $\longrightarrow \subseteq \bigcup_{N \subseteq \mathscr{N}} S \times \{N\} \times DC(N) \times S$, called the transition relation.*

A transition fires if it observes data items in its respective ports/nodes of the component and according to the observed data, the automaton may change its state. We write $s \xrightarrow{N,g} s'$ instead of $(s, N, g, s') \in \longrightarrow$ and refer to N as the node-set and g the guard for the transition. By an instance of $s \xrightarrow{N,g} s'$ we mean a transition of the form $s \xrightarrow{N,\delta} s'$ where δ is a data assignment for the nodes in N with $\delta \models g$. Here the symbol $\models$ stands for the satisfaction relation which results from interpreting data constraints over data assignments.

The intuitive operational behavior of a constraint automaton can be specified by its *runs*. A run in a constraint automaton is defined as a (finite or infinite) sequence of consecutive transition instances

$$r = s_0 \xrightarrow{N_0,\delta_0} s_1 \xrightarrow{N_1,\delta_1} s_2 \xrightarrow{N_2,\delta_2} \dots$$

We require that runs are either infinite or finite runs where the last state s_n does not have any outgoing transition where the node set N only consists of mixed nodes. This requirement can be understood as a *maximal progress assumption* for the mixed nodes.

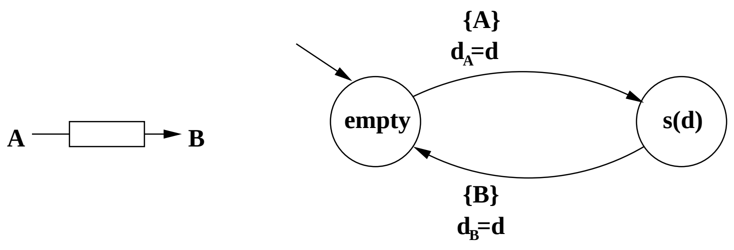

Fig. 1. Constraint Automata for FIFO Channel

Figure 1 shows a CA for a FIFO1 channel AB in Reo, which is given in the left of the picture. Node A serves as input port where data items can be written into the channel while B can be regarded as output port where the stored data item is taken out and delivered to the environment. State $empty$ represents the configuration in which the buffer is empty, while state $s(d)$ stands for the configuration where data element d is stored in the buffer. The CA given here is a parametric version with meta symbols for data items (Formal definition can be found at [5]).

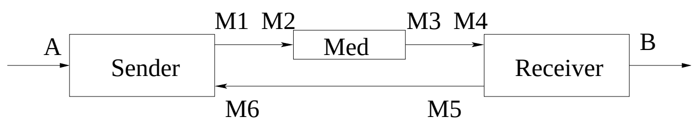

Fig. 2. The Components of Parrow's Protocol

Example 1. As another example, we consider the Parrow's Protocol (PP) [18], which is a simplified version of the well-known Alternating Bit Protocol (ABP). PP provides an error free communication over a medium that might lose messages. Figure 2 shows the components that are involved in this protocol. Data elements from a set Msg are communicated between a Sender and a Receiver. Once the Sender reads a message from its port A, it sends this datum through the communication medium Med to the Receiver, which sends the message out through its port B. The communication medium Med is faulty, thus a message sent through Med can turn up as an error message. Every time the Receiver receives a message via Med, it sends an acknowledgement to the Sender. For simplicity it is assumed that acknowledgements are never lost. We model three components and the three synchronous channels by CA. The pictures are given as in Figure 3.

Constructing complex connectors out of simpler ones is done by the join operation in Reo. Joining two nodes destroys both nodes and produces a new node on which all of their coincident channel ends coincide. Each channel in Reo is mapped to a constraint automaton. We now show how Reo's join operation can be realized by the product construction of constraint automata.

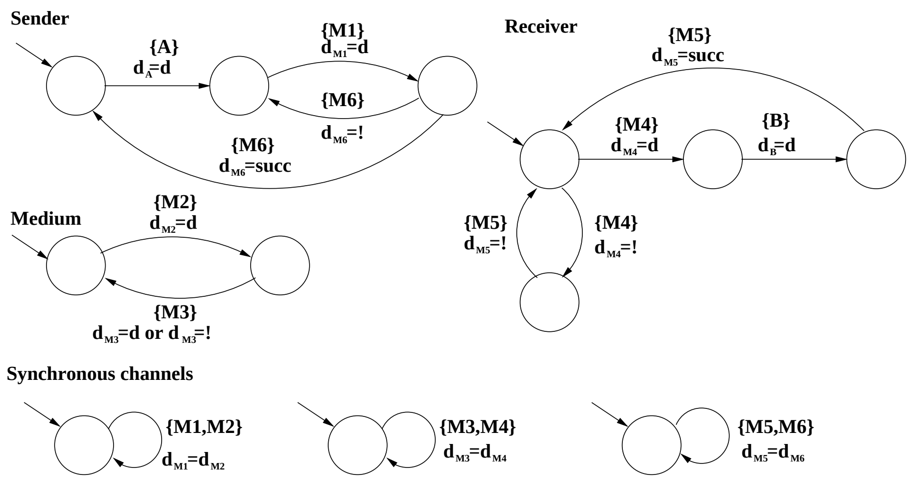

Fig. 3. The Constraint Automata for the Components and Channels in Parrow's Protocol

The product for two given constraint automata $\mathscr{A}_1 = (S_1, s_{0,1}, \mathscr{N}_1, \longrightarrow_1)$ and $\mathscr{A}_2 = (S_2, s_{0,2}, \mathscr{N}_2, \longrightarrow_2)$ is defined as a constraint automaton $\mathscr{A}_1 \bowtie \mathscr{A}_2$ with the components

$$(S_1 \times S_2, \langle s_{0,1}, s_{0,2} \rangle, \mathscr{N}_1 \cup \mathscr{N}_2, \longrightarrow)$$

where $\longrightarrow$ is given by the following rules:

– If $s_1 \xrightarrow{N_1, g_1}_1 s_1'$, $s_2 \xrightarrow{N_2, g_2}_2 s_2'$, $N_1 \cap \mathscr{N}_2 = N_2 \cap \mathscr{N}_1 \neq \emptyset$ and $g_1 \wedge g_2$ is satisfiable, then $\langle s_1, s_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle s_1', s_2' \rangle$.
– If $s_1 \xrightarrow{N,g}_1 s_1'$, where $N \cap \mathscr{N}_2 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N,g} \langle s_1', s_2 \rangle$.
– If $s_2 \xrightarrow{N,g}_2 s_2'$, where $N \cap \mathscr{N}_1 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N,g} \langle s_1, s_2' \rangle$.

The first rule is applied when there are two transitions in the automata which can be fired together. This happens only if there is no shared name in the two automata that is present on one of the transitions but not present on the other one. In this case the transition in the resulting automaton has the union of the name sets on both transitions, and the data constraint is the conjunction of the data constraints of the two transitions. The second rule is applied when a transition in one automaton can be fired independently of the other automaton, which happens when the names on the transition are not included in the other automaton. The third rule is symmetric to the second one. A parametric picture for the product of the CA of the Sender, the Receiver, the Medium and the synchronous channels in Example 1 is given in Figure 4.

Another operator that is helpful for abstraction purposes and can be used in Reo to build connectors from networks by declaring the internal topology of the network as hidden is the hiding operator. Hiding takes an input a constraint automaton $\mathscr{A} = (S, s_0, \mathscr{N}, \longrightarrow)$ and a non-empty node-set $M \subseteq \mathscr{N}$. The result is a constraint automaton $\texttt{hide}(\mathscr{A}, M)$ that behaves as $\mathscr{A}$ except that data flow at the nodes $A \in M$ is made

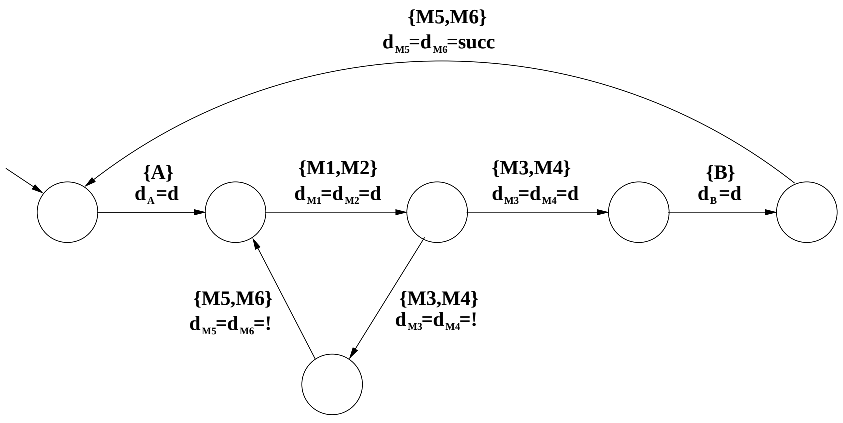

Fig. 4. The Product of Constraint Automata for the Components and Channels in Fig.3

invisible. Formally, $\texttt{hide}(\mathscr{A}, M) = (S, s_0, \mathscr{N} \setminus M, \longrightarrow_M, Q_{0,M})$ where $s \xrightarrow{\bar{N},\bar{g}}_M s'$ iff there exists a transition $s \xrightarrow{N,g} s'$ such that $\bar{N} = N \setminus M$ and $\bar{g} = \exists M[g]$. Here $\exists M[g]$ stands short for $\bigvee_{\delta \in DA(M)} g[d_A/\delta.A|A \in M]$, where $g[d_A/\delta.A|A \in M]$ denotes the syntactic replacement of all occurrences of d_A in g for $A \in M$ with $\delta.A$. Therefore, $\exists M[g]$ formalizes the set of data assignments for $\bar{N}$ that are obtained from a data assignment δ for N where g holds by dropping the assignments for the nodes in $N \cap M$.

3 Resource-Sensitive Timed Constraint Automata

In this section, we present an extension of the constraint automata model for Reo circuits that yields the basis for reasoning about resources in temporal behavior of channel-based component connectors. *Resource-Sensitive Timed Constraint Automata* (RSTCA for short) rely on the assumption that the execution time of interactions depends on the available resources, while timeout is also permitted. As we have indicated previously, we will add new dimensions to the CA model such that the temporal properties can be properly specified. We consider both timeout behavior and the time being taken when the interactions being executed in the system evolution. The time values will not only depend on the corresponding operation to be performed and the state that the system resides in. Therefore, we have two types of transitions:

– *interactive* transitions where the time needed for the interaction depends on the available resource value, and
– *timeout* transitions where the system can evolve after a given time while no interaction happens.

Before touching the technical details for RSTCA, we first consider a mathematical account of the notion of resource. According to [19], the following properties are reasonable requirements for a model of resource:

– A set $\mathbf{R}$ of resource elements;
– A (partial) combination $\circ : \mathbf{R} \times \mathbf{R} \rightharpoonup \mathbf{R}$ of resource elements;
– A comparison $\sqsubseteq$ of resource elements; and
– A zero resource element e.

which correspond to a preordered partial commutative monoid $(\mathbf{R}, \circ, e, \sqsubseteq)$, subject to the condition that if $r \sqsubseteq s$ and $r' \sqsubseteq s'$ then $r \circ r' \sqsubseteq s \circ s'$. For simplicity, we use $(\mathbb{N}, +, 0, \leq)$ as the model of resource in the following. A resource assignment for resource r is given by $r : n$ which means that n units of resource r is available. In general, a resource assignment is a tuple of resource assignments $\langle r_1 : n_1, r_2 : n_2, \cdots, r_k : n_k \rangle$ for resources $r_1, r_2, \cdots, r_k$. A resource constraint rc for resource $r_1, r_2, \cdots, r_k$ is a conjunction of atoms of the form $r_i \bowtie m$ where $\bowtie \in \{<, \leq, >, \geq, =\}$. RA denotes the set of all resource assignments and RC the set of all resource constraints. We use the symbol $\models$ for the satisfaction relation for resource constraints which results from interpreting resource constraints over resource assignments. The judgement $r : n \models rc$ is read as "resource assignment $r : n$ is sufficient to satisfy rc". We say that a resource constraint rc is satisfiable if there exists a resource assignment x such that $x \models rc$. The monoidal structure allows us to define a multiplicative conjunction $\otimes$ on resource constraints, which is given by

$$r : n \models rc_1 \otimes rc_2 \text{ iff there are two assignments } r : n_1 \text{ and } r : n_2 \text{ such that}$$
$$n_1 \circ n_2 \sqsubseteq n, \text{ and } r : n_1 \models rc_1 \text{ and } r : n_2 \models rc_2$$

The semantics of such a multiplicative conjunction is: the n units of resource r is sufficient to satisfy $rc_1 \otimes rc_2$ just in case that it can be divided into two parts n_1 and n_2 such that n_1 units of r satisfies rc_1 and n_2 units of r satisfies rc_2.

During the rest of the paper we will use the following notation: $\mathbf{T} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ is the domain to define time values. We write $\mathbf{R}\mid_{rc}$ for the subset of $\mathbf{R}$ in which all the elements satisfy the resource constraint rc and $\{\mathbf{R} \to \mathbf{T}\}$ for the function space from $\mathbf{R}$ to $\mathbf{T}$, i.e., the set of possible functions with domain $\mathbf{R}$ and codomain $\mathbf{T}$.

Definition 2 (Resource-Sensitive Timed Constraint Automata). *A RSTCA is defined as a tuple*

$$\mathscr{T} = (S, S_0, \mathscr{N}, R, \longrightarrow)$$

where S is a countable set of control states (also called locations), $S_0 \subseteq S$ is the set of initial states, $\mathscr{N}$ is a finite set of nodes, R is a finite set of resource names, and the edge

$$\longrightarrow \subseteq (S \times \mathbf{T} \times S) \cup (\bigcup_{N \subseteq \mathscr{N}} S \times \{N\} \times DC(N) \times RC \times \{\mathbf{R} \to \mathbf{T}\} \times S)$$

denotes the transitions and we have two types of transitions:

– *timeout transitions: $s \xrightarrow{t} s'$ where $t \in \mathbf{T}$;*
– *interactive transitions: $s \xrightarrow{N,g,rc,C} s'$ where N, g are as in ordinary constraint automata, rc is the resource constraint that should be satisfied to trigger the execution of the transition, and $C : \mathbf{R}\mid_{rc} \to \mathbf{T}$ returns the time value that the transition need to be completed, which depends on the available resource values.*

A configuration in $\mathscr{T}$ is a pair $\langle s, x\rangle$ where $s \in S$ is a state and x is the tuple of resource assignments.

For each state s, the timeout transition $s \xrightarrow{t} s'$ indicates the time that the system can remain at the state s waiting for an interaction to happen and the state to which the system evolves if no interaction happens on time. An interactive transition represents a set of possible interactions given by the transition instances that result by replacing the data constraint g with a data assignment δ where g holds, and replace resource constraint rc with a resource assignment x at state s which satisfies rc respectively. The time duration for executing the such a transition instance will be $C(x)$. Furthermore, available resource values might be changed throughout the computation and communication. Thus, we posit the existence of a modification function μ, in which $\mu(N, \delta, x) = x'$ has the interpretation that the effect of the interaction δ at port N on resource x is to modify it to x'.

We also assume that the interactive transitions always have a higher priority than timeout transitions, and if a RSTCA $\mathscr{T}$ have both an interactive transition and a timeout transition $s \xrightarrow{t} s'$ at a state s, it means that the system will stay at state s for t time units and evolve to state s' if the interactive transition is not enabled in this duration. But at any time point in $[0, t)$, if it can interact with another system by taking the interactive transition, the interaction will happen immediately and the system will move to another state. This idea is represented in Figure 5, where $t_1 \in [0, t)$ and the behavior of the RSTCA on the left side is in fact like on the right side, i.e., the system remembers what it can do at state s in the duration $[0, t)$ and do it whenever it is possible, and forget it at time t, when it arrives at a new state s'.

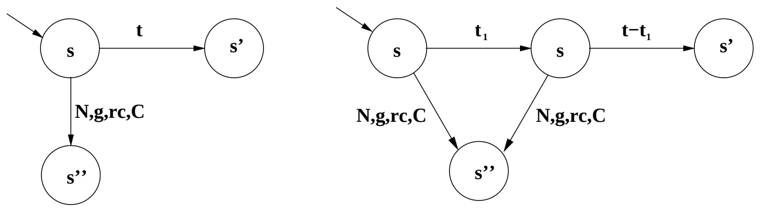

Fig. 5. Timeout and Interactive Transitions

A state is called *terminal* iff it has no outgoing interactive transitions and allows the possibility for unbounded passage of time, i.e., timeout transitions are not allowed in it.

Given a state s and resource assignment x, a transition instance $\langle s, x\rangle \xrightarrow{N,\delta,C(x)} \langle s', x'\rangle$ denotes that if the N-interaction is available, g holds for data assignment δ and the resource assignment x satisfies rc, then the transition happens after $C(x)$ units of time, the new state will be s' and the available resource after the transition is given by x'.

Definition 3. *For a given RSTCA $\mathscr{T}$, suppose s_0 be a state and $c_0 = \langle s_0, x\rangle$ a possible configuration of $\mathscr{T}$. A tuple $(c_0, N, \delta, \bar{t}, t, c)$ is a step of $\mathscr{T}$ for the state s_0 if there exists* *a configuration $c = \langle s, x' \rangle$ and $k \geq 1$ states $s_1, s_2, \cdots, s_k$ such that for all $1 \leq j \leq k$ we have $s_{j-1} \xrightarrow{t_j} s_j$, and there exists a transition $\langle s_{k-1}, x \rangle \xrightarrow{N, \delta, C(x)} \langle s, x' \rangle$ such that $\bar{t} = [\sum_{j=1}^{k-1} t_j, \sum_{j=1}^{k} t_j)$ and $t = C(x)$. We denote by $\texttt{Steps}(\mathscr{T}, s)$ the set of steps of $\mathscr{T}$ for the state s.*

We say that a timed s-run of $\mathscr{T}$ is a (finite or infinite) sequence of successive steps of $\mathscr{T}$ starting in state s. Formally, a timed s-run is a sequence of steps as

$$(c_0, N_0, \delta_0, \bar{t}_0, t_0, c_1), (c_1, N_1, \delta_1, \bar{t}_1, t_1, c_2), \cdots \qquad (1)$$

where $c_0 = \langle s, x \rangle$ for some possible resource assignment x at state s. We denote by $TR(\mathscr{T}, s)$ the set of timed s-runs of $\mathscr{T}$. In addition, we say that the sequence $(N_0, \delta_0, \bar{t}_0), (N_1, \delta_1, \bar{t}_1), \cdots$ is a functional s-run of $\mathscr{T}$ if there is a timed s-run of $\mathscr{T}$ as given in (1).

Intuitively, a step is an interactive transition proceeded by zero or more timeout transitions. The duration $\bar{t}$ in a step $(c_0, N, \delta, \bar{t}, t, c)$ where $c_0 = \langle s_0, x_0 \rangle$ and $c = \langle s, x \rangle$ indicates the possible time values when an interaction could start. Additionally, timed runs include both time values which inform us about possible timeouts (denoted by the intervals $\bar{t}_i$) and the time consumed to execute the interactive transitions in each step of the run.

For the same timed run in a RSTCA, there may exist different instances which are obtained by instantiating every time interval $\bar{t}_i$ by a concrete time value $\hat{t}_i \in \bar{t}_i$.

Definition 4. *Suppose $(c_0, N_0, \delta_0, \bar{t}_0, t_0, c_1), (c_1, N_1, \delta_1, \bar{t}_1, t_1, c_2), \cdots$ is a timed run for a given RSTCA $\mathscr{T}$, the sequence $(c_0, N_0, \delta_0, \hat{t}_0, t_0, c_1), (c_1, N_1, \delta_1, \hat{t}_1, t_1, c_2), \cdots$ is an instanced timed run if for all i, $\hat{t}_i \in \bar{t}_i$. Additionally, we say that the sequence $(N_0, \delta_0, \hat{t}_0), (N_1, \delta_1, \hat{t}_1), \cdots$ is a instanced functional run of $\mathscr{T}$.*

Example 2. Figure 6 shows a resource-sensitive timed variant for the CA of the components and channels of Parrow's Protocol. Here we assume that the internal computation of Sender takes t_S time units, while the exact time of the Medium and Receiver are t_M and t_R respectively. For all the other interactive transitions of the components, we assume that there is no constraints on the resources and the transitions are performed immediately. For the three synchronous channels, we assume that the communication time depends on the bandwidth, i.e., the amount of data that can be transferred over a certain period of time. In this example, the resource constraint for these channels is that the bandwidth w should be more than $10k/s$, and the duration for the interaction over every synchronous channel is $1/w$ time units.

We now explain how to construct a RSTCA via product and hiding. In the following we assume that the common nodes are those where data flow has to be synchronized.

Definition 5. *Let $\mathscr{T}_1 = (S_1, S_0^1, \mathscr{N}_1, R, \longrightarrow_1)$ and $\mathscr{T}_2 = (S_2, S_0^2, \mathscr{N}_2, R, \longrightarrow_2)$ be two RSTCA, their product $\mathscr{T}_1 \bowtie \mathscr{T}_2$ is the RSTCA $\mathscr{T} = (S, S_0, \mathscr{N}, R, \longrightarrow)$ where $S = S_1 \times S_2$, $S_0 = S_0^1 \times S_0^2$, $\mathscr{N} = \mathscr{N}_1 \cup \mathscr{N}_2$ and the interactive transitions are defined by the following synchronization and interleaving rule:*

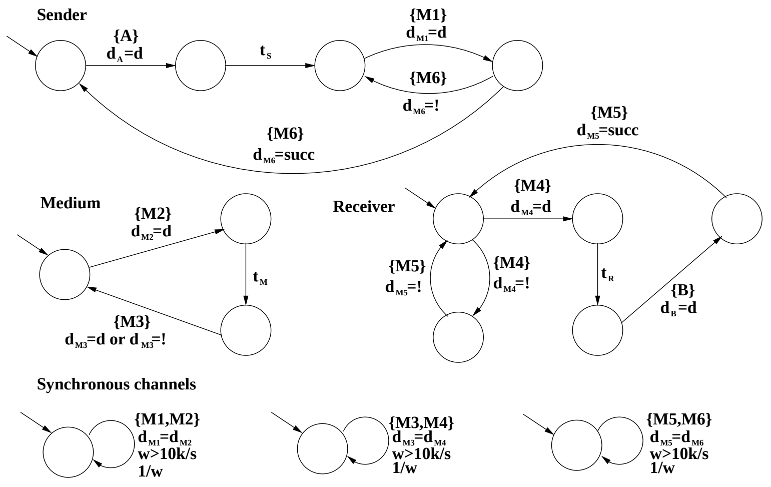

Fig. 6. The RSTCA for the Components and Channels in Fig.3

– *If* $s_1 \xrightarrow{N_1,g_1,rc_1,C_1}_1 s'_1$, $s_2 \xrightarrow{N_2,g_2,rc_2,C_2}_2 s'_2$, $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1 \neq \emptyset$, $g_1 \wedge g_2$ *and* $rc_1 \otimes rc_2$ *are satisfiable, then* $\langle s_1, s_2\rangle \xrightarrow{N_1 \cup N_2, g_1\wedge g_2, rc_1 \otimes rc_2, C_1 \odot C_2} \langle s'_1, s'_2\rangle$.
– *If* $s_1 \xrightarrow{N,g,rc,C}_1 s'_1$, *where* $N \cap \mathcal{N}_2 = \emptyset$ *then* $\langle s_1, s_2\rangle \xrightarrow{N,g,rc,C} \langle s'_1, s_2\rangle$.
– *If* $s_2 \xrightarrow{N,g,rc,C}_2 s'_2$, *where* $N \cap \mathcal{N}_1 = \emptyset$ *then* $\langle s_1, s_2\rangle \xrightarrow{N,g,rc,C} \langle s_1, s'_2\rangle$.

and the following rules for timeout transitions:

$$\frac{s_1 \xrightarrow{t_1} s'_1, s_2 \xrightarrow{t_2} s'_2, t_1 = t_2 = t}{\langle s_1, s_2\rangle \xrightarrow{t} \langle s'_1, s'_2\rangle}$$

$$\frac{s_1 \xrightarrow{t_1} s'_1, s_2 \xrightarrow{t_2} s'_2, t_1 < t_2}{\langle s_1, s_2\rangle \xrightarrow{t_1} \langle s'_1, s_2\rangle,\ s_2 \xrightarrow{t_2 - t_1} s'_2} \qquad \frac{s_1 \xrightarrow{t_1} s'_1, s_2 \xrightarrow{t_2} s'_2, t_1 > t_2}{\langle s_1, s_2\rangle \xrightarrow{t_2} \langle s_1, s'_2\rangle,\ s_1 \xrightarrow{t_1 - t_2} s'_1}$$

The interleaving rules are in the style of labelled transition systems. They formalize the case where no synchronization is required since no common nodes are involved. The synchronization rule expresses the synchronization case which means that both automata have to “agree” on the I/O-operations at their common nodes, while the I/O-operations at their individual nodes is arbitrary. In the synchronization, $C_1 \odot C_2$ depends on the allocation of resources: if resource assignment $r : n$ satisfies $rc_1 \otimes rc_2$, then $C_1 \odot C_2(r : n) = max\{C_1(r : n_1), C_2(r : n_2)\}$ where $n_1 \circ n_2 \sqsubseteq n$ and $r : n_1 \models rc_1$, $r : n_2 \models rc_2$. In general, there may be different allocation strategies that satisfy the condition. Thus, for every such allocation, there is a corresponding value for $C_1 \odot C_2(r : n)$. To decide the exact time value, the concept of scheduler is needed. The details are not of importance here and we will leave the problem of how scheduling can be included in the RSTCA framework as future work.

Note that the definition for this composition operator is an extension of the original product of constraint automata, which has the feature of being neither parallel nor sequential. But it is more powerful than classical sequential and parallel operators and is the source of the expressive power of Reo: not only it does the sequential composition of asynchronous steps, it simultaneously also composes synchronous steps in parallel. More discussions on the reason for this form of composition can be found at [5].

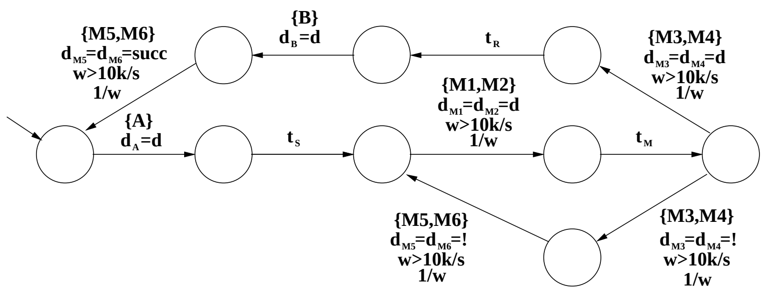

Fig. 7. The Product of the RSTCA for the Components and Channels in Fig.3

Example 3. Consider the RSTCA for the components and channels of Parrow's Protocol as given in Figure 6. Composing them together via the product operator yields the RSTCA as given in Figure 7.

The effect of hiding a node that is internal is that data flow at that node is no longer observable from outside. However, the resource is still consumed and the time being taken should remain the same whether or not the node is hidden.

Definition 6. *The hiding operator takes as input a RSTCA* $\mathscr{T} = (S, S_0, \mathscr{N}, R, \longrightarrow)$ *and a non-empty node-set* $M \subseteq \mathscr{N}$. *The result is a RSTCA* $\mathtt{hide}(\mathscr{T}, M)$ *that behaves as* $\mathscr{T}$ *except that data flow at the nodes* $A \in M$ *is made invisible. Formally,* $\mathtt{hide}(\mathscr{T}, M) = (S, S_0, \mathscr{N} \setminus M, R, \longrightarrow_M)$ *where*

– $s \xrightarrow{\bar{N},\bar{g},rc,C}_M s'$ *iff there exists a transition* $s \xrightarrow{N,g,rc,C} s'$ *such that* $N \setminus M = \bar{N}$ *and* $\bar{g} = \exists M[g]$. *Here* $\exists M[g]$ *stands short for* $\bigvee_{\delta \in DA(M)} g[d_A/\delta.A | A \in M]$, *where* $g[d_A/\delta.A | A \in M]$ *denotes the syntactic replacement of all occurrences of* d_A *in* g *for* $A \in M$ *with* $\delta.A$.

and the timeout transition $s \xrightarrow{t}_M s'$ *iff there exists a timeout transition* $s \xrightarrow{t} s'$.

4 Simulation

Simulation relations were first introduced by Milner in [14] for the purpose of comparing programs, and widely used later to show abstraction and refinement between models and specifications. They provide a sufficient condition for language inclusion that can be established with low complexity, and their precongruence properties are suited for compositional reasoning. In [5] simulation relations for ordinary constraint automata were defined to verify if two automata are language equivalent or the language of one is contained in the language of the other. In this section we propose a notion of *resource-sensitive timed simulation* as a way to guarantee not only the inclusion of languages induced by Reo circuits, but also a higher (or at least equal) performance. For example, we may ask a connector implementation to be always faster than what is required by the specification when the same resource is consumed, where both the specification and the implementation are given as RSTCA.

We first consider the non-timed version of simulation for ordinary constraint automata. As discussed in [5], one constraint automaton $\mathscr{T}$ is simulated by another constraint automaton $\mathscr{A}$ if all the languages that are accepted by $\mathscr{T}$ are also accepted by $\mathscr{A}$. In addition to the non-timed simulation, we require some time conditions to hold. For example, we may hope a constraint automata (the implementation) to be always faster than the time constraints imposed by another (the specification) while no more resource is needed. Additionally, we may require that the implementation always complies with the timeouts established by the specification.

We first introduce the functional simulation relation $\lesssim_f$ where only functional aspects of a system are considered while the performance aspects such as the time needed for an interaction are ignored. However, we should note that the time that a system spends waiting for the environment to react has the possibility to affect the behavior of the system. This is because the time may cause a timeout transition and change the system from one state to another. Therefore, a simulation relation focusing on the functional behavior must also take into account the maximal time the system can stay in each state.

Definition 7. *For a given RSTCA $\mathscr{T} = (S, S_0, \mathscr{N}, R, \longrightarrow)$, the functional simulation is defined as the coarsest binary relation $\lesssim_f \subseteq S \times S$, such that for all $s_1, s_2 \in S$ with $s_1 \lesssim_f s_2$ and all $N \subseteq \mathscr{N}, \delta \in DA(N)$, resource assignment x and $t \in \mathbf{T}$,*

– *if $\langle s_1, x\rangle \xrightarrow{N,\delta,t} \langle s_1', \mu(N,\delta,x)\rangle$, then there exists $s_2' \in S$ and $t' \in \mathbf{T}$, such that $\langle s_2, x\rangle \xrightarrow{N,\delta,t'} \langle s_2', \mu(N,\delta,x)\rangle$ and $s_1' \lesssim_f s_2'$,*
– *if $s_1 \xrightarrow{t} s_1'$, then there exists s_2', such that $s_2 \xrightarrow{t} s_2'$ and $s_1' \lesssim_f s_2'$.*

One RSTCA $\mathscr{T}_2$ functionally simulates another RSTCA $\mathscr{T}_1$ (denoted as $\mathscr{T}_1 \lesssim_f \mathscr{T}_2$) iff every initial state of $\mathscr{T}_1$ is functionally simulated by an initial state of $\mathscr{T}_2$ [1].

Note that the idea underlying Definition 7 is that if one RSTCA $\mathscr{T}_2$ functionally simulates $\mathscr{T}_1$, then $\mathscr{T}_1$ does not allow any behavior that is forbidden in $\mathscr{T}_2$. In the following, we introduce another notion of simulation for RSTCA, which focuses on not only functional behavior, but also resource-related timing properties. The simulation establishes requests over the function from resource values to time values corresponding to the performance of the interactive transitions if they are in the same context of resources.

[1] Here we assume that $\mathscr{T}_1$ and $\mathscr{T}_2$ rely on the same set of names and resources.

Definition 8. *For a given RSTCA $\mathscr{T} = (S, S_0, \mathscr{N}, R, \longrightarrow)$, the (strong) simulation is defined as the coarsest binary relation $\lesssim \subseteq S \times S$, such that for all $s_1, s_2 \in S$ with $s_1 \lesssim s_2$ and all $N \subseteq \mathscr{N}, \delta \in DA(N)$, a resource assignment x and $t \in \mathbf{T}$,*

- *if $\langle s_1, x\rangle \xrightarrow{N,\delta,t} \langle s_1', \mu(N,\delta,x)\rangle$, then there exists $s_2' \in S$ and $t' \in \mathbf{T}$, such that $t \leq t'$ and $\langle s_2, x\rangle \xrightarrow{N,\delta,t'} \langle s_2', \mu(N,\delta,x)\rangle$ and $s_1' \lesssim s_2'$,*
- *if $s_1 \xrightarrow{t} s_1'$, then there exists s_2', such that $s_2 \xrightarrow{t} s_2'$ and $s_1' \lesssim s_2'$.*

One RSTCA $\mathscr{T}_2$ simulates another RSTCA $\mathscr{T}_1$ (denoted as $\mathscr{T}_1 \lesssim \mathscr{T}_2$) iff every initial state of $\mathscr{T}_1$ is simulated by an initial state of $\mathscr{T}_2$.

From Definition 7 and 8, we can easily derive the following result:

Corollary 1. *Any strong simulation is also a functional simulation.*

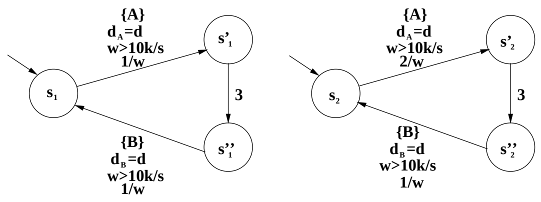

Fig. 8. Simulation

Example 4. We consider the two RSTCA given in Figure 8 for two different implementations of a FIFO channel offered by two providers. Here, state s_1 functionally simulates state s_2 in the same figure, but does not (strongly) simulate it. But on the other direction, we have both $s_1 \lesssim_f s_2$ and $s_1 \lesssim s_2$.

We shall need the familiar property that simulation is a congruence with respect to the product and hiding operators, that is, in our setting, represented by the following theorem:

Theorem 1. *If $\mathscr{T}_1 \lesssim \mathscr{T}_1'$, and $\mathscr{T}_2 \lesssim \mathscr{T}_2'$, then*

(1) $\mathscr{T}_1 \bowtie \mathscr{T}_2 \lesssim \mathscr{T}_1' \bowtie \mathscr{T}_2'$,
(2) $\texttt{hide}(\mathscr{T}_1, M) \lesssim \texttt{hide}(\mathscr{T}_1', M)$.

Proof. The proof is carried out by constructing witnessing simulations. We consider the following relation

$$\mathscr{R} = \{(\langle s_1, s_2\rangle, \langle s_1', s_2'\rangle) : s_1 \lesssim s_1', s_2 \lesssim s_2'\}$$

for (1) and show it to be a simulation.

We only consider the synchronization case. The proof for interleaving and time out transitions are similar. If $\langle\langle s_1, s_2\rangle, x\rangle \xrightarrow{N,\delta,t} \langle\langle \hat{s}_1, \hat{s}_2\rangle, \mu(N,\delta,x)\rangle$ is a transition in $\mathscr{T}_1 \bowtie \mathscr{T}_2$, then according to Definition 5, there exists $N_1, N_2 \subseteq \mathscr{N}$, resource constraints rc_1, rc_2, and some function C_1 and C_2, such that $s_1 \xrightarrow{N_1,g_1,rc_1,C_1}_1 \hat{s}_1$ and $s_2 \xrightarrow{N_2,g_2,rc_2,C_2}_2 \hat{s}_2$ are transitions in $\mathscr{T}_1$ and $\mathscr{T}_2$, respectively, where $N = N_1 \cup N_2$, δ satisfies $g_1 \wedge g_2$, x satisfies $rc_1 \otimes rc_2$, and there exists an allocation of resources such that $C_1 \odot C_2(x) = t$. Since $s_1 \lesssim s_1'$, $s_2 \lesssim s_2'$, it follows that there exist C_1' and C_2', such that $s_1' \xrightarrow{N_1,g_1,rc_1,C_1'}_1 \hat{s}_1'$ and $s_2' \xrightarrow{N_2,g_2,rc_2,C_2'}_2 \hat{s}_2'$ for some $\hat{s}_1'$ and $\hat{s}_2'$ are transitions in $\mathscr{T}_1$ and $\mathscr{T}_2$ respectively, and $\hat{s}_1 \lesssim \hat{s}_1'$, $\hat{s}_2 \lesssim \hat{s}_2'$. Moreover, if we consider the same allocation of resources, then we have $C_1 \leq C_1'$ and $C_2 \leq C_2'$ due to Definition 8. Therefore, $C_1' \odot C_2'(x) \geq t$. Let $t' = C_1' \odot C_2'(x)$, then $\langle\langle s_1', s_2'\rangle, x\rangle \xrightarrow{N,\delta,t'} \langle\langle \hat{s}_1', \hat{s}_2'\rangle, \mu(N,\delta,x)\rangle$ is a transition in $\mathscr{T}_1' \bowtie \mathscr{T}_2'$ and $(\langle s_1, s_2\rangle, \langle s_1', s_2'\rangle) \in \mathscr{R}$.

To prove (2) it suffices to show that given a RSTCA $\mathscr{T} = (S, S_0, \mathscr{N}, R, \longrightarrow)$, any simulation $\lesssim$ for $\mathscr{T}$ is also a simulation for $\texttt{hide}(\mathscr{T}, M)$. By considering Definition 6, we can obtain the result easily.

5 Related Work

Some timed models have been proposed for coordinating services with real-time properties. For example, Arbab *et al.* [2] proposed an operational semantics for Reo in terms of Timed Constraint Automata (TCA) and introduced a temporal logic for specifying and verifying real-time properties or connectors. Orthogonally, a Continuous-Time Constraint Automata (CCA) model was proposed in [6], which integrates the features of continuous-time Markov Chains, and introduces a stochastic variant of the constraint automata model where transitions might have a certain delay according to some probability distribution over a continuous time domain. [8] presented an approach for automatic translation from web service choreography description to timed automata. There are also some work on time extensions of finite state machines [17], timed interface automata [7], etc. These models have the implicit assumption that unbounded resources are available. However, in practice, real-time systems are always restricted by resources. To deal with this problem, several approaches have been proposed to integrate real-time models with the scheduling and resource allocation, which aim to facilitate reasoning about systems sensitive to real-time and resource related properties. For example, [10] proposed a compositional model for reasoning about schedulers which allocate resources to tasks, but it is only suitable to analyse asynchronous systems because it is based on an asynchronous language. [9] defined a hierarchy of resource models to handle the resource allocation and reclamation. However, the models are relatively abstract and not compositional. [13] introduced a timed extension of the extended finite state machine model and a testing methodology, taking into account the temporal issues as we discussed in this paper. However, there is no discussion about compositionality for that model.

As in Continuous-Time Constraint Automata model [6], we also have two types of transitions in the RSTCA model describing interactive transitions and timeout transitions. The difference consists of two aspects: On one hand, our model of timeout transitions is not restricted to Markovian transitions, instead we use time variables to describe the time information, which can be more general and satisfy other kinds of distributions. On the other hand, the time that an interactive transition will take is not just a simple time value describing time passage, but depends on the available resources.

6 Conclusion

In this paper we provided an operational model for reasoning about resource and time information related to component connectors under the assumption that the time duration for interactions depends on the available resources. We defined the RSTCA model for this purpose, together with notions of simulation that are preserved under the composition operators product and hiding. Since these are the only operators needed for compositional construction of networks in the channel-based coordination language Reo, our framework fits well in this context and provides the basis for resource-sensitive performance analysis of component connectors.

In terms of future work, what we would like to do in the next step is the integration of our model and the stochastic timed model as in [6]. Another issue we intend to study is to investigate some resource allocation and consumption strategies, like scheduling [15]. Development of special models and logics for reasoning about resource-sensitive timed features in Reo will also be studied.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. F. Arbab, C. Baier, F. de Boer, and J. Rutten. Models and Temporal Logics for Timed Component Connectors. In Jorge R. Cuellar and Zhiming Liu, editors, *SEFM2004, 2nd International Conference on Software Engineering and Formal Methods*, pages 198–207. IEEE Computer Society, 2004.
3. Farhad Arbab. Reo: A Channel-based Coordination Model for Component Composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.
4. Farhad Arbab, Christel Baier, Jan Rutten, and Marjan Sirjani. Modeling component connectors in reo by constraint automata (extended abstract). In Antonio Brogi, Jean-Marie Jacquet, and Ernesto Pimentel, editors, *Proceedings of FOCLASA 2003, the Foundations of Coordination Languages and Software Architectures*, volume 97 of *ENTCS*, pages 25–46. Elsevier, 2003.
5. Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan Rutten. Modeling component connectors in Reo by constraint automata. *Science of Computer Programming*, 61:75–113, 2006.
6. Christel Baier and Verena Wolf. Stochastic Reasoning About Channel-Based Component Connectors. In P. Ciancarini and H. Wiklicky, editor, *COORDINATION 2006*, volume 4038 of *LNCS*, pages 1–15. Springer-Verlag, 2006.
7. L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proceedings of EMSOFT*, volume 2491 of *LNCS*, pages 108–122. Springer, 2002.
8. Gregorio Diaz, Juan-José Pardo, María-Emilia Cambronero, Valentín Valero, and Fernando Cuartero. Automatic Translation of WS-CDL Choreographies to Timed Automata. In Mario Bravetti and Leïla Kloul and Gianluigi Zavattaro, editor, *EPEW 2005 and WS-FM 2005*, volume 3670 of *LNCS*, pages 230–242. Springer, 2005.

9. Naiyong Jin and Jifeng He. Resource Semantic Models for Programming Languages. Technical Report 277, UNU/IIST, April 2003.
10. G. Lowe. Scheduling-oriented models for real-time systems. *The Computer Journal*, 38:443–456, 1995.
11. M. P. Papazoglou and D. Georgakopoulos. Service Oriented Computing. *Comm. ACM*, 46(10):25–28, 2003.
12. Daniel A. Menascé. Composing Web Services: A QoS View. *IEEE Internet Computing*, 8(6):88–90, 2004.
13. Mercedes G. Merayo, Manuel Núñez, and Ismael Rodríguez. Extending efsms to specify and test timed systems with action durations and timeouts. In E. Najm et al., editor, *FORTE 2006*, volume 4229 of *LNCS*, pages 372–387. Springer, 2006.
14. Robin Milner. An algebraic definition of simulation between programs. In David C. Cooper, editor, *Proceedings of the 2nd International Joint Conference on Artifiial Intelligence, London, UK*. William Kaufmann, British Computer Society, 1971.
15. M. R. Mousavi, M. A. Reniers, T. Basten, and M. R. V. Chaudron. PARS: A Process Algebra with Resources and Schedulers. In *1st Int. Workshop on Formal Modeling and Analysis of Timed Systems, FORMATS'03*, volume 2791 of *LNCS*, pages 134–150. Springer, 2003.
16. Manuel Núñez and Ismael Rodríguez. Conformance testing relations for timed systems. In *5th Int. Workshop on Formal Approaches to Software Testing (FATES 2005)*, volume 3997 of *LNCS*, pages 103–117. Springer, 2006.
17. J. C. Park and R. E. Miller. Synthesizing protocol specifications from service specifications in timed extended finite state machines. In *17th IEEE International Conference on Distributed Computing Systems, ICDCS'97*, pages 253–260. IEEE Computer Society, 1997.
18. J. Parrow. *Fairness Properties in Process Algebra*. PhD thesis, Uppsala University, Sweden, 1985.
19. David Pym and Chris Tofts. A calculus and logic of resources and processes. *Formal Aspects of Computing*, 18:495–517, 2006.