Encoding PAMR into (Timed) EFSMs[⋆]

Manuel Núñez and Ismael Rodríguez

Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
{mn,isrodrig}@sip.ucm.es

Abstract. In this paper we show how the formal framework introduced in PAMR (Process Algebra for the Management of Resources) can be included into a notion of Extended Finite State Machines (in short EFSM). First, we give the definition of process. Following the lines of PAMR, a process consists not only of information about its behavior but also of information about the preference for resources. This information will be encoded into a model based on EFSMs. In contrast with the original definition of PAMR, a notion of time is included in our processes, that is, transitions take time to be performed. Systems are defined as the composition of several processes. We present different implementation relations, depending on the interpretation of time, and we relate them. Finally, we study how tests cases are defined and applied to implementations.

1 Introduction

There exists a growing interest in including microeconomic concepts into different areas of computer science. For example, several systems related to e-commerce use a notion of *preference* or, equivalently, *utility function* (see e.g. [23,3,8]). These notions have been also considered in the field of networks to define algorithms dealing with resource allocation (e.g. [17,27,12]). However, most of these works are restricted to solve a concrete problem, that is, they do not provide a general framework to be applied to different fields.

PAMR [19] is a formal language very suitable to specify systems which depend on the resources that they own. Sometimes, in order to formally describe a system, it is useful to specify information about the available resources for the different components of the system. Let us introduce the following simple running example. Consider a system consisting in the parallel execution of n subsystems $(P_1 \ldots P_n)$ and m different kinds of resources that they may use (let us suppose that the total quantity of the resource i is equal to x_i). The performance of these subsystems depends on these resources (for example, the portion of memory used by each subsystem, time quantum of CPU, time quantum of access to the bus, etc). Each subsystem P_j has an initial distribution of resources $x_1^j \ldots x_m^j$, that

⋆ Research supported in part by the CICYT project TIC 2000-0701-C02-01. This research was carried out while the first author was visiting the GET-INT. He would like to thank Ana Cavalli for helpful discussions on testing and for her financial support under the *Platonis* project.

is, in the beginning P_j owns x_i^j units of the resource i. Given the fact that the quantity of resources that subsystems own cannot be bigger than the total amount, we work under the constraint: $\forall\ 1 \leq i \leq m :\ \sum_j x_i^j \leq x_i$. Finally, subsystems have a *preference* on how they *like* resources. For example, suppose that P_{j_1} runs at the same speed if we replace one unit of the resource i_1 by four units of the resource i_2, while P_{j_2} runs at the same speed if one unit of the resource i_1 is replaced by two units of the resource i_2. In particular, P_{j_1} will perform better if we replace three of its units of i_2 by one additional unit of i_1.

In PAMR a process does not only take into account its activity (that is, which actions can be performed at a given point of time) but it also considers which resources can be used during its execution. Besides, processes may *exchange* resources with other processes. For instance, in our running example, if P_{j_1} gives to P_{j_2} three units of i_2 and receives from P_{j_2} one unit of i_1, then both subsystems run faster. So, PAMR provides mechanisms to, starting with an initial distribution of resources, accomplish a better performance of the system. Processes are able to exchange resources but *harmful* exchanges are forbidden. For instance, it is not allowed that P_{j_2} exchanges three units of i_2 by one unit of i_1 with P_{j_1} because both subsystems get worse, and thus the whole system deteriorates.

The formal language PAMR has been already applied to very different frameworks: A variant of e-commerce [15], the management of software projects [20], and the definition of (part of) the behavior of agents in a cooperative learning environment [16]. However, the current formulation of PAMR lacks two characteristics that could strongly increase its application to the specification and analysis of communication protocols. First, there is no notion of input/output actions. Second, no notion of time appears in the language. In this paper we deal with these two limitations by considering a model where PAMR fits. We will consider the well-known formalism of *Extended Finite States Machines* [13] (in short EFSM). Our aim is to show how the concepts underlying the conception of PAMR can be encoded into a kind of EFSM. In order to include time aspects, we consider that some actions take time to be performed. This time will mainly depend on the available resources, that is, in order to perform an action the bigger amount of suitable resources a process may use the faster this action will be performed. In PAMR (and in the mechanism presented in this paper) processes have a *necessity function*. Given an action, this function returns a positive real value if the process has enough resources to perform this action; otherwise, the value ∞ is returned. We will consider that this value denotes the time that it takes to perform the corresponding action (with respect to the available resources).

In addition to the presentation of our formalism we study *conformance testing* relations in our framework. Conformance is the term used by system analyzers to describe the situation in which an implementation is adequate with respect to a given specification. In order to properly define this notion, and thus to have the formal basis for the process of testing, there has been a considerable effort, that in particular has been the seed for the joint ISO/ITU–T working group on "Formal Methods in Conformance Testing" (in [6] a summary of the work carried out by the group is presented). In order to formalize the notion of conformance two are the most extended methods: by means of an *implementation relation* or by *requirements*. We will concentrate on the first approach. An implementation relation relates implementations from a given set Imp with specifications from another set Spec. Our study considers relations based on conf [5] and more precisely on the update of this relation to deal with inputs and outputs: ioco [25, 26]. In order to cope with time, we will not take into account only that a system may perform a given action, but we will also record the amount of time that the system needs to do so (according to the resources that it owns). Unfortunately, conformance testing relations for timed systems have not been yet extensively studied. We propose three conformance relations according to the interpretation of *good* implementation for a given specification. Regarding our relations, time aspects add some extra complexity. For example, even though an implementation I had the same traces as an specification S, we should not consider that I conforms to S if the implementation is always *slower* than the specification. Moreover, it can be the case that a system performs a sequence of actions for different times. These facts motivate the definition of several conformance relations. For example, it can be said that an implementation conforms to a specification if the implementation is always *faster*, or if the implementation is at least as *fast* as the worst case of the specification. We think that the relations that we introduce in this paper can be useful for the study of conformance for other models of timed systems. For example, the definitions can be easily adapted to timed automata [1]. Other definitions of timed I/O automata (e.g. [10,24]) are restricted to deterministic (regarding actions) behavior. In this case, some of our relations will be equivalent among them (i.e. they will relate the same automaton).

In terms of related work, there are models to specify systems sharing resources (e.g. [4]), but in this case resources are just accessed, not traded; this access induces some delays in the behavior of processes. Our proposal is somehow related to the *ODP trading function* [11]. Nevertheless, in our case a process only uses (and nobody else can use them until they are exchanged) the resources that it owns. Finally, management of resources appears in fields like operating systems or concurrent programming. Resources are usually owned by a *mediator* which allows processes to use them. Regarding conformance testing for timed systems, some proposals have already appeared in the literature (e.g. [18,7,10, 24]). Our proposal differs from these ones in several points, mainly because the treatment of time is different. We do not have a notion of clock(s) together with time constraints; we associate time to the execution of actions (the time that it takes for a system to perform and action).

The rest of the paper is organized as follows. In Section 2 we define our language in terms of processes and systems. In Section 3 we study implementation relations for our framework where we make an interpretation of the behavior with respect to time. We relate these implementation relations. We also introduce a new relation that classifies policies for exchanging resources with respect to a specification. In Section 4 we show how test cases are defined and describe how to apply them to implementations. Finally, in Section 5 we present our conclusions and some directions for further research.

2 PAMR Processes as (Timed) EFSMs

In this section we show how the concepts behind the definition of PAMR can be encoded into an extended notion of EFSM. The main differences with respect to usual EFSMs consist in the addition of *time* and the existence of *symbolic* transitions denoting exchanges of resources. Next we introduce the definition of Timed EFSM. We suppose that the number of different variables is equal to m.

Definition 1. A *Timed Extended Finite State Machine*, in the following TEFSM, is a tuple $M = (S, I, O, Tr, s_{in}, \bar{y})$ where S is a finite set of states, I is the set of input actions, O is the set of output actions, Tr is the set of transitions, s_{in} is the initial state, and $\bar{y} \in \mathbb{R}_+^m$ is a tuple of variables.

Each transition $t \in Tr$ is a tuple $t = (s, s', i, o, Q, Z, C)$ where $s, s' \in S$ are the initial and final states of the transition, $i \in I$ and $o \in O$ are the input and output actions, respectively, associated with the transition, $Q : \mathbb{R}_+^m \longrightarrow \texttt{Bool}$ is a predicate on the set of variables, $Z : \mathbb{R}_+^m \longrightarrow \mathbb{R}_+^m$ is a transformation over the current variables, and $C : \mathbb{R}_+^m \longrightarrow \mathbb{R}_+ \cup \{\infty\}$ is the time that the transition needs to be completed according to the available resources.

A *configuration* in M is a pair $(s, \bar{x})$ where $s \in S$ is the current state and $\bar{x}$ is the current tuple of variable values.

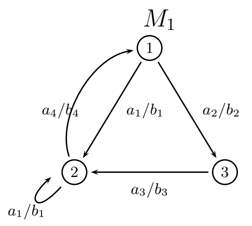
We say that $tr = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ is a (timed) *trace* of M if there exist transitions $t_1, \ldots, t_r \in Tr$ such that $t_1 = (s, s_1, i_1, o_1, Q_1, Z_1, C_1)$,. . ., $t_r = (s_{r-1}, s', i_r, o_r, Q_r, Z_r, C_r)$, the predicate Q is defined such that it holds $Q(\bar{x}) = (Q_1(\bar{x}) \ \wedge \ Q_2(Z_1(\bar{x})) \ \wedge \ \ldots \ \wedge \ Q_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots)))$, the transformation Z is defined as $Z(\bar{x}) = Z_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots))$, and C is defined as $C(\bar{x}) = C_1(\bar{x}) + C_2(Z_1(\bar{x})) + \cdots + C_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots))$.

Let $I' \subseteq I$. The *first occurrence* of I' in the trace tr, denoted by $\texttt{First}(tr, I')$, is defined as the first input $i_j \in I'$, with $1 \leq j \leq r$, appearing in the trace. □

Intuitively, for a configuration $(s, \bar{x})$, a transition $t = (s, s', i, o, Q, Z, C)$ indicates that if the machine is in state s, receives the input i, and the predicate Q holds for $\bar{x}$, then after $C(\bar{x})$ units of time (assuming $C(\bar{x}) \neq \infty$) the machine emits the output o and the values of the variables are transformed according to Z. Traces are defined as a sequence of transitions. In this case, the predicate, the transformation function, and the time associated with the trace are computed from the corresponding to each transition belonging to the sequence. We suppose that addition in real numbers is extended in the following way: $\infty + r = \infty$. Finally, the notion of first occurrence will be used when deleting internal actions from the composition of machines.

Example 1. In Figure 1 we present two TEFSMs. For example, let us suppose that the initial value of variables is $\bar{x} = (2, 0, 0, 2)$ and that the initial state of M_1 is the state labelled by 1. Then, the transition t_{12} can be performed, it will take time $\frac{1}{2}$, and after that, the value of the variables will be given by $(3, 0, 0, 1)$. □

As we said in the introduction, we separate between the behavior of the process and the management of resources. We introduce three functions that are



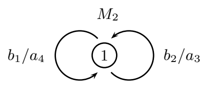
$I = \{a_1, a_2, a_3, a_4\}, O = \{b_1, b_2, b_3, b_4\}$

$$\begin{aligned} t_{12} &= (s_1, s_2, a_1, b_1, Q_1, Z_1, C_1) \\ t_{13} &= (s_1, s_3, a_2, b_2, Q_2, Z_2, C_2) \\ t_{32} &= (s_2, s_3, a_3, b_3, Q_3, Z_3, C_3) \\ t_{21} &= (s_2, s_1, a_4, b_4, Q_4, Z_4, C_4) \\ t_{22} &= (s_2, s_2, a_1, b_1, Q_5, Z_5, C_5) \end{aligned}$$

$$Z_i(\bar{x}) = \bar{x} + \begin{cases} (1,0,0,-1) & \text{if } i \in \{1,2,5\} \\ (0,1,-1,0) & \text{if } i \in \{3,4\} \end{cases}$$

$$Q_i(\bar{x}) \equiv Z_i(\bar{x}) \geq \bar{0} \ \wedge \ \begin{cases} x_i > 0 & \text{if } i \in \{1,2,3,4\} \\ x_1 > 0 & \text{if } i = 5 \end{cases}$$

$$C_i(\bar{x}) = \begin{cases} \frac{1}{x_i} & \text{if } i \in \{1,2,3,4\} \ \wedge \ x_i \neq 0 \\ \frac{1}{x_1} & \text{if } i = 5 \ \wedge \ x_1 \neq 0 \\ \infty & \text{otherwise} \end{cases}$$



$I = \{b_1, b_2\}, O = \{a_3, a_4\}$

$$\begin{aligned} t_1 &= (s_1, s_1, b_1, a_4, Q_1, Z_1, C_1) \\ t_2 &= (s_1, s_1, b_2, a_3, Q_2, Z_2, C_2) \end{aligned}$$

$$Z_i(\bar{x}) = \bar{x} + \begin{cases} (1,1,-1,-1) & \text{if } i = 1 \\ (1,0,0,0) & \text{if } i = 2 \end{cases}$$

$$Q_i(\bar{x}) \equiv Z_i(\bar{x}) \geq \bar{0} \ \wedge \ \begin{cases} x_1 > 0 \ \wedge \ x_2 > 0 & \text{if } i = 1 \\ x_3 > 0 & \text{if } i = 2 \end{cases}$$

$$C_i(\bar{x}) = \begin{cases} \frac{1}{x_i \cdot x_2} & \text{if } i = 1 \ \wedge \ x_1 \neq 0 \ \wedge \ x_2 \neq 0 \\ \frac{1}{x_3} & \text{if } i = 2 \\ \infty & \text{otherwise} \end{cases}$$

We suppose that $\bar{x} \in \mathbb{R}^4_+$. We denote by x_i the *i-th* component of $\bar{x}$.

Fig. 1. Examples of TEFSM.

responsible for controlling all the details related to the resources that a machine may use at a given state. Consider $M = (S, I, O, Tr, s_{in}, \bar{y})$.

- *Utility functions.* A function $u : S \times \mathbb{R}^m_+ \longrightarrow \mathbb{R}^+$ such that u is monotonic[1] and non-decreasing in the second argument. A utility function makes, for each state, a classification of the different basket of resources. For instance, $u(s, \bar{x}_1) < u(s, \bar{x}_2)$ means that if the corresponding machine is currently in the state s then it prefers to have the basket of resources $\bar{x}_2$ to the basket $\bar{x}_1$. Note that this function depends on the state as *preferences* are not the same in all the states of the machine (they are usually strongly influenced by the actions that can be performed).
- *Necessity functions.* A function $n : O \times \mathbb{R}^m_+ \longrightarrow \mathbb{R}^+ \cup \{\infty\}$ such that n is monotonic and non-increasing in the second argument. The purpose of this function is to decide whether the machine owns enough resources to perform the corresponding output action (i.e., it returns a value different from ∞). We consider that only output actions need resources to be performed because input actions are *passive* entities in the sense that they are performed once a suitable *signal* is received. In this case, the machine performing the corresponding output action will use resources to perform the communication. Time will be introduced into our processes by means of this function.
- *Consumption functions.* A function $c : O \times \mathbb{R}^m_+ \rightarrow \mathbb{R}^m_+$. They indicate how resources are created/consumed after performing an action. For example,

[1] In the following we consider that $\mathbb{R}^m_+$ is ordered by the relation $(x_1, \ldots, x_m) \leq (y_1, \ldots, y_m)$ iff $x_i \leq y_i$, for any $1 \leq i \leq m$.

$c(o, \bar{x}) = \bar{y}$ means that if the machine currently owns the basket of resources $\bar{x}$ then the amount of resources owned after performing o is given by $\bar{y}$.

A *process* is a `TEFSM` where transitions adequately reflect the corresponding associated functions. Intuitively, predicates in transitions will indicate that the machine has enough resources to perform the corresponding action and that after creating/consuming resources no *debts* are generated; transformation functions will record the new basket of resources after the transition is performed; finally, the time associated with transitions is given by the necessity function.

Definition 2. Let $M = (S, I, O, Tr, s_{in}, \bar{y})$ be a `TEFSM` and u, n, c be utility, necessity and consumption functions, respectively. We say that $P = (M, u, n, c)$ is a *process* if for each $t = (s, s', i, o, Q, Z, C) \in Tr$ the predicate Q, and the functions Z and C fulfill $Q(\bar{x}) \equiv (n(o, \bar{x}) < \infty \wedge Z(\bar{x}) \geq \bar{0})$, $Z(\bar{x}) = c(o, \bar{x})$, and $C(\bar{x}) = n(o, \bar{x})$. □

Example 2. Consider the machine M_1 presented in Example 1. Let n and c be two functions defined, respectively, as

$$n(b_j, \bar{x}) = \begin{cases} \frac{1}{x_j} & \text{if } x_j > 0 \\ \infty & \text{otherwise} \end{cases} \qquad c(b_j, \bar{x}) = \bar{x} + \begin{cases} (1, 0, 0, -1) & \text{if } j \in \{1, 2\} \\ (0, 1, -1, 0) & \text{if } j \in \{3, 4\} \end{cases}$$

If u_1 is a utility function for M_1 then we have that (M_1, u_1, n, c) is a process. Accordingly, if we extend n and c in the following way

$$n(a_j, \bar{x}) = \begin{cases} \frac{1}{x_1 \cdot x_2} & \text{if } j = 4 \wedge x_1 > 0 \wedge x_2 > 0 \\ \frac{1}{x_3} & \text{if } j = 3 \wedge x_3 > 0 \\ \infty & \text{otherwise} \end{cases} \qquad c(a_j, \bar{x}) = \bar{x} + \begin{cases} (1, 1, -1, -1) & \text{if } j = 4 \\ (1, 0, 0, 0) & \text{if } j = 3 \end{cases}$$

and u_2 is a utility function for M_2 then (M_2, u_2, n, c) is also a process. □

In this paper we do not consider *usual* variables. That is, variables are always associated with resources.[2] Let us remark that, for each transition, Z, Q, and C will be applied to the current amount of resources (they will be indicated by the current configuration). Let us also note that even though the utility function does not explicitly appear in the definition of transitions, it will be taken into account when processes exchange resources.

We will define systems by composing several processes. Processes will communicate among them in two ways. First, they will jointly perform actions by sending an output action from one process that will be received (as an input) by another one. Besides, in order to improve their performance, processes will exchange resources. These exchanges will be guided by a *policy*. We consider the

[2] In order to cope with this restriction it is enough to consider that the whole set of variables ranges over $A \times \mathbb{R}_+^m$, being A the type of the variables keeping track of data. Note that A may possibly be a cartesian product of different sorts. Predicates and functions associated with transitions should be adapted accordingly. For example, for a transition $t = (s, s', i, o, Q, Z, C)$ we would have $Q = Q_1 \wedge Q_2$ where Q_1 is a predicate over A while Q_2 is a predicate over $\mathbb{R}_+^m$.

two policies already introduced in [19].[3] Under a *preserving utility* policy exchange of resources are allowed only if, after the exchange, at least one process improves and no process gets worse. Intuitively, processes are the owners of the resources and they will not give up them if they do not receive a *compensation*. Under a *maximizing utility* policy exchanges are allowed only if the whole system improves (even if some of the components deteriorate). In order to measure when a process/system improves we take into account the corresponding utility functions. Exchanges are denoted by a matrix $\mathcal{E} \in (\mathbb{R}^m_+)^{n \times n}$. An element $\mathcal{E}_{i_1 i_2} = (x_1, \ldots, x_i, \ldots, x_m)$ indicates that the process i_1 gives to the process i_2 a total of x_i units of the *i-th* resource. So, the total tuple of resources given by the process i_1 to the rest of processes is $\sum_k \mathcal{E}_{i_1 k}$ while it receives from other processes the tuple $\sum_k \mathcal{E}_{k i_1}$.

Definition 3. Let $P_1, \ldots, P_n$ be processes such that $P_i = (M_i, u_i, n, c)$, for some utility functions u_i, and some necessity and consumption functions n and c, respectively. For each i, let $c_i = (s_i, \bar{x}_i)$ be configurations over P_i.

An *exchange matrix* (usually denoted by $\mathcal{E}, \mathcal{E}', \ldots$) is an $n \times n$ matrix where the components belong to $\mathbb{R}^m_+$.

We say that the exchange indicated by $\mathcal{E}$ is allowed under the *preserving policy* (resp. *maximizing policy*) for the configurations $c_1, \ldots, c_n$, denoted by $\texttt{allowed}_{pres}(c_1, \ldots, c_n, \mathcal{E})$ (resp. $\texttt{allowed}_{max}(c_1, \ldots, c_n, \mathcal{E})$), if we have that $\forall\, 1 \leq i \leq n : \bar{x}_i - \sum_k \mathcal{E}_{ik} \geq \bar{0}$ and the following conditions hold:

- *Preserving policy*:
  - $\exists\, 1 \leq r \leq n : u_r(s_r, \bar{x}_r + \sum_k \mathcal{E}_{kr} - \sum_k \mathcal{E}_{rk}) > u_r(s_r, \bar{x}_r)$, and
  - $\forall\, 1 \leq i \leq n : u_i(s_i, \bar{x}_i + \sum_k \mathcal{E}_{ki} - \sum_k \mathcal{E}_{ik}) \geq u_i(s_i, \bar{x}_i)$
- *Maximizing policy*:
  - $\sum_i u_i(s_i, \bar{x}_i) < \sum_i u_i(s_i, \bar{x}_i + \sum_k \mathcal{E}_{ki} - \sum_k \mathcal{E}_{ik})$ □

The three conditions defining the preserving policy indicate, respectively, that no process gives resources that it does not own, that (at least) one process improves after the exchange, and that no one deteriorates. The second condition of the maximizing policy expresses that the total utility of the processes (measured as the addition of the corresponding utilities) increases after the exchange.

We compose processes to define systems. Some of the actions of the processes will be hidden indicating that they are not visible, that is, they can be neither controlled nor observed. In order to facilitate the understanding, we have not defined a *compressed* version of systems where internal communications are omitted. So, we generate the whole graph (including both internal and external actions) and then we *delete* internal communications (getting what we call a *simplified system*). In addition to the usual input/output transitions, systems have a kind of *symbolic* transitions indicating exchange of resources. These transitions are parameterized by the allowed exchange matrixes for the current configuration. In this case, transition will be labelled with *special* input and output

[3] The choice of a *good* policy is not a trivial task, as this problem is related with the social welfare aggregator problem. Arrow's *impossibility theorem* [2] shows that there does not exist such an aggregator fulfilling a certain set of *desirable* properties.

symbols (the symbol $*$ and the corresponding exchange matrix, respectively). A system will be able to perform a transition labelled by an external input only when no more exchanges are allowed. That is, processes exchange resources until they reach a *good* distribution. In this case, transitions take into account the current configurations of the processes.

Definition 4. Let $P_1, \ldots, P_n$ be processes, where $P_i = (M_i, u_i, n, c)$, for some utility functions u_i, and some necessity and consumption functions n and c, respectively; each M_i is given by $M_i = (S_i, I_i, O_i, Tr_i, s_{in_i}, \bar{y}_i)$. Let $I \subseteq \bigcup_i I_i$ and $O \subseteq \bigcup_i O_i$, such that $I \cap O = \emptyset$. The *system* created by the composition of $P_1, \ldots, P_n$ under the exchange policy Pol with respect to the actions sets I and O, denoted in short by $\mathtt{Sys}(P_1, \ldots, P_n, I, O, Pol)$, is defined as the TEFSM $(S, I \cup \{*\}, O \cup (\mathbb{R}_+^m)^{n \times n}, Tr, s_{in}, \bar{y})$ where:

- The initial state s_{in} is defined as $s_{in} = (s_{in_1}, \ldots, s_{in_n})$.
- The initial tuple of (tuples of) variable values is $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_n)$.
- $S = S_1 \times \cdots \times S_n$. Actually, it is enough to consider those states reachable from s_{in} by sequences of transitions belonging to Tr.
- Let $s \in S$ with $s = (s_1, \ldots, s_n)$. We have $(s, s, *, \mathcal{E}, Q, Z, C) \in Tr$, where Q and Z are defined as $Q(\bar{x}_1, \ldots, \bar{x}_n) \equiv \mathtt{allowed}_{Pol}((s_1, \bar{x}_1), \ldots, (s_n, \bar{x}_n), \mathcal{E})$ and $Z(\bar{x}_1, \ldots, \bar{x}_n) = (\bar{x}_1 + \bar{z}_1, \ldots, \bar{x}_n + \bar{z}_n)$, where $\bar{z}_i = \sum_k \mathcal{E}_{ki} - \sum_k \mathcal{E}_{ik}$. Besides, the time function C is defined as $C(\bar{x}_1, \ldots, \bar{x}_n) = 0$.
- Let $s = (s_1, \ldots, s_j, \ldots, s_n)$ and $s' = (s_1, \ldots, s'_j, \ldots, s_n)$ be two states. We have $(s_j, s'_j, i, o, Q_j, Z_j, C_j) \in Tr_j$ implies $(s, s', i, o, Q, Z, C) \in Tr$, where $Q(\bar{x}_1, \ldots, \bar{x}_n) \equiv Q_j(\bar{x}_j) \wedge (i \in I \Rightarrow \nexists (s, s, *, \mathcal{E}, Q', Z', C') \in Tr : Q'(\bar{x}_1, \ldots, \bar{x}_n))$, $Z(\bar{x}_1, \ldots, \bar{x}_n) = (\bar{x}_1, \ldots, Z_j(\bar{x}_j), \ldots, \bar{x}_n)$, and $C(\bar{x}_1, \ldots, \bar{x}_n) = C_j(\bar{x}_j)$.

Let $tr = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ be a trace for the previous TEFSM. We say that tr is a *chained trace* if $o_r \in O$ and there exists $1 \leq k \leq r$ such that $i_k \in I$, for any $1 \leq j \leq k-1$ we have $i_j = *$ and $o_j \in (\mathbb{R}_+^m)^{n \times n}$, and for any $k+1 \leq l \leq r$ we have $i_l \notin I \cup O$ and $i_l = o_{l-1}$. □

Let us note that actions not belonging to $I \cup O$ are considered as internal. For the sake of simplicity, we consider that exchanges do not consume time.[4] A chained trace consists of a sequence of exchanges, an external input action, a consecutive sequence of paired output/input actions, and finally an external output action. Let us remind that the predicate Q and the functions Z and C associated with chained traces take into account the exchanges performed before the first visible input appears in the trace (see Definition 1).

In order to abstract internal computations, systems are transformed into *simplified systems*. The idea is that transitions of a simplified systems are those chained traces belonging to the original system.

[4] Time can be associated with exchanges of resources by replacing the assignment $C(\bar{x}) = 0$ in the fourth clause of the previous definition by $C(\bar{x}) = e(\bar{x})$, where e is a function computing time with respect to the amount of exchanged resources.

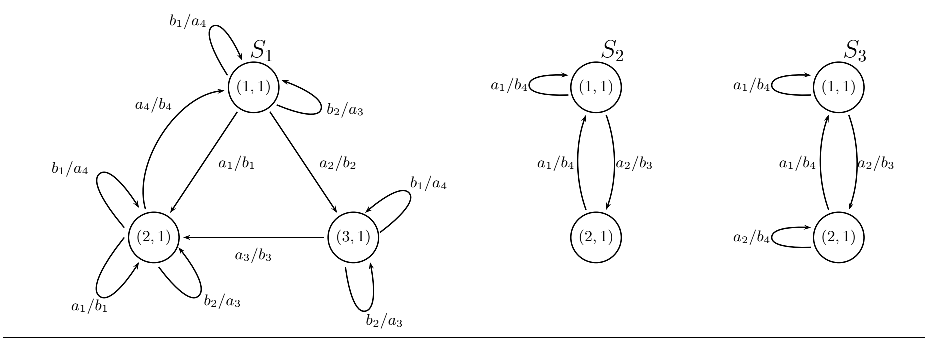


Fig. 2. Examples of Systems and Simplified Systems.

Example 3. In Figure 2 (left) we present the composition of the processes defined in Example 2. We have omitted transitions related to exchanges. Let us consider that their utility functions are respectively defined as:

$$u_1(s_j,\bar{x}) = \begin{cases} x_1 + x_2 & \text{if } j = 1 \\ x_1 + x_4 & \text{if } j = 2 \\ x_3 & \text{if } j = 3 \end{cases} \qquad u_2(s_1,\bar{x}) = x_1 \cdot x_2 + x_3$$

Suppose that the initial distributions are $\bar{x}_1 = (1,1,2,1)$ and $\bar{x}_2 = (5,1,0,0)$, respectively, and that we are in the initial state (state $(1,1)$ of the composition). A possible exchange under the maximizing policy would transform these distributions into $\bar{x}_1 = (0,0,0,1)$ and $\bar{x}_2 = (6,2,2,0)$. Note that this exchange is not allowed under a preserving policy. In the latter case we could have, for example, an exchange leading to $\bar{x}_1 = (3,0,0,1)$ and $\bar{x}_2 = (3,2,2,0)$.

We suppose that the sets of actions I and O are respectively defined as $I = \{a_1, a_2\}$ and $O = \{b_3, b_4\}$. These sets are more relevant when defining the associated simplified system. In Figure 2 (center) we present the simplified system corresponding to the previous system. □

Definition 5. Let $Comp = (S, I \cup \{*\}, O \cup (\mathbb{R}_+^m)^{n\times n}, Tr, s_{in}, \bar{y})$ be a system. We say that $M' = (S', I, O, Tr', s_{in}, \bar{y})$ is the *simplified system* associated with *Comp*, denoted by $\texttt{Simp}(Comp)$, if S' and Tr' fulfill the recursive definition:

- $s_{in} \in S'$, and
- If $s \in S'$ and $tr = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ is a chained trace of *Comp* then $s' \in S'$ and $(s, s', \texttt{First}(tr, I), o_r, Q, Z, C) \in Tr'$.

We say that $i_1/o_1, \ldots, i_r/o_r$ is a *non-timed evolution*, or simply *evolution*, of M' if there exists a trace $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ of M' such that $Q(\bar{y})$ holds. Given a simplified system *Scomp*, we denote by $\texttt{NTEvol}(Scomp)$ the set of non-timed evolutions of *Scomp*.

We say that the pair $((i_1/o_1, \ldots, i_r/o_r), t)$ is a *timed evolution* of M' if there exists a trace $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ of M' such that $Q(\bar{y})$ holds and $t = C(\bar{y})$. Given a simplified system *Scomp*, we denote by $\texttt{TEvol}(Scomp)$ the set of timed evolutions of *Scomp*. □

Let us remind that $\texttt{First}(tr, I)$ denotes the first (an unique) occurrence of an external input in the chained trace tr. As we said before, a chained trace is converted into a single transition. Then, an evolution is a trace from the initial state of the simplified system. Let us note that all the actions appearing in evolutions are visible (as both internal actions and exchanges are removed by considering transitions formed from chained traces). We distinguish between timed and non-timed evolutions (where we simply *forget* the associated time) because it simplifies the forthcoming definitions of implementation relations.

3 (Timed) Implementation Relations

In this section we introduce our implementation relations. All of them follow the same pattern: An implementation I *conforms* to a specification S if for any possible evolution of S the outputs that the implementation I may perform after a given input are a subset of those for the specification. This pattern is borrowed from `ioco` [25,26] but we do not consider *quiescent* states (that is, states where no external outputs are available). In addition to the non-timed conformance of the implementation, we require some time conditions to be hold (this is a major difference with respect to `ioco` where time is not considered). For example, we may ask an implementation to be always faster than the time constraints imposed by the specification. The different considerations of time produce that there is not a unique way to define an implementation relation.

Next, we formally define the sets of specifications and implementations: `Spec` and `Imp`. A specification is the composition of a set of processes, that is a system, or equivalently its associated simplified system. Regarding implementations, we have to determine what is visible. We consider that exchanges of resources are *autonomous* as they only concern internal planning of the corresponding processes. Even in this case, these exchanges strongly influence the behavior of implementations as they may allow/disallow some actions to be performed. Let us remind that a necessary condition for an action to be performed is that the corresponding process owns enough resources. Another approach could be to consider that exchanges can be either observed or controlled (by adding adequate points of control and/or observation into the implementation). However, we think that such framework could be more appropriate for a kind of *testing in context* (see e.g. [21,14]) where we desire to test one component of a system (in this case a process) assuming that the rest of the system is correct. We will comment on this approach in the conclusions of this paper. Besides, we assume that all the input actions are always enabled in any state of the implementation (a similar assumption is taken in [26]). So, we can assume that for any input i and any state of the implementation s there always exists a transition $(s, s, i, \texttt{null}, Q, Z, C)$ where `null` is a special (empty) output symbol, $Q(\bar{x}) \equiv \neg \bigvee \{Q'(\bar{x}) \mid \exists \text{ a transition } (s, s', i, o, Q', Z', C')\}$, $Z(\bar{x}) = \bar{x}$, and $C(\bar{x}) = 0$. Other solutions consist in adding a transition leading to an *error* state or generating a transition to the initial state. Finally, we consider that implementations may not present non-observable non-deterministic behavior (see [22] for a framework for deterministic implementations and non-deterministic specifications). For example, an implementation cannot have simultaneously two transitions as $(s, s_1, i, o, Q_1, Z_1, C_1)$ and $(s, s_2, i, o, Q_2, Z_2, C_2)$. Note that we do not restrict observable non-determinism, that is, we may have both the transition $(s, s_1, i, o_1, Q_1, Z_1, C_1)$ and $(s, s_2, i, o_2, Q_2, Z_2, C_2)$, with $o_1 \neq o_2$.

First, we introduce an implementation relation where time is not considered.

Definition 6. Let S and I be two simplified systems. We say that I *non-timely conforms* to S, denoted by $I\ \mathtt{conf}_{nt}\ S$, if for each non-timed evolution $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r) \in \mathtt{NTEvol}(S)$, with $r \geq 1$, we have that $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r) \in \mathtt{NTEvol}(I)$ implies $e' \in \mathtt{NTEvol}(S)$. □

Example 4. Consider the simplified systems S_2 and S_3 depicted in Figure 2 (center and right). Suppose that we have an initial distribution such that all the transitions can be performed. We have $S_3\ \mathtt{conf}_{nt}\ S_2$. Note that the non-timed evolutions from S_3 having as prefix the sequence $a_2/b_3, a_2/b_4$ are not checked because S_2 (playing the role of specification) cannot perform those evolutions.

Let us now consider that the system S_2 is extended with the (implicit) transition $((2,1),(2,1),a_2,\mathtt{null},\mathtt{true},Z,C)$ where $Z(\bar{x}) = \bar{x}$ and $C(\bar{x}) = 0$, so that it fulfills the conditions required for implementations (input enabling). Then, S_2 does not conform to S_3. For example, S_3 may perform the non-timed evolution $e = a_2/b_3, a_2/b_4$, S_2 has the non-timed evolution $e' = a_2/b_3, a_2/\mathtt{null}$, but e' does not belong to the set of non-timed evolutions of S_3. Note that e and e' have the same prefix $a_2/b_3, a_2$. □

Next, we introduce our timed implementation relations. In the following, we call an *instance* of an evolution $e = (i_1/o_1, \ldots, i_r/o_r)$ to a pair (e, t). In the $\mathtt{conf}_a$ relation (conforms *always*) we consider that for any timed evolution (e, t) of the implementation we have that if e is a non-timed evolution of the specification then (e, t) is also a timed evolution of the specification. In the $\mathtt{conf}_w$ relation (conforms in the *worst* case) the implementation is forced, for each timed evolution fulfilling the previous conditions, to be faster than the slowest instance of the same evolution in the specification. The $\mathtt{conf}_b$ relation (conforms in the *best* case) is similar but considering the fastest instance. Let us remind that different instances of the same evolution may appear in a specification as result of the different configurations produced by exchanges of resources.

Definition 7. Let S and I be two simplified systems. We define the following implementation relations:

- $I\ \mathtt{conf}_a\ S$ iff $I\ \mathtt{conf}_{nt}\ S$ and for any timed evolution $(e, t) \in \mathtt{TEvol}(I)$ we have $e \in \mathtt{NTEvol}(S) \Longrightarrow (e, t) \in \mathtt{TEvol}(S)$.
- $I\ \mathtt{conf}_w\ S$ iff $I\ \mathtt{conf}_{nt}\ S$ and for any timed evolution $(e, t) \in \mathtt{TEvol}(I)$ we have $e \in \mathtt{NTEvol}(S) \Longrightarrow (\exists\, t' : (e, t') \in \mathtt{TEvol}(S)\ \wedge\ t \leq t')$.
- $I\ \mathtt{conf}_b\ S$ iff $I\ \mathtt{conf}_{nt}\ S$ and for any timed evolution $(e, t) \in \mathtt{TEvol}(I)$ we have $e \in \mathtt{NTEvol}(S) \Longrightarrow (\forall\, t' : ((e, t') \in \mathtt{TEvol}(S) \Longrightarrow t \leq t'))$.

□

Theorem 1. The relations given in Definition 7 are related as follows:

$$I\ \texttt{conf}_a\ S \Rightarrow I\ \texttt{conf}_w\ S \Leftarrow I\ \texttt{conf}_b\ S$$

Proof Sketch: We only need to consider the evolutions of I belonging also to S (for the rest of evolutions, the premises of the corresponding conformance relation do not hold). First, note that the condition about the non-timed conformance is the same in all the definitions. So, we only need to take into account the conditions on time appearing in the second clause of the corresponding relations. If $I\ \texttt{conf}_a\ S$ then we have that each timed evolution in I fulfilling the conditions given in the definition of $\texttt{conf}_a$ does also appear in S, so we have $I\ \texttt{conf}_w\ S$. If $I\ \texttt{conf}_b\ S$ then each timed evolution of I fulfilling the conditions given in the definition of $\texttt{conf}_b$ is faster than the fastest instance of the same evolution for S. Therefore, it is also faster than the slowest one for S, and so $I\ \texttt{conf}_w\ S$. □

It is interesting to note that if specifications are restricted to take always the same time for each given evolution (independently from the possible derivation taken for such evolution) then the relations $\texttt{conf}_b$ and $\texttt{conf}_w$ would coincide, but they would be still different from the $\texttt{conf}_a$ relation.

Lemma 1. Let $M = (S, I, O, Tr, s_{in}, \bar{y})$ be a simplified system. Let us suppose that there do not exist $((i_1/o_1, \ldots, i_r/o_r), t), ((i_1/o_1, \ldots, i_r/o_r), t') \in \texttt{TEvol}(M)$ with $t \neq t'$. For any simplified system I we have $I\ \texttt{conf}_b\ M$ iff $I\ \texttt{conf}_w\ M$. □

We also introduce a new implementation relation parameterized by the chosen policy for the exchange of resources. In this case, we suppose that the tester can indicate to the implementation under test which policy should be followed to perform exchanges. So, given a set of possible policies, we provide a mechanism to decide which one is more suitable for a given specification. First, we *rephrase* the definition of systems and simplified systems. We consider them as *functions* depending on one parameter (the corresponding policy).

Definition 8. Let us consider the set π of all the policies.

Let $\texttt{Systems}$ be the set of systems. The *parameterized system* created by the composition of the processes $P_1, \ldots, P_n$ with respect to the sets I and O is defined as the function $\texttt{PS} : \pi \longrightarrow \texttt{Systems}$ such that $\texttt{PS}(Pol) = \texttt{Sys}(P_1, \ldots, P_n, I, O, Pol)$.

Let $\texttt{SimpSystems}$ be the set of simplified systems. The *parameterized simplified system* created by the parameterized system $\texttt{PS}$ is defined as the function $\texttt{SPS} : \pi \longrightarrow \texttt{SimpSystems}$ such that $\texttt{SPS}(Pol) = \texttt{Simp}(\texttt{PS}(Pol))$.

Let Pol_1, Pol_2 be two policies and $\texttt{SPS}$ be a simplified parameterized system. Let us consider $I_1 = \texttt{SPS}(Pol_1)$ and $I_2 = \texttt{SPS}(Pol_2)$. Let S be a simplified system. We say that the policy Pol_1 is *preferred* to the policy Pol_2 for $\texttt{SPS}$ to conform to S, denoted by $Pol_2 \sqsubseteq_{\texttt{SPS},S} Pol_1$, if we have that for any conformance relation $\texttt{conf}_y$ such that $I_2\,\texttt{conf}_y\,S$ there exists another conformance relation $\texttt{conf}_x$ such that $I_1\ \texttt{conf}_x\ S$ and $\texttt{conf}_x \Rightarrow \texttt{conf}_y$. □

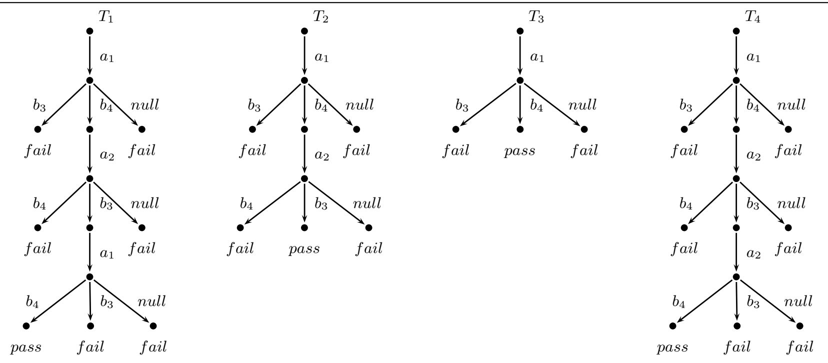


Fig. 3. Examples of Test Cases.

4 Definition and Application of Test Cases

A test represents a sequence of inputs applied to the implementation. Once an output is received, we check whether it is the expected one or not. In the latter case, a fail signal is produced. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. If we are testing an implementation with input and output sets I and O, respectively, tests are deterministic acyclic I/O labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf or another input action. Leaves can be labelled either by *pass* or by *fail*. In the first case we add a *time stamp*. This time will be contrasted with the one that the implementation took to arrive to that point.

Definition 9. A *test case* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C)$ where S is the set of states, I and O are disjoint sets of input and output actions, respectively, $Tr \subseteq S \times I \cup O \times S$ is the transition relation, $s_0 \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:

- S_I is the set of *input* states. We have that $s_0 \in S_I$. For any input state $s \in S_I$ there exists a unique outgoing transition $(s, a, s') \in Tr$. For this transition we have that $a \in I$ and $s' \in S_O$.
- S_O is the set of *output* states. For any output state $s \in S_O$ we have that for any $o \in O$ there exists a unique state s' such that $(s, o, s') \in Tr$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I, s' \in S$ such that $(s, i, s') \in Tr$.
- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. Besides, for any state $s \in S_F \cup S_P$ we have that there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s, a, s') \in Tr$.

Finally, $C : S_P \longrightarrow \mathbb{R}_+$ is a function associating time stamps with passing states.

Let $\sigma = i_1/o_1, \ldots, i_r/o_r$. We write $T \overset{\sigma}{\Longrightarrow} s$, if $s \in S_F \cup S_P$ and there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that $\{(s_0, i_1, s_{12}), (s_{r2}, o_r, s)\} \subseteq Tr$, for any $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in Tr$, and for any $1 \leq j \leq r-1$ we have $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$.

We say that a test case T is an *instance* of the test case T' if they only differ in the associated function C assigning times to passing states.

We say that a test case T is *valid* if the graph induced by T is a tree with root at the initial state s_0. □

In Figure 3 we present some examples of test cases (time stamps are omitted). Next we define the application of a tests suite (i.e. a set of tests) to an implementation. We say that the tests suite $\mathcal{T}$ is *passed* if for any test the terminal states reached by the composition of implementation and test belong to the set of *passing* states. Besides, we give timing conditions according to the different implementation relations.

Definition 10. Let I be a simplified system and T be a valid test. We write $I \parallel T \overset{\sigma}{\Longrightarrow}_t s^T$ if $T \overset{\sigma}{\Longrightarrow} s^T$ and $(\sigma, t) \in \texttt{TEvol}(I)$.

We say that I *passes* the set of tests $\mathcal{T}$, denoted by $\texttt{pass}(I, \mathcal{T})$, iff for any test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ and $\sigma \in \texttt{NTEvol}(I)$ there do not exist s^T and t such that $I \parallel T \overset{\sigma}{\Longrightarrow}_t s^T$ and $s^T \in S_F$.

We say that I *passes* the set of tests $\mathcal{T}$ *for any time* iff $\texttt{pass}(I, \mathcal{T})$ and for any $(\sigma, t) \in \texttt{TEvol}(I)$ such that $T' \overset{\sigma}{\Longrightarrow} s^{T'}$ for some $T' \in \mathcal{T}$, there exists $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P$ and $t = C(s^T)$.

We say that I *passes* the set of tests $\mathcal{T}$ *in the worst time* iff $\texttt{pass}(I, \mathcal{T})$ and for any $(\sigma, t) \in \texttt{TEvol}(I)$ such that $T' \overset{\sigma}{\Longrightarrow} s^{T'}$ for some $T' \in \mathcal{T}$, there exists $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P$ and $t \leq C(s^T)$.

We say that I *passes* the set of tests $\mathcal{T}$ *in the best time* iff $\texttt{pass}(I, \mathcal{T})$ and for any $(\sigma, t) \in \texttt{TEvol}(I)$ such that $T' \overset{\sigma}{\Longrightarrow} s^{T'}$ for some $T' \in \mathcal{T}$, we have that for any $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P$ it holds that $t \leq C(s^T)$. □

Due to space limitations we cannot include a test derivation algorithm. The algorithm is based on that given for `ioco` [26] but taking into account the time associated with transitions. These times will be used to define the function assigning time to successful states. For instance, the first three tests in Figure 3 are derived for the system S_2 appearing in Figure 2. Moreover, the last test in the same figure can be used to determine that S_2 does not conform to S_3. Given a specification S, if we call $\texttt{tests}(S)$ to the set of tests returned by the derivation algorithm, and we replace $\mathcal{T}$ in the previous definition by $\texttt{tests}(S)$, we obtain alternative characterizations for the conformance relations presented in Definition 7.

5 Conclusions and Future Work

In this paper we have shown how the concepts underlying PAMR can be added in a natural way to the formalism of EFSMs. We have defined different implementation relations by taking into account timing relations between specifications and implementations. We consider that this paper represents only the basis of our study, so some points should be extended or improved. A line for future work consists in considering that exchanges can be controlled and/or observed. In this case, we would be able to control the exchanges that a process (that is, a part of a system) can do. Therefore, the definition of simplified systems should be modified to consider that exchanges may appear in the corresponding traces. This extension can be also used to perform a better study of the role played by policies in systems. Finally, we are also studying how the testing equivalence presented in [9], where the transitive closure of `conf` is studied as a testing equivalence, can be adapted to deal with our new implementation relations.

Acknowledgments. We would like to thank Alex Petrenko for useful discussions on conformance testing relations and in the application of these relations to PAMR. We would also like to thank Natalia López for her help while preparing this paper. Finally, we thank the anonymous referees for their helpful remarks.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. K.J. Arrow. *Social Choice and Individual Values*. Wiley, 2nd edition, 1963.
3. M. Barbuceanu and W.K. Lo. Multi-attribute utility theoretic negotiation for electronic commerce. In *AMEC 2000, LNAI 2003*, pages 15–30. Springer, 2001.
4. P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1-2):179–219, 1997.
5. E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS specifications, their implementations and their tests. In *Protocol Specification, Testing and Verification VI*, pages 349–360. North Holland, 1986.
6. A. Cavalli, J.P. Favreau, and M. Phalippou. Standardization of formal methods in conformance testing of communication protocols. *Computer Networks and ISDN Systems*, 29:3–14, 1996.
7. D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems*, 1997.
8. M. Dastani, N. Jacobs, C.M. Jonker, and J. Treur. Modelling user preferences and mediating agents in electronic commerce. In *Agent Mediated Electronic Commerce, LNAI 1991*, pages 163–193. Springer, 2001.
9. D. de Frutos-Escrig, L.F. Llana-Díaz, and M. Núñez. Friendly testing as a conformance relation. In *Formal Description Techniques for Distributed Systems and Communication Protocols (X), and Protocol Specification, Testing, and Verification (XVII)*, pages 283–298. Chapman & Hall, 1997.

10. T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Workshop on Testing of Communicating Systems*, pages 197–214. Kluwer Academic Publishers, 1999.
11. ISO/IEC. ODP Trading Function. Draft International Standard 13235, ISO - Information Processing Systems, 1995.
12. S. Kalyanasundaram, E.K.P. Chong, and N.B. Shroff. Optimal resource allocation in multi-class networks with user-specified utility functions. *Computer Networks*, 38:613–630, 2002.
13. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
14. L.P. Lima and A. Cavalli. A pragmatic approach to generating tests sequences for embedded systems. In *10th Workshop on Testing of Communicating Systems*, pages 288–307. Chapman & Hall, 1997.
15. N. López, M. Núñez, I. Rodríguez, and F. Rubio. A formal framework for e-barter based on microeconomic theory and process algebras. In *Innovative Internet Computer Systems, LNCS 2346*, pages 217–228. Springer, 2002.
16. N. López, M. Núñez, I. Rodríguez, and F. Rubio. Including malicious agents into a collaborative learning environment. In *8th Intelligent Tutoring Systems, LNCS 2363*, pages 51–60. Springer, 2002.
17. S.H. Low and D.E. Lapsley. Optimization flow control I: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–875, 1999.
18. D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Trans. on Computer Systems*, 13(4):356–398, 1995.
19. M. Núñez and I. Rodríguez. `PAMR`: A process algebra for the management of resources in concurrent systems. In *FORTE 2001*, pages 169–185. Kluwer Academic Publishers, 2001. An extended version of this paper is available at: `http://dalila.sip.ucm.es/~manolo/papers/pamr.ps`.
20. M. Núñez and I. Rodríguez. Applying `PAMR` to the management of software projects: Humans as resources, 2002. Submitted for publication.
21. A. Petrenko, N. Yevtushenko, and G. von Bochmann. Fault models for testing in context. In *Formal Description Techniques for Distributed Systems and Communication Protocols (IX), and Protocol Specification, Testing, and Verification (XVI)*, pages 163–178. Chapman & Hall, 1996.
22. A. Petrenko, N. Yevtushenko, and G. von Bochmann. Testing deterministic implementations from their nondeterministic specifications. In *9th Workshop on Testing of Communicating Systems*, pages 125–140. Chapman & Hall, 1996.
23. L. Rasmusson and S. Janson. Agents, self-interest and electronic markets. *Knowledge Engineering Review*, 14(2):143–150, 1999.
24. J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001.
25. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.
26. J. Tretmans. Testing concurrent systems: A formal approach. In *CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.
27. H. Yaiche, R.R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwith allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking*, 8(5):667–678, 2000.