PAMR: A PROCESS ALGEBRA FOR THE MANAGEMENT OF RESOURCES IN CONCURRENT SYSTEMS*

Manuel Núñez and Ismael Rodríguez
Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
{mn,ir}@sip.ucm.es

Abstract In this paper we present a *process algebra for the management of resources* in concurrent systems. Our aim is to define a formal framework that can help in the task of specifying systems that depend, for their execution, on a set of *resources* that they use. Usually systems consist in a set of processes. In order to improve their performance, these processes will be able to exchange resources among them. In our language, processes will consist in a *behavior* (formalized as a LOTOS process) and in information about the resources that they own. Systems will be defined as the parallel composition of a set of processes. We will study some examples applying the features of **PAMR.** These examples will try to show the usefulness of our language for specifying and analyzing concurrent systems where resources play an important role.

Keywords: Extension of FDT's; Semantical Foundations; Process Algebras.

1. INTRODUCTION

During the last two decades Process Algebras have been used to specify and verify different kinds of systems. Nevertheless, most process algebraic models lack the ability to appropriately express the relation between the behavior of processes and the resources that they use. Actually,r esources are usually modeled as *processes.* In this paper we will present a process algebra to deal with systems where resources must be taken into consideration. In order to illustrate the kind of systems that we will deal with, let us introduce the following simple running example. Let us consider a system consisting in the parallel execution of n subsystems $(P_1 \ldots P_n)$ and m different kinds of resources that these subsystems may use (let us suppose that the total quantity of the resource i is

*Research supported in part by the CICYT project TIC 2000-0701- C02-0 1.

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 is, in the beginning, subsystem 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: $\forall\ 1 \leq i \leq \leq m : \ \sum_j x_i^j \leq x_i$. Finally, let us suppose that subsystems have a *preference* on how they *like* resources. For example, suppose that a subsystem 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 another subsystem 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 order to describe this kind of systems, we will introduce a language where the usual description of the behavior of processes is extended with additional information. In our language, a process will not only take into account its activity (that is, which actions can be performed at a given point of time) but it will also consider which resources can be used during its execution. Besides, subsystems may *exchange* resources with other subsystems. For instance, in the example given above, 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, our language will provide mechanisms to, starting with an initial distribution of resources, accomplish a better performance of the studied system.

Example 1 Consider a system where two programs are running. Using operating systems terminology suppose that one of the programs is an I/O-bound process (e.g. a process printing a paper) while the other one is a CPU-bound process (e.g. a process compiling the LaTeX source of another paper). Even if the initial distribution would assign the same resources to both processes (in terms of CPU use and access to I/O), it is clear that both processes would perform better if they exchange their resources in the adequate way. That is, even though the I/O-bound process will keep some time quantum of CPU, it is willing to reduce it (with respect to the original distribution) in order to get better access to I/O, and the other way around for the CPU-bound process. □

We will define our language in two steps. First, we consider a *base language* where we can specify the usual behavior of *processes.* For this purpose, we will use a LOTOS like language. As we already said, processes will also contain information about the available resources: The resources that they own, the resources that they need to perform actions, etc. In particular, there will be a function indicating their preferences on resources: A *utility function.* This function will compute values according to the actions that the process is able to perform immediately. For example, consider a program that initially behaves as

a CPU-bound process and after a while its behavior is as an I/O-bound process. This must be reflected in its utility function. In the beginning, it will prefer to keep a bigger time quantum of CPU, but in the end, its utility function will report bigger values if it keeps better access to I/O.

The second step consists in the combination of processes to build *systems.* A system will be the parallel composition of a set of processes. Processes will be able to communicate with each other by two kinds of operations. First, we will have the usual mechanism for synchronization in parallel operators, that is, there will be some actions that processes may perform asynchronously and some actions that they have to perform synchronously. More important, in order to improve their performances, processes will be able to exchange resources. Besides, *harmful* exchanges will be forbidden. For instance, it will not be 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. We will consider a parallel operator where communications are not restricted to be exactly between two processes. For instance, we will allow that three processes synchronize in an action a or that four processes exchange, in one step, resources among them.

Regarding exchange of resources, we will consider two *policies.* The first one, that we call *preserving utility,* will allow exchanges of resources only in the case that at least one of the processes improves its situation, and no process gets worse after the exchange. For example, this policy would not allow P_{j_1} to *trade* with P_{j_2} one unit of i_1 by one unit of i_2, because even though P_{j_1} improves its situation we have that P_{j_2} gets worse. The second policy,that we call *maximizing utility,* will not be based on the particular situations of processes but in the situation of the system. In this case, exchanges will be allowed only if they improve the overall performance of the system. For example, this policy would allow the previous exchange only if the *profit* for P_{j_1} is bigger than the *loss* for P_{j_2}. In fact, this policy is very similar to consider that processes are not the owner of resources. This is so because part of the resources of a process can be *expropriated* (as far as the whole system improves).

In terms of related work, our language is based on a previous proposal [12]. In this paper we have extended that language to get a more appropriate framework for the specification of the systems that we are dealing with. Nevertheless, the theoretical framework introduced in [12], where some concepts of microeconomic theory were used, can be inherited by adding some slight modifications. There are models to specify systems sharing resources (e.g. [5]), but in this case resources are just accessed, not traded; this access induces some delays in the behavior of processes. If we consider the maximizing utility policy, our proposal is somehow related to the *ODP trading function* [9]. Nevertheless, under a preserving utility policy, it is the case that a process only uses (and nobody else can use them) the resources that it owns. Besides, under this pol-

icy, trade permanently transfers the *ownership* of the traded resources. Finally, management of resources appears in fields like operating systems or concurrent programming. Resources are usually owned by a *mediator* which allows the processes to use them. So, given the fact that we separate the behavior of a process and how it manages its resources, we think that our language can be successfully used for the specification of this kind of systems.

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 present two examples of the use of our language and we study the absence of deadlocks for the defined systems. Finally, in Section 4 we present our conclusions and some directions for further research. An extended version of this paper, where a bisimulation semantics is defined and more complex examples are presented, can be found in [13].

2. DEFINITION OF THE LANGUAGE

In this section we present PAMR as well as its operational semantics. First, we will introduce the notion of *process.* Then, we will say that a system is the parallel composition of several *communicating* processes. We will define the operational semantics of systems by means of three rules. The first rule will describe how processes exchange resources among them. Processes will exchange resources until no more *useful* exchanges are possible. Then, the last two rules define how systems perform actions (possibly by synchronizing among the corresponding processes). We will finish this section by presenting some useful results for verifying some properties in Section 3.

Before we present the formal definition of our language, we introduce some mathematical notation which will be used in the rest of the paper.

Definition 1 We consider $\mathbf{R}_+ = \{x \in \mathbf{R} \mid x \geq 0\}$. By abuse of notation, we will consider $\frac{r}{0} = \infty$. Let $r \in \mathbf{R}_+$. Then, $\mathtt{trunc}(r)$ denotes the natural number resulting from discarding the decimal part of r (e.g. $\mathtt{trunc}(1.2) = 1$).

We will usually denote *vectors* in $\mathbf{R}^n$ (for $n \geq 2$) by $\bar{x}, \bar{y}, \ldots$. Given $\bar{x} \in \mathbf{R}^n$, x_i denotes its *i-th* component. We extend some usual arithmetic operations to vectors. Let $\bar{x}, \bar{y} \in \mathbf{R}^m$. We define $\bar{x} + \bar{y} = (x_1 + y_1, \ldots, x_n + y_n)$. We write $\bar{x} \leq \bar{y}$ if for any $1 \leq i \leq n$ we have $x_i \leq y_i$.

We will usually denote *matrices* in A^{n*m} (for $n, m \geq 2$, and a set A) by calligraphic letters $\mathcal{E}, \mathcal{E}_1 \ldots$. Let $A \in A^{n*m}$, and i, j be such that $1 \leq i \leq n$ and $1 \leq j \leq m$. We will denote the $(n * (i - 1) + j)th$ component of A by A_{ij}, that is, if we consider A as a matrix, this component corresponds to the element located at file i and column j.

Given a set A, $P(A)$ denotes the set containing all the subsets of A. □

$$(\text{PRE})\ \frac{}{\mu;B \xrightarrow{\mu} B} \qquad (\text{PAR1})\ \frac{B_1 \xrightarrow{\mu} B_1' \wedge \mu \notin A}{B_1 \|_A B_2 \xrightarrow{\mu} B_1' \|_A B_2}$$

$$(\text{CHO1})\ \frac{B_1 \xrightarrow{\mu} B_1'}{B_1 + B_2 \xrightarrow{\mu} B_1'} \qquad (\text{PAR2})\ \frac{B_1 \xrightarrow{a} B_1' \wedge B_2 \xrightarrow{a} B_2' \wedge a \in A}{B_1 \|_A B_2 \xrightarrow{a} B_1' \|_A B_2'}$$

$$(\text{CHO2})\ \frac{B_2 \xrightarrow{\mu} B_2'}{B_1 + B_2 \xrightarrow{\mu} B_2'} \qquad (\text{PAR3})\ \frac{B_2 \xrightarrow{\mu} B_2' \wedge \mu \notin A}{B_1 \|_A B_2 \xrightarrow{\mu} B_1 \|_A B_2'}$$

$$(\text{HID1})\ \frac{B_1 \xrightarrow{\mu} B_1' \wedge \mu \notin A}{\text{hide } A \text{ in } B_1 \xrightarrow{\mu} \text{hide } A \text{ in } B_1'} \qquad (\text{HID2})\ \frac{B_1 \xrightarrow{a} B_1' \wedge a \in A}{\text{hide } A \text{ in } B_1 \xrightarrow{\tau_a} \text{hide } A \text{ in } B_1'}$$

$$(\text{REC})\ \frac{B\{X := B/X\} \xrightarrow{\mu} B'}{X := B \xrightarrow{\mu} B'}$$

Figure 1. Operational Semantics for the Base Language.

The behaviors of processes will be defined by means of a usual process algebra. In this case, we will consider a simple LOTOS like *base language.* First, we give an auxiliary definition introducing the sorts for actions.

Definition 2 Let **Act** be the set of *visible* actions ($a, b \ldots$ range over **Act**). Let $\mathbf{Act}_\tau$ be the set of *internal* actions ($\mu, \mu \ldots$ range over $\mathbf{Act} \cup \mathbf{Act}_\tau$). We suppose that $\tau \in \mathbf{Act}_\tau$, that there exists a bijection $f: \mathbf{Act} \rightarrow (\mathbf{Act}_\tau - \{\tau\})$, and $\mathbf{Act} \cap \mathbf{Act}_\tau = \emptyset$. Given a visible action $a \in \mathbf{Act}$, we will denote $f(a)$ by τ_a. We denote by **ACT** the *set of actions,* that is, $\mathbf{ACT} = \mathbf{Act} \cup \mathbf{Act}_\tau$. Finally, let *Id* be a set of (basic) process identifiers. □

Let us note that we will have not only a unique internal action, but a whole set of them. In addition to the usual τ action, we consider an internal action for each of the visible actions. For an *external* observer, all the internal actions will be equal. The difference among them comes from the fact that they will need different resources to be performed. So, if a process needs a set of resources $\bar{x}$ to perform a visible action *a,* and this action is *hidden,* the resulting action, that is τ_a, will need the same amount of resources $\bar{x}$ to be performed. Sets of internal actions appear in other models for concurrent processes (for example, for I/O automata [10]).

Definition 3 The set of *basic processes,* denoted by, B is given by the BNF-expression $B ::= \text{stop} \mid X \mid \mu\,;B \mid B + B \mid B \|_A B \mid \text{hide } A \text{ in } B \mid X := B$, where $\mu \in \mathbf{ACT}$, $A \subseteq \mathbf{Act}$, and $X \in Id$. □

The operational semantics for the base language is given in Figure 1, and it is standard. Let us remind that $B\{B'/X\}$ represents the replacement of the free occurrences of the variable X in B by the term B'. Let us also remark that, in rule (HID2), the result of hiding a visible action a is not τ but τ_a. The

following definition will be used later on. It computes the actions that a basic process may perform immediately

Definition 4 Let $B \in B$. We define its *set of immediate actions,* denoted by $\mathbf{Imm}(B)$, as the set $\mathbf{Imm}(B) = \{\mu \in \mathbf{ACT} \mid \exists B' \in B : B \xrightarrow{\mu} B'\}$. □

As we sketched in the introduction, a process will not only consist in a behavior. On the contrary, a process will keep track of the resources assigned to it and of some information relating resources and actions. Specifically, a process P will be defined as a tuple $(B, \bar{x}, u, n, c)$ where the intuitive meaning of the components is:

- $B \in B$ is a *basic process* indicating the behavior of P.
- $\bar{x} \in \mathbf{R}_+^m$ indicates that P owns x_i units of the *i-th* resource, for any $1 \leq i \leq m$.
- u is a function indicating *preferences* between *baskets* of resources. This function takes as parameters a set of actions (the actions that B may perform immediately) and a set of resources, and returns a real number. For example, $u(A, \bar{x}) < u(A, \bar{y})$ means that if $\mathbf{Imm}(B) = A$ then P would prefer to own the basket $\bar{y}$ to $\bar{x}$. So, processes will try to increase the value of u, given A, by exchanging resources with other processes.
- n is a function relating *resources* and speed of execution. This will be modeled by a function taking as parameters an action and a set of resources: $n(a, \bar{x}) = \infty$ means that a needs more than $\bar{x}$ to be performed; $n(a, \bar{x}) = r \in \mathbf{R}_+$ means that a takes r units of time to be performed if the process has the use of the resources $\bar{x}$. A necessary condition for P to perform a is $n(a, \bar{x}) < \infty$. Let us remark that in some situations there will be a strong relation between the functions u and n: Bigger utilities will be produced by bigger amounts of (or better) resources, and this will imply faster performance of actions. So, we may have relations like $n(a, \bar{x}) = \frac{K}{u(\{a\}, \bar{x})}$, for a given constant $K \in \mathbf{R}_+$. Also note that this is not always the situation. For instance, there can be actions that do not need any resources to be performed, actions that need only a strict subset of them, etc. That is why we have preferred to keep both functions.
- c is a function indicating the *consumed* resources after performing an action. In some situations, the execution of an action needs to consume some resources (for example, writing in a bounded buffer); in other situations, resources are created after performing actions (for example, a pop operation from a bounded stack). These situations will be modeled by a function that takes as inputs an action and a set of resources, and returns a set of resources. That is, $c(a, \bar{x}) = \bar{y}$ means that, after performing

a, the set of resources of the process varies from $\bar{x}$ to $\bar{y}$. For example, if $\bar{z} = c(a, \bar{x}) - \bar{x}$ then $z_i > 0$ indicates that, after performing *a,* the process has created z_i units of the *i-th* resource. A necessary condition for a process to perform an action *a* is $c(a, \bar{x}) \geq \bar{0}$. We do not allow *debts* because they could generate inconsistencies. For example, such debts could produce that two processes use simultaneously a printer.

Definition 5 Let us consider that there exists a number $m > 0$ of different resources. We say that the tuple $P = (B, \bar{x}, u, n, c)$ is a *process* if $B \in B$ (the *basic process* defining the behavior of the process), $\bar{x} = (x_1, \ldots x_m) \in \mathbf{R}_+^m$ (the *amounts* of resources owned by *P*), $u : \mathcal{P}(\mathtt{ACT}) \times \mathbf{R}_+^m \longrightarrow \mathbf{R}$ (the *utility function*), $n : \mathtt{ACT} \times \mathbf{R}_+^m \longrightarrow \mathbf{R}_+ \cup \{\infty\}$ (the *necessity* function), and finally $c : \mathtt{ACT} \times \mathbf{R}_+^m \longrightarrow \mathbf{R}^m$ (*consumption* of resources function).

Given a process P_i, we will usually consider $P_i = (B_i, \bar{x_i}, u_i, n_i, c_i)$, that is, indices will denote the process to which $B, \bar{x}, \ldots$ are related. □

The utility and necessity functions must fulfill some simple *rational* properties. We suppose that utility does not decrease if resources increase, that is, for any $A \in P(\mathtt{ACT})$, and any $1 \leq i \leq m$, we have $\frac{\triangle u(A,\bar{x})}{\triangle x_i} \geq 0$. Given the fact that utility functions will be always applied to the initial actions of a behavior *B,* we suppose $u(\emptyset, \bar{x}) = 0$. This property means that a *deadlocked* process does not need any resources, so they will be shared by the rest of the processes (because those resources do not add any utility to the owner). In microeconomic theory, ther e are other restrictions that are usually imposed (strict monotonicity, convexity, etc) but they are not needed in our current framework (see [11, 12] for more details). Regarding the necessity function, we will impose the condition that the function *n* is non-increasing with the number of resources, that is, for any $1 \leq i \leq m$ and $\mu \in \mathtt{ACT}$, we have $\frac{\triangle n(\mu,\bar{x})}{\triangle x_i} \leq 0$. Moreover, we will suppose that $n(a, \bar{x}) = n(\tau_a, \bar{x})$, that is, a visible action *a* needs the same resources that its internal counterpart τ_a.Finally, let us comment on a *trick* that we will intensively use in the examples of Section 3. There are many situations where resources cannot be split. For example, it makes no sense to consider that a process *owns* half of a printer. We will indicate that half of a printer is so *useful* as no printer at all by using utility functions like $u(A, (x_1, \ldots, x_i, \ldots, x_n)) = f(A, x_1, \ldots, \mathtt{trunc}(x_i), \ldots, x_n))$.

ure 1) to processes. The idea is that a process may perform a transition $\xrightarrow{\mu}$ if its corresponding behavior so does, and the process has enough resources. Formally, if $P = (B, \bar{x}, u, n, c)$ then we have the following rule:

$$\text{(PRO)}\frac{B \xrightarrow{\mu} B' \ \wedge \ n(\mu, \bar{x}) < \infty \ \wedge \ c(\mu, \bar{x}) \geq \bar{0}}{P \xrightarrow{\mu} P'}$$

where $P' = (B', c(\mu, \bar{x}), u, n, c)$. Let us note that we have overloaded the symbol $\rightarrow$: It is used both to denote transitions of behaviors and transitions of processes.

A *system* will be the parallel composition of a (finite, non-empty) collection of processes. Processes will communicate in two ways: Either by synchronizing in the execution of actions or by exchanging resources. Given the nature of the systems that we would like to describe, we have decided to use a relatively complex parallel operator. Specifically, our parallel operator is a simplification of [7] where *m* among *n* cooperation is not considered (similar proposals appear in [3,6]). The idea is that synchronizations are not restricted to be binary. The parallel operator will have a tuple of subsets of **Act** as parameter: $(A_1, \ldots, A_n)$. So, if we have the parallel composition of $P_1, \ldots, P_n$, a process P_i will perform the actions in $\mathbf{ACT} - A_i$ asynchronously; if $a \in A_i$ then any P_j such that $a \in A_j$ must synchronize with P_i in order to perform *a*. Similarly, exchanges of resources may involve more than two processes.

Definition 6 Let $A_1, \ldots A_n \subseteq \mathbf{Act}$. A *system* S consists in the parallel composition of *n* processes $P_1, \ldots P_n$ synchronizing, respectively, in the set A_i. We denote the system S by $\|_n^{A_i} P_i$. We denote the set of systems by S. □

From now on we will assume that systems are initially *compatible* with the available resources. That is, if we want to specify a system having the use of a vector of resources $\bar{r}$ by $S = \|_n^{A_i} P_i$ then the sum of the total quantity of resources initially assigned to $P_1, \ldots, P_n$ must be equal to $\bar{r}$., that is, $\sum_i \overline{x_i} = \bar{r}$. In the following we introduce the operational rules for systems. The first rule uses two predicates that will be defined in forthcoming definitions.

Definition 7(*The Exchange Rule*). Let $S = \|_n^{A_i} P_i$ be a system, where for any $1 \leq i \leq n$ we have $P_i = (B_i, \overline{x_i}, u_i, n_i, c_i)$. The operational transitions denoting exchange of resources that S may perform are given by the rule:

$$(\text{Par1}) \frac{\mathtt{valid}(S, \mathcal{E}) \wedge \mathtt{allowed}(S, \mathcal{E})}{S \overset{\mathcal{E}}{\rightsquigarrow} \|_n^{A_i} P_i' \left[\forall\, 1 \leq i \leq n :\ P_i' = (B_i, \overline{x_i} - \sum_j \mathcal{E}_{ij} + \sum_j \mathcal{E}_{ji}, u_i, n_i, c_i)\right]}$$

where $\mathcal{E} \in (\mathbf{R}_+^m)^{n*n}$. We denote by $\rightsquigarrow^*$ the reflexive and transitive closure of $\rightsquigarrow$. We say that S is a *local equilibrium,* denoted by $S \not\rightsquigarrow$, if there do not exist S' and ε such that $S \overset{\mathcal{E}}{\rightsquigarrow} S'$. □

Intuitively, a transition $\|_n^{A_i} P_i \overset{\mathcal{E}}{\rightsquigarrow} \|_n^{A_i} P_i'$ indicates that P_i gives to P_j the quantities of resources indicated by ε_{ij} and receives from it the quantities given by ε_{ji}. The total amount of resources that P_i owns is equal to its original resources minus the resources that it gives plus the resources that it receives. As we will see later, processes will not be allowed to perform actions until the system reaches a local equilibrium. The idea is that an equilibrium represents a

good distribution of resources (because no more *useful* exchanges can be made) and so processes must be *delayed* until a local equilibrium is reached. If we allow a process P_i to perform transitions before the system is an equilibrium (for example, as soon as $n_i(a, \overline{x_i}) < \infty$ and $c(a, \bar{x}) \geq \bar{0}$ hold) then the system will not be working at its *best* possible performance(s).

Now we will define the predicates `valid`(S,ε) and `allowed`(S,ε). The first predicate holds if the processes do not give more resources that the ones that they initially own, and the diagonal of the matrix is filled with $\bar{0}$.

Definition 8 Let $S = \|_n^{A_i} P_i$ be a system, where for any $1 \leq i \leq n$ we have $P_i = (B_i, \overline{x}_i, u_i, n_i, c_i)$. We say that $\mathcal{E} \in (\mathbf{R}_+^m)^{n*n}$ is a *valid exchange matrix for S,* denoted by `valid`(S,ε),if for any $1 \leq i \leq n$ we have $\Sigma_j\, \varepsilon_{ij} \leq \overline{x}_i$ and $\varepsilon_{ii} = \bar{0}$. □

Regarding the predicate `allowed`(S, ε) we have several possibilities. This choice would depend on the kind of policy[1] that we want to implement. As we said in the introduction we consider two possible policies. The first policy, the *preserving utility* policy, will allow exchange of resources 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.* The second policy, the *maximizing utility* policy, will allow exchanges if the overall situation of the system improves. In order to measure the improvement of the system, we consider the sum of the utilities of all the processes.[2] As we commented in the introduction of this paper, such a policy could be interpreted by thinking that processes are not the *owners* of resources because they can be *expropriated* without any compensation. Nevertheless, this policy is more efficient than a totally centralized system of resources delivery because it avoids the return and redelivery of the same resources to a process. In economic terms, the first of these policies can be seen as a kind of *market economy* while the second one could be interpreted as a kind of *planned economy.*

Definition 9 Let $S = \|_n^{A_i} P_i$ be as system, where for any $1 \leq i \leq n$ we have $P_i = (B_i, \overline{x}_i, u_i, n_i, c_i)$, and let ε be a matrix such that `valid`(S, ε).

The `allowed`(S, ε) predicate under a *Preserving Utility Policy* is defined as $\forall\ 1 \leq i \leq n$ we have $u_i(\mathtt{Imm}(B_i), \overline{x_i}) \leq u_i(\mathtt{Imm}(B_i), \overline{x_i} - \sum_j \mathcal{E}_{ij} + \sum_j \mathcal{E}_{ji})$ and $\exists\ 1 \leq k \leq n :\ u_k(\mathtt{Imm}(B_k), \overline{x_k}) < u_k(\mathtt{Imm}(B_k), \overline{x_k} - \sum_j \mathcal{E}_{kj} + \sum_j \mathcal{E}_{jk})$.

The `allowed`(S, ε) predicate under a *Maximizing Utility Policy* is defined as $\sum_i u_i(\mathtt{Imm}(B_i), \overline{x_i}) < \sum_i u_i(\mathtt{Imm}(B_i), \overline{x_i} - \sum_j \mathcal{E}_{ij} + \sum_j \mathcal{E}_{ji})$. □

[1] The choice of a *good* policy is not a trivial task. Actually, it is imposible to choose a *perfect* policy. This problem is related with the social welfare aggregator problem. Arrow's *impossibility therom* shows that there does not exist such an aggregator fulfilling a certain set of *desirable* properties (see [1] for more details).

[2] Using microeconomics terminology, this is a *Benthan* social welfare aggregator.

From now on, if the results depend on the chosen policy, we will indicate under which one we are working. Once we have finished the definition of rule (Par1), let us remark that exchange of resources produces a lot of nondeterminism. For example, consider the processes P_{j_1} and P_{j_2} presented in the introduction. For any $a > 0$, and $2 \le x \le 4$, exchanges where P_{j_1} receives a units of the resource i_1 from P_{j_2} and P_{j_1} gives $x \cdot a$ units of the resource i_2 from P_{j_2} will be allowed.

Definition 10 (*The Synchronization Rules*). Let $S = \|_n^{A_i} P_i$ be a system. The operational transitions denoting the actions that S may perform are given by the rules:

$$\text{(Par2)}\ \frac{S \not\rightsquigarrow \wedge\ P_j \xrightarrow{\mu} P_j' \wedge \mu \notin A_j}{S \xrightarrow{\mu} \|_n^{A_i} P_i'' \qquad \left[\forall\, 1 \le i \le n:\ P_i'' = \begin{cases} P_j' & i = j \\ P_i & i \ne j \end{cases}\right]}$$

$$\text{(Par3)}\ \frac{S \not\rightsquigarrow \wedge\ P_j \xrightarrow{a} P_j' \wedge a \in A_j \wedge \forall\, 1 \le k \le n:\ (a \in A_k) \Longrightarrow (P_k \xrightarrow{a} P_k')}{S \xrightarrow{a} \|_n^{A_i} P_i'' \qquad \left[\forall\, 1 \le i \le n:\ P_i'' = \begin{cases} P_i' & \text{if } a \in A_i \\ P_i & \text{if } a \notin A_i \end{cases}\right]}$$

□

Let us note that we have overloaded the symbol denoting transitions. The condition $S \not\rightsquigarrow$ indicates that the system has reached a local equilibrium. If S is not a local equilibrium, more useful exchanges can be made (i.e. the system may improve) and so actions should not be performed. Rule (Par2) is applied for interleaving actions. In this case, the only process that changes is the one involved in the transition. Rule (Par3) says that if a process P_j may perform an action a belonging to its synchronization set A_j, then all the processes having a in their synchronization sets must also perform a; all these processes will perform this action synchronously.

Finally,let us remark that our process algebra is a conservative extension of LOTOS: It is enough to consider that the functions u and n are always equal to 0 and that $c(\mu, \bar{x}) = \bar{x}$, for any $\mu \in$ ACT.

2.1. Some Properties of the Language

In this section we study some properties that local equilibria fulfill according to the functions defining the involved processes. These properties will be used in the next section of this paper when analyzing the defined specifications. The first result says that any system may evolve into a local equilibrium.

Lemma 1F or any system S there exists S' such that $S \rightsquigarrow^* S' \not\rightsquigarrow$. □

The proof of this result is immediate. Let us remark that if S is already a local equilibrium then $S' = S$ (that is, there is a $\rightsquigarrow^*$ derivation with length equal to zero). The following result states that once a local equilibrium has been reached, no process will keep resources that do not add utility to it if there exists another process that uses the resource.

Lemma 2 Let $S = \|_n^{A_i} P_i$. Let us suppose that there exist two processes P_i, P_j, > 0,and $1 \leq i \leq m$ such that for any $\overline{x}$ we have

- $u_i(\mathtt{Imm}(B_i), \overline{x}) = u_i(\mathtt{Imm}(B_i), (x_1, \ldots, x_r - \epsilon, \ldots, x_n))$,and
- $u_j(\mathtt{Imm}(B_j), \overline{x}) < u_j(\mathtt{Imm}(B_j), (x_1, \ldots, x_r + \epsilon, \ldots, x_n))$.

Under the previous conditions, for any S' such that $S \rightsquigarrow^* S' \not\rightsquigarrow$ we have that $x'_{ir} \leq x_{ir} - \epsilon$, where x_{ir} denotes the amount of the resource r owned by P_i in S (similar for x'_{ir} and S').

Proof Sketch: By contradiction. Suppose $x'_{ir} > x_{ir} - \epsilon$. We have that, under both policies, there exists an exchange where P_i gives part of the resource r to a process P_k (there is at least a process increasing its utility by increasing the amount of r). So, S' is not a local equilibrium. □

An immediate corollary of the previous result is that if the condition holds for any ε such that $x_{ir} \geq \epsilon > 0$ then $x'_{ir} = 0$. Note that in the previous result is essential to suppose that another process increase its utility by increasing its amount of the resource r. The next result states that under a maximizing utility policy, all the local equilibria that can be reached from a system S have the same *global* utility.

Definition 11 Let $S = \|_n^{A_i} P_i$ be a system, where for any $1 \leq i \leq n$ we have $P_i = (B_i, \overline{x_i}, u_i, n_i, c_i)$. The *total* utility of S, denoted by $\mathtt{total}(S)$, is defined as $\sum_i u_i(\mathtt{Imm}(B_i), \overline{x_i})$. □

Theorem 1 Let $S = \|_n^{A_i} P_i$ be a system, where for any $1 \leq i \leq n$ we have $P_i = (B_i, \overline{x_i}, u_i, n_i, c_i)$ Let S_1, S_2 be systems such that $S \rightsquigarrow^* S_1 \not\rightsquigarrow$, and $S \rightsquigarrow^* S_2 \not\rightsquigarrow$. If the *maximizing utility* policy is being used then we have $\mathtt{total}(S_1) = \mathtt{total}(S_2)$.

Proof: Let $S_1 = \|_n^{A_i} P_{1i}$ and $S_2 = \|_n^{A_i} P_{2i}$, where for any $1 \leq i \leq n$ we consider $P_{1i} = (B_i, \overline{x_{1i}}, u_i, n_i, c_i)$ and $P_{2i} = (B_i, \overline{x_{2i}}, u_i, n_i, c_i)$. Suppose that $\mathtt{total}(S_1) > \mathtt{total}(S_2)$. Let ε be an exchange matrix such that for any $1 \leq i \leq n$ we have $\sum_j \mathcal{E}_{ji} - \mathcal{E}_{ij} = x_{1i} - x_{2i}$ and $\mathtt{valid}(S_2, \varepsilon)$. Given the fact that $\sum_i x_{1i} = \sum_i x_{2i}$, such a matrix fulfilling $\mathtt{allowed}(S_2, \varepsilon)$ always exists. Therefore, there exists S' such that $S_2 \overset{\mathcal{E}}{\rightsquigarrow} S'$, which represents a contradiction. □

Note that different local equilibria may have different assignment of resources among processes; the previous result only assures that the sum of the utilities is the same, not that the composition of the baskets are equal. Note that this result does not hold under a preserving utility policy: Different equilibria do not have necessarily the same total utility.

3. EXAMPLES

In this section we will show how PAMR can be used to specify and analyze concurrent systems where resources play an important role. We will present two classical examples: The dining philosophers and consumers/producers. Besides, we will study the absence of deadlock in these systems.

In the following we assume that an undefined value of a function is set to an arbitrary value. Actually, these cases will not be possible because they will correspond with a set of actions (resp. with an action) not reachable by the corresponding processes.

3.1. The Dining Philosophers

In this classical problem, five (male) philosophers stay by a dining table, which contains five forks. We will consider that these are the *resources* of the system: $fork_1, \ldots, fork_5$. These philosophers have only two tasks: To think and to eat. When a philosopher wants to eat, he must take both forks staying besides his dish (we suppose that philosopher i must take forks i and (i mod 5) + 1). The behavior of the philosophers can be described as:

$$Philosopher_i := think_i ; eat_i ; Philosopher_i$$

We suppose that the initial distribution of forks gives to philosopher i the *i-th* fork, that is, $\overline{x_i} = (\delta_{i1}, \ldots, \delta_{i5})$ where $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ otherwise. Utility functions are defined as:

$$u_i(\{think_i\}, \bar{x}) = 0 \qquad u_i(\{eat_i\}, \bar{x}) = \mathtt{trunc}(fork_i) \cdot \mathtt{trunc}(fork_{(i \bmod 5)+1})$$

That is, if a philosopher wants to eat, he gets utility only if he owns both forks; otherwise, it will be the same to have one or none. Besides, a philosopher gets no utility by holding *useless* forks. We set $u_i(\{think_i\}, \bar{x})$ to an arbitrary value because no resources are needed to think. The necessity functions are defined as follows:

$$n_i(think_i, \bar{x}) = K_i \qquad n_i(eat_i, \bar{x}) = \frac{E_i}{u_i(\{eat_i\}, \bar{x})}$$

In this case, the time that philosophers spend thinking may be different, but does not depend on the resources. Besides, we also assume that they eat at different speeds. Finally, no resources are consumed by performing actions, so

the consumption function is defined as $c_i(a, \bar{x}) = \bar{x}$. In conclusion, the system can be defined as

$$Dining\_Philosophers \;= \|_5^{A_i} P_i$$

where for any $1 \leq i \leq 5$, $A_i = \emptyset$ and $P_i = (Philosopher_i, \overline{x_i}, u_i, n_i, c_i)$.

The following result shows that this system cannot get deadlocked. As in the next example, the definition of the utility functions plays an important role in the absence of deadlocks.

Lemma 3 (*Absence of Deadlocks for Dining_Philosophers*). Consider a system *S* such that

$$Dining\_Philosophers \rightsquigarrow^* S_1 \xrightarrow{a_1} S_1' \rightsquigarrow^* S_2 \cdots \xrightarrow{a_n} S_n' \rightsquigarrow^* S$$

If S is not a local equilibrium then there exist *S'*, e such that $S \overset{\mathcal{E}}{\rightsquigarrow} S'$; otherwise, there exist *S'*, *a* such that $S \xrightarrow{a} S'$.

Proof: The first case is trivial from Lemma 1. If a philosopher is willing to think, this action may be performed (no resources are needed). Suppose that all the philosophers desire to eat, and none of them can. This implies that all of them have zero utility. Regardless of the chosen policy, *S* is not a local equilibrium, because any exchange where some philosopher takes his two forks is allowed (note that fractions of forks will not be exchanged because they do not increase utility).

3.2. The Bounded-Buffer Producers/Consumers Problem

Consider *n* producers and *m* consumers. The former have access to a buffer where they can place their products; these products will be taken by the latter. The buffer is bounded: A consumer cannot get out a product if the buffer is empty and a producer cannot put in a product if the buffer is full. In order to avoid any damage in the structure, the buffer must be accessed preserving mutual exclusion. The behaviors of the involved processes are given by:

$$Producer := \; produce \; ; \; enqueue_1 \; ; \; enqueue_2 \; ; \; Producer$$

$$Consumer := \; dequeue_1 \; ; \; dequeue_2 \; ; \; consume \; ; \; Consumer$$

In order to visualize how mutual exclusion is implemented, the *enqueue* and *dequeue* operations are split into two different steps. The initial resources of the system are: a unit of *mutual exclusion,* zero *produced* units, and *p* free *places.* We assume that these *resources* are randomly distributed among the processes in such a way that $\sum_{i=1}^{n+m} \overline{x_i} = (1, 0, p)$. We will define a unique utility function for all consumers and producers:

$$u(\{enqueue_1\},\bar{x}) = \begin{cases} 1 & \text{if } \mathtt{trunc}(x_1)\cdot\mathtt{trunc}(x_3)\geq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$u(\{enqueue_2\},\bar{x}) = \begin{cases} 2 & \text{if } x_1 = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$u(\{dequeue_1\},\bar{x}) = \begin{cases} 1 & \text{if } \mathtt{trunc}(x_1)\cdot\mathtt{trunc}(x_2)\geq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$u(\{dequeue_2\},\bar{x}) = \begin{cases} 2 & \text{if } x_1 = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$u(\{produce\},\bar{x}) = K_1 \qquad u(\{consume\},\bar{x}) = K_2$$

Let us note that the order structure of the buffer is not fully represented: The distinct products or distinct free places are not distinguished. Regarding the necessity function, it can be defined from the utility function as:

$$n(a,\bar{x}) = \frac{K_3}{u(\{a\},\bar{x})}$$

Let us remark that if a process does not own the necessary resources, its utility function will return 0, and so, the necessity function will be equal to infinite (let us remember that we consider $\frac{r}{0} = \infty$). In this example, the last two resources are created and consumed, so we have to define the value of the consumption function in the appropriate way:

$$c(enqueue_1,\bar{x}) = (x_1, x_2, x_3 - 1) \qquad c(enqueue_2,\bar{x}) = (x_1, x_2 + 1, x_3)$$
$$c(dequeue_1,\bar{x}) = (x_1, x_2 - 1, x_3) \qquad c(dequeue_2,\bar{x}) = (x_1, x_2, x_3 + 1)$$
$$c(produce,\bar{x}) = \qquad c(consume,\bar{x}) = \bar{x}$$

Finally, the system may be specified as:

$$Bounded\_Buffer = \|_{n+m}^{A_i} P_i$$

where, for any $1 \leq i \leq n$ we have $P_i = (Producer, \overline{x_i}, u, n, c)$, and for any $n+1 \leq j \leq n+m$ we have $P_j = (Consumer, \overline{x}j, u, n, c)$. Besides, for any $1 \leq k \leq n+m$ we have $A_k = \emptyset$.

One important property that we must have is *mutual exclusion.* The next result states that accesses to the buffer are done by preserving this property: Once a process performs an action belonging to $\{enqueue_1, dequeue_1\}$, it is assured that this process will perform the corresponding second part of the performed action ($enqueue_2$ and $dequeue_2$ respectively) before any other process performs an action belonging to $\{enqueue_1, dequeue_1\}$.

Lemma 4 Let S be a system such that

$$Bounded\_Buffer \rightsquigarrow^* S_1 \xrightarrow{a_1} S_1' \rightsquigarrow^* S_2 \xrightarrow{a_2} \cdots \rightsquigarrow^* S \not\rightsquigarrow$$

Let us suppose $S \xrightarrow{a} S' \rightsquigarrow^* S'' \not\rightsquigarrow$, where $a \in \{enqueue_1, dequeue_1\}$ and a has been performed by P_j (that is, $P_j \xrightarrow{a} P_j'$). For any transition $S'' \xrightarrow{b} S'''$ such that $b \in \{enqueue_1, dequeue_1, enqueue_2, dequeue_2\}$ we have that if $a = enqueue_1$ then $b = enqueue_2$ (resp. if $a = dequeue_1$ then $b = dequeue_2$) and this transition has been performed by the evolution of P_j' from S' to S''.

Proof Sketch: Let us consider the producers case (for consumers is similar). Under the *maximizing utility policy,* when a producer performs its $enqueue_1$ action, for any reachable local equilibrium, the unit of mutual exclusion resource is given again to that producer because no other process would have more utility than this one by owning it (note that this producer will have utility 2). Under the *preserving utility policy,* when a producer performs an $enqueue_1$ action, the mutual exclusion resource cannot be taken by another process because the utility of the owner would decrease. □

Lemma 5 (*Absence of Deadlocks for Bounded_Buffer*). Let us consider a system S such that $Bounded\_Buffer \rightsquigarrow^* S_1 \xrightarrow{a_1} S_1' \rightsquigarrow^* S_2 \cdots \xrightarrow{a_n} S_n' \rightsquigarrow^* S$. If S is not a local equilibrium then there exist S', ε such that $S \overset{\varepsilon}{\rightsquigarrow} S'$; otherwise, there exist S', a such that $S \xrightarrow{a} S'$.

Proof Sketch: If there are no products then any producer will be able to produce. If there are no free places, any consumer will be able to consume. In both policies, the mutual exclusion resource will be taken (and owned) by a process only if it makes that process to perform an action. Therefore, at least one process will get the resources it needs. □

4. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a formalism to specify systems where resources can be exchanged among subsystems. These exchanges will improve the performance of the system. We have studied some (theoretical) properties of the language. Finally, we have specified and studied two examples where the characteristics of our language have been shown.

The referees of this paper have suggested new interesting directions of research. For example, necessity functions play no relevant role in the current theory. We would like to define a real-time extension of our language where these functions play a fundamental role: They will induce delays in the behavior of processes. These delays can be either deterministic or can be defined by means of a random variable. In the latter case, we will consider some of the current models of stochastic process algebras (e.g. [2,4, 8]). Besides, we will consider that the exchange of resources takes time. So, sequences of $\rightsquigarrow$ transitions should be grouped. Another interesting point is that utility functions (as well as necessity and consumption functions) are *static,* that is, they do not change along the performance of the system. Once time is taken into

account, it is straightforward to extend the previous framework to consider *dynamic* functions. Finally, the relation between our framework and management of software projects should be explored. Indeed, the tasks of a project can be seen as processes while the members of the project can be seen as the resources *used* by the tasks.

Acknowledgments. We would like to thank the referees of this paper for the careful reading and the very interesting suggestions to improve the current framework. We would also like to thank the referees of [12] for pointing out some interesting directions which have been reflected in this paper. Finally, we would like to thank David de Frutos, Natalia López, Javier Rodríguez, and Fernando Rubio for helpful discussions on the topic of this paper.

REFERENCES

[1] K. J. Arrow. *Social Choice and Individual Values.* Wiley, 2nd edition, 1963.

[2] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science,* 202:1–54, 1998.

[3] T. Bolognesi. A graphical composition theorem for networks of LOTOS processes. In *10th International Conference on Distributed Computing Systems,* pages 88–95. IEEE-CS Press, 1990.

[4] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-markov processes. To appear in Theoretical Computer Science, 2001.

[5] 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.

[6] J. Davies and S. Schneider. A brief history of timed CSP. *Theoretical Computer Science,* 138:243–271, 1995.

[7] H. Garavel and M. Sighireanu. A graphical parallel composition operator for process algebras. In *Formal Description Techniques for Distributed Systems and Communication Protocols (XII), and Protocol Specification, Testing, and Verification (XIX),* pages 185–202. Kluwer Academic Publishers, 1999.

[8] H. Hermanns, U. Herzog, and J.-P.Katoe n. Process algebra for performance evaluation. To appear in Theoretical Computer Science, 2001.

[9] ISO/IEC. ODP Trading Function. Draft International Standard 13235, ISO - Information Processing Systems, 1995.

[10] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th ACM Symp. on Principles of Distributed Computing,* pages 137–151, 1987.

[11] A. Mas-Colell, M.D. Whinston, and J.R. Green. *Microeconomic Theory.* Oxford University Press, 1995.

[12] M. Núñez. Including microeconomic theory into FDTs: A first approach. Available at: `http://dalila.sip.ucm.es/~manolo/papers/exchange.ps.gz`, 2000.

[13] M. Núñez and I. Rodríguez. `PAMR:` A process algebra for the management of resources in concurrent system. Available at: `http://dalila.sip.ucm.es/~manolo/papers/pamr.ps.gz`, 2001.