A Bounded True Concurrency Process Algebra for Performance Evaluation *

M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, J. José Pardo, and Hermenegilda Macià

Escuela Politécnica Superior de Albacete
Universidad de Castilla-La Mancha. 02071 Albacete, Spain.
{MCarmen.Ruiz, Diego.Cazorla, Fernando.Cuartero, Juan.Pardo, Hermenegilda.Macia}@uclm.es

Abstract. A *True Concurrency Timed Process Algebra* which takes into account the number of resources/processors that processes have at disposal is presented. Both its syntax and its operational semantics are defined. An algorithm which allows us to estimate the required time to evolve between states and which allows us to evaluate the performance of a system is also presented. A simple example is used to illustrate how a system can be specified by means of our language and how the performance evaluation algorithm works.

1 Introduction

It is a well-known fact that concurrency involves a radical increase in the complexity of a system. Theoretical models to study these systems at a formal level became of great importance. The first models abstract many real properties of concurrent systems which are considered non essential for them, such as duration and structure of actions, properties of communication networks, distribution of system components, number of processors and so on. By leaving these non essential properties aside, it is easier to study the behaviour of processes at a formal level due to a clearer design specification. However, these simplifications have a cost which is the loss of all the information which has been abstracted. This fact can be observed in early formal models such as *Petri nets* and *process algebras*.

Once the basic models have been thoroughly studied, it is necessary to extend them to deal with these other aspects, usually quantitative ones, which have been lost in the abstraction. Extended models have been proposed to represent aspects such as fairness [1, 2], timed models [3, 4], Markovian timed models [5, 6] and probabilistic models [7–9].

The aim of this paper is to introduce a timed process algebra, called **BTC** (for Bounded True Concurrency), which allows us to evaluate the performance

* This work has been partially supported by the MCyT project "Description and Performance of Distributed Systems and Application to Multimedia Systems" (Ref. TIC2003-07848-c02-02) and the JCCM project "Design and Implementation of Efficient Multimedia Systems by using Formal Techniques" (Ref. PAC-03001)

of a system, considering that the (limited amount of) available processors have to be shared by all the processes. We can generalised this approach and consider systems that share homogeneous resources of any kind (processors, buses, disks, etcetera). Nevertheless, we cannot consider yet processes that need to use heterogeneous resources (e.g. one processor unit and two disk units). This extension of the process algebra is left for future work.

The most important consequence of this approach is that if there are more processes than resources then not all of them can be simultaneously executed. A process has to wait until it allocates the resources needed to continue its execution. This means that we can find two kinds of delays in the execution of a process: delays related to the synchronization of processes, and delays related to the allocation of resources. The former is usual in a (theoretical) concurrent context, but the latter is only taken into account if we consider a limited amount of available resources.

In order to present our model, it is worth to review an important characteristic of the currents formal models of concurrency. They can be roughly divided into two groups: *Interleaving* based models, in which the independent execution of two processes is modelled by specifying the possible interleaving of their actions; and *true concurrency* models, in which the causal relations between actions are explicitly represented. Our process algebra is based on the notion of true concurrency where concurrent behaviour can be distinguished from interleaving one by considering that $(a|b) \not\equiv (a.b + b.a)$

Furthermore, the bounded true concurrency approach we present in this paper considers all the cases between interleaving based models and true concurrency based models. In particular, if we consider only one resource unit, we will be considering an interleaving approach, while if we consider infinite resources we will have a true concurrency approach. From this point of view, our model extends both interleaving and true concurrency based models.

The performance of a system may be different depending on the number of available resources. If we denote by $[P]_N$ a process which is executed in the scope of N resources, we will have that $[a|b]_1 \equiv [a.b + b.a]_1$ but $[a|b]_n \not\equiv [a.b + b.a]_n$ $\forall n > 1$. In general, given a process P, $\forall n, m \in \mathbb{N}$ with $n \neq m$, we have that the performance of $[P]_n$ may be (or may be not) different from the performance of $[P]_m$. Actually, one of the main applications of this process algebra consists in finding a natural number n such that $\forall i \in \mathbb{N}, i \geq n$, $[P]_i$ is *equivalent* (from a performance point of view) to $[P]_n$, i.e., n represents the maximal degree of parallelism we can exploit in the system, and as a result, we obtain the optimum number of resources needed in order to speed up the performance of the system.

BTC is based on CSP [10]. We have extended its syntax in order to consider duration of actions by means of a timed prefix operator. The operational semantics has been also extended in order to consider the context (number of resources) in which processes are executed. A notion of timed bag of actions is also introduced to represent the concurrent (simultaneous) execution of processes.

In the literature we may find several theoretical models that consider the notion of resource consuming. Timed process algebras that consider resources are presented in [11–16]. In [11, 12] a discrete timed process algebra for limited parallelism (CCSLP) is presented. Time is associated to actions by considering special actions t_n where n represents the consumption of n (homogeneous) resource units. This means that, if we want to specify an action that lasts 2 units of time, we have to write $a.t_1.t_1.NIL$, where a does not represent the execution of the action but just the moment in which the action starts its execution. From our point of view, the main problem of that model is that it may be difficult to know when an action finishes. If we consider three actions (a, b, c) with duration 2 (i.e., $a.t_1.t_1.NIL$, $b.t_1.t_1.NIL$, $c.t_1.t_1.NIL$), and two resources (processors), a possible sequence of actions is $a.b.t_2.c.t_2.t_2$, where the first t_2 represents the execution of the first unit of a and b, the second t_2 represents the execution of the first unit of c and the second unit of a or b, and the last t_2 represents the execution of the second unit of c and the second unit of b or a. Thus, it is difficult to analyse the performance of a system because it may be impossible to know when the system has reached a state.

In [13] a process algebra for shared processors is presented. The paper is focused on processes sharing a single processor scheduled according to some scheduling strategy. Multi-processors systems can be described but only with static allocation of processes to processors. This situation does not happen in our model where there is no assignment of processes to processors, i.e., a sequential process may be interrupted and later on it can continue its execution in a different processor.

ACSR, a process algebraic approach to the specification and analysis of resource-bound real-time systems is introduced in [14]. In that process algebra, the use of shared (heterogeneous) resources is represented by timed actions and synchronization is supported by instantaneous events. Timed actions are always executed for one time unit (*tick*) and they consume a set of resources during that time. ACSR supports also priorities which are used to decide among processes (actions) competing for resources. In [15] an extension of ACSR considering dense time is presented. In that model, timed actions have a duration given by a positive and finite real number u, where A^u represents the execution of a resource-consuming action A for a time u. Synchronization between timed actions is allowed but only in the case that both actions have the same duration and they use disjoint sets of resources. Other extensions of ACSR are presented in [16] where PACSR (probabilistic ACSR) and ACSR-VP (ACSR with value passing) are introduced.

The approach taken in ACSR and extensions is slightly different from ours. In ACSR, a timed action is a set of pairs $(resource, priority)$, and two (or more) timed actions may be executed simultaneously only if they use different resources. Let us consider a system with two processors. If we have three processes that use different resources, then the three processes may be executed simultaneously, but, actually, we have only two processors. In order to consider that there are only two available processors, we need to consider two resources, `cpu1` and `cpu2`, each one representing one processor (in ACSR it is not allowed to have two units of a resource). As an example we can consider the following three processes (in ACSR syntax): $P = \{(cpu1, 1)\}^1 : NIL$, $Q = \{(cpu2, 1)\}^1 : NIL$ and $R = \{(cpu1, 1)\}^1 : NIL$. Then we have that in the first step P and Q or R and Q may evolve simultaneously, but not P and R because they share the resource `cpu1`. In other words, resources are statically assigned to processes. This does not occur in our model where resources are dynamically assigned to processes.

A different approach is considered in [17] where a process algebra for the management of resources in concurrent systems (`PARM`) is presented. In that paper a heterogeneous resource context and a resource trading policy (based on utility functions) are introduced. The main idea is that several processes can exchange resources in order to improve the performance of the global system, provided that none of them can get worse after the exchange. Although this is a very interesting work, it is not very closed to our approach because no quantification about the performance of a system is considered. A related approach is considered in [18] where an optimal resource allocation in multi-class networks scheme is presented.

The rest of the paper is organised as follows. First, we present the syntax of **BTC** and some basic notations. Next, the operational semantics of the language is defined by means of a set of inference rules. Building on it, we define an algorithm for performance evaluation and illustrate it with a simple example of how a system can be specified and how the performance algorithm works. Then we present our conclusions and outline some lines for current and future work.

2 Introducing the Language BTC

In this section we introduce the process algebra **BTC** by means of its syntax and an intuitive interpretation of the operators.

Definition 1. *Let $\mathcal{A}ct_T$ be a finite set of timed actions and $\mathcal{A}ct_U$ a finite set of untimed actions, $\mathcal{A}ct_U \cap \mathcal{A}ct_T = \emptyset$. The syntax of* **BTC** *is defined by the following BNF expression:*

$$P ::= stop \mid a.P \mid \; <b, \alpha>.P \mid P \;\oplus\; P \mid P \;+\; P \mid P \parallel_A P \mid recX.P$$

where $A \subseteq \mathcal{A}ct_U$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, and $\alpha \in \mathbb{N}$, where $\mathbb{N}$ represents the set of natural numbers. Furthermore we assume a set of process variables Id ranged over by X, X', a set of processes $\mathcal{P}$ ranged over by P, Q, R, and a set of actions $\mathcal{A}ct = \mathcal{A}ct_U \cup \mathcal{A}ct_T$.

Let us now informally describe the interpretation of these operators. Later we will provide them with an operational semantics, so the meaning of each operator and the relationship among them will be completely formalised.

stop. This represents a deadlock, that is, no action can be executed.

Prefix. This is the classical prefix operator. It will we use to represent a process P prefixed by an untimed action. As usual, the process $a.P$, once the action a has been executed, behaves like the process P. Mainly, we will use the (untimed) prefix operator to represent a possible synchronization of a process.

Timed prefix. The classical prefix operator has been enriched with information about the amount of time the action b takes in its execution (the value α). The process $<b,\alpha>.P$, once the action b has been executed, behaves like the process P.

Internal choice. This is essentially the classical internal choice operator. Given two processes P_1 and P_2, $P_1 \oplus P_2$ represents the process that behaves like P_1 or like P_2 as the result of an internal decision of the system.

External choice. This is also interpreted in the usual way. Given two processes P_1 and P_2, $P_1 + P_2$ represents the process that behaves like P_1 or like P_2 as the environment requests.

Parallel composition. $P_1 \parallel_A P_2$ represents the parallel execution of the processes P_1 and P_2, where they synchronize on the actions in A. We will denote by $P_1 \parallel P_2$ the parallel execution of P_1 and P_2 without synchronization, that is, $P_1 \parallel_{\{\}} P_2$.

Recursion. The process $recX.P$ represents the classical recursion operator where occurrences of X are substituted by $recX.P$. This operator allows us to define infinite behaviours.

During the rest of the paper we will use the following notation to deal with multisets (bags). We will use the convention that $B = \{2.x_1,\ 3.x_2,\ 5.x_3\}$ represents the multiset defined over the set $X = \{x_1,\ x_2,\ x_3,\ x_4\}$ by:

$$B : \{x_1,\ x_2,\ x_3,\ x_4\} \longrightarrow \mathbb{N}$$

$$B(x_1) = 2,\ B(x_2) = 3,\ B(x_3) = 5,\ B(x_4) = 0$$

We will use capital letters B, B_1, B_2, C, C_1, C_2 to denote multisets, while capital letters such us A, A_1, A_2 will represent sets. Both the empty multiset and the empty set will be represented by $\emptyset$. We will also use the following operations:

- $B_1 \cup B_2$: Union of multisets.
- $|B|$: Number of actions in the multiset, that is, $|B| = \sum_{x \in X} B(x)$.
- $B \cap A$: Multiset obtained by taking from B the actions belonging to A, that is,

$$(B \cap A)(x) \ = \begin{cases} B(x) & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases}$$

Definition 2. *Let $\mathcal{B}(X) = \{B \mid \forall x \notin X, B(x) = 0\}$ be the set of multisets over the set X. A timed bag is a pair (B,α), where $B \in \mathcal{B}(X)$, and $\alpha \in \mathbb{N}$ is the duration of every element in B. The set of timed bags over the set X, denoted by $TB(X)$, is defined as $TB(X) = \{(B,\alpha) \mid B \in \mathcal{B}(X) \wedge \alpha \in \mathbb{N}\}$.*

Intuitively, a timed bag relates a multiset to a unique natural number that represents the time every action (element) in the multiset needs to finish its execution.

3 Operational Semantics

In this section we present an operational semantics. By means of this mechanism, we provide the language operators with a meaning and an accurate interpretation by describing how a process is able to turn into another. This evolution is represented by using labelled transition systems.

We will have that *transitions* are triples $(P, Q, (B, \alpha))$. We will usually denote transitions by $P \xrightarrow{B,\alpha} Q$.

Intuitively, the meaning of the previous transition is that the process P executes the actions belonging to the multiset B and then it behaves like the process Q. The execution of each action belonging to the same multiset takes α units of time.

In order to consider internal (non observable) evolutions of a process, we consider a new kind of action, $\tau \notin \mathcal{A}ct$, which represents an internal action. This action is used to define the evolution of the internal choice of two processes, and it has duration 0.

In Table 1 we can find the rules of the operational semantics. We assume that $\alpha, \alpha' \in \mathbb{N}$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, $B_i \in \mathcal{B}(\mathcal{A}ct \cup \{\tau\})$ and $B' \in \mathcal{B}(\mathcal{A}ct_U)$. We also consider a constant $\mathcal{N}$ that represents the amount of processors/resources at disposal.

The rule $R1a$ is the classical one for the prefix operator. The rule $R1b$ states that given a process P and an action b with an execution time α, b is firstly executed and, after α units of time, the process behaves like P. In rule $R1c$ a partial execution of the timed action b is allowed, i.e., action b is executed for α' units of time and the remaining time $(\alpha - \alpha')$ will be performed later on.

The rule $R1c$ allows a prefix operator transition to be split into any number of consecutive transitions. This rule is very similar to the rule **ActT** presented in [15]. The only difference is that, in that case, a dense time domain is considered and, consequently, an infinite number of transitions are derived. In our model this does not occur because we work in a discrete time domain.

The rules $R2$ and $R3$ are the classical ones for the internal and external choice operators.

The rule $R4$ represents the basic synchronization mechanism. We only consider those synchronization actions that may be performed by processes P and Q. As synchronization actions are untimed actions, the whole process evolves in 0 units of time to $P' \parallel_A Q'$.

The rule $R5a$ captures the simultaneous execution of, at most, $\mathcal{N}$ no synchronization actions. Although we have considered as premises that both processes may perform different timed bags with the same time α, this does not represent a loss of generality because applying rule $R1c$ we can split transitions in order to have the same time in both premises.

The rules $R5b$ and $R5c$ are similar to $R5a$ but consider that only one of the two processes (P or Q) evolves.

Finally, the rule $R6$ captures the semantics for recursion in the usual fashion.

Table 1. Operational Semantics

R1a) $$\frac{}{a.P \xrightarrow{\{a\},0} P}$$

R1b) $$\frac{}{(b,\alpha).P \xrightarrow{\{b\},\alpha} P}$$

R1c) $$\frac{0 < \alpha' < \alpha}{(b,\alpha).P \xrightarrow{\{b\},\alpha'} (b,\alpha-\alpha').P}$$

R2a) $$\frac{}{P \oplus Q \xrightarrow{\{\tau\},0} P}$$

R2b) $$\frac{}{P \oplus Q \xrightarrow{\{\tau\},0} Q}$$

R3a) $$\frac{P \xrightarrow{B,\alpha} P'}{P + Q \xrightarrow{B,\alpha} P'}$$

R3b) $$\frac{Q \xrightarrow{B,\alpha} Q'}{P + Q \xrightarrow{B,\alpha} Q'}$$

R4) $$\frac{P \xrightarrow{B',0} P' \wedge Q \xrightarrow{B',0} Q' \wedge B' = B' \cap A \neq \emptyset}{P \parallel_A Q \xrightarrow{B',0} P' \parallel_A Q'}$$

R5a) $$\frac{P \xrightarrow{B_1,\alpha} P' \wedge Q \xrightarrow{B_2,\alpha} Q' \wedge B_i \cap A = \emptyset \wedge |B_1| + |B_2| \leq \mathcal{N}}{P \parallel_A Q \xrightarrow{B_1 \cup B_2,\alpha} P' \parallel_A Q'}$$

R5b) $$\frac{P \xrightarrow{B,\alpha} P' \wedge B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P' \parallel_A Q}$$

R5c) $$\frac{Q \xrightarrow{B,\alpha} Q' \wedge B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P \parallel_A Q'}$$

R6) $$\frac{P\{recX.P/X\} \xrightarrow{B,\alpha} P'}{recX.P \xrightarrow{B,\alpha} P'}$$

Definition 3. *The operational semantics of* **BTC** *is defined as the least multiset of transitions we can derive by using the rules in Table 1.*

4 Performance Evaluation Algorithm

With the formal resource-aware model that we have just defined, the timed characteristics of the system have been captured. Now we are concerned with defining an algorithm which allows us to estimate the minimum time needed to reach a given state.

By applying the rules of the operational semantics, we build a transition graph where we can abstract the information about actions and consider only the information about time (duration of actions). This graph is a weighted directed graph, where weights are always positive numbers, and finding the shortest path from the initial node is the problem we want to solve, problem which is solved by using Dijkstra algorithm.

Let T_{S_i} be the expected time to evolve from the *initial state* S_0 to a state S_i, and $In(S_i)$ the set of predecessor nodes of S_i. For every state $S_j \in In(S_i)$ an edge labelled by α_{ji} which represents the time to evolve from S_j to S_i exists.

Then, T_{S_i} may be obtained by computing recursively the following equation

$$T_{S_i} = \begin{cases} 0 & \text{if } S_i = S_0 \\ \min\{T_{S_j} + \alpha_{ji} \mid S_j \in In(S_i)\} & \text{otherwise} \end{cases} \tag{1}$$

In order to compute (1), we number the n nodes of the graph from 0 to $n-1$, and we assume it is represented by an adjacency matrix where $cost[i][j]$ is defined as follows

$$cost[i][j] = \begin{cases} \alpha_{ij} & \text{if } S_i \in In(S_j) \\ \infty & \text{otherwise} \end{cases}$$

Finally, the performance evaluation algorithm is shown in Fig. 1.

5 A Simple Example.

In this section, we present a simple example of specification using **BTC** (addition of two matrices). We show the transition graph we obtain by applying the rules of the operational semantics, and we compute the minimum and maximum time needed to finish the operation depending on the size of the matrices and the available processors.

We assume, without loss of generality, that we have the same number of threads/processes as rows (columns) in the matrices to sum. This system can be modelled as follows:

$$\begin{array}{l} \text{System} \equiv P_1 \parallel P_2 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_n \\ P_i \equiv (rd_i, 4).(add_i, 1).(wr_i, 2).stop \end{array}$$

where rd_i is an action meaning "read a row", add_i is "add a row" and wr_i is "write a row".

```
MinTime(int v, int cost[][], int n, int dist[], bool S[]) {
// v: initial node
// cost: adjacency matrix with weights
// n: number of nodes in the graph
// dist: shortest known path from initial node
// S: true if shortest path is known
// big_number: constant which value is a big number
int u, minimum;

for (int i=0; i<n; i++) {
    S[i] = false; dist[i] = cost[v][i];
}
S[v] = true; dist[v] = 0;
for (int num=0; num<n; num++) {
// Choose u from among nodes not in S such that dist[u] is minimum.
    minimum = big_number;
    for (int i=0; i<n; i++)
      if (!S[i] && (dist[i]<minimum))
            minimum=dist[i]; u=i;
    S[u]=true;
    for (int w=0; w<n; w++) // Update distances
        if (!S[w] && (dist[w]>dist[u]+cost[u][w]))
            dist[w] = dist[u] + cost[u][w];
    }
}
```

Fig. 1. Performance evaluation algorithm

Figure 2 shows the transition graph we obtain when we consider two matrices with just three rows in a system with two processors ($\mathcal{N} = 2$). Taking into account the symmetry in the three possible paths from the initial node, just one of them has been depicted. For the sake of clarity, we have always chosen the biggest number of actions we can execute simultaneously.

There are several paths in which the required time to do this addition is minimum. In the transition graph in Fig. 2 we have marked with dotted lines one of them. Every node in this shortest path has a representation in terms of **BTC** as follows:

$N_0 = (rd_1, 4).(add_1, 1).(wr_1, 2)||(rd_2, 4).(add_2, 1).(wr_2, 2)||(rd_3, 4).(add_3, 1).(wr_3, 2)$
$N_1 = (add_1, 1).(wr_1, 2)||(add_2, 1).(wr_2, 2)||(rd_3, 4).(add_3, 1).(wr_3, 2)$
$N_3 = (add_1, 1).(wr_1, 2)||(wr_2, 2)||(rd_3, 3).(add_3, 1).(wr_3, 2)$
$N_9 = (add_1, 1).(wr_1, 2)||(rd_3, 1).(add_3, 1).(wr_3, 2)$
$N_{15} = (wr_1, 2)||(add_3, 1).(wr_3, 2)$
$N_{21} = (wr_1, 1)||(wr_3, 2)$
$N_{23} = (wr_3, 1)$
$N_{26} = STOP$

We observe that the minimum time needed to do this addition of matrices is 11 units of time while if we were to work in a sequential way (or interleaving) we would have needed 21 units of time.

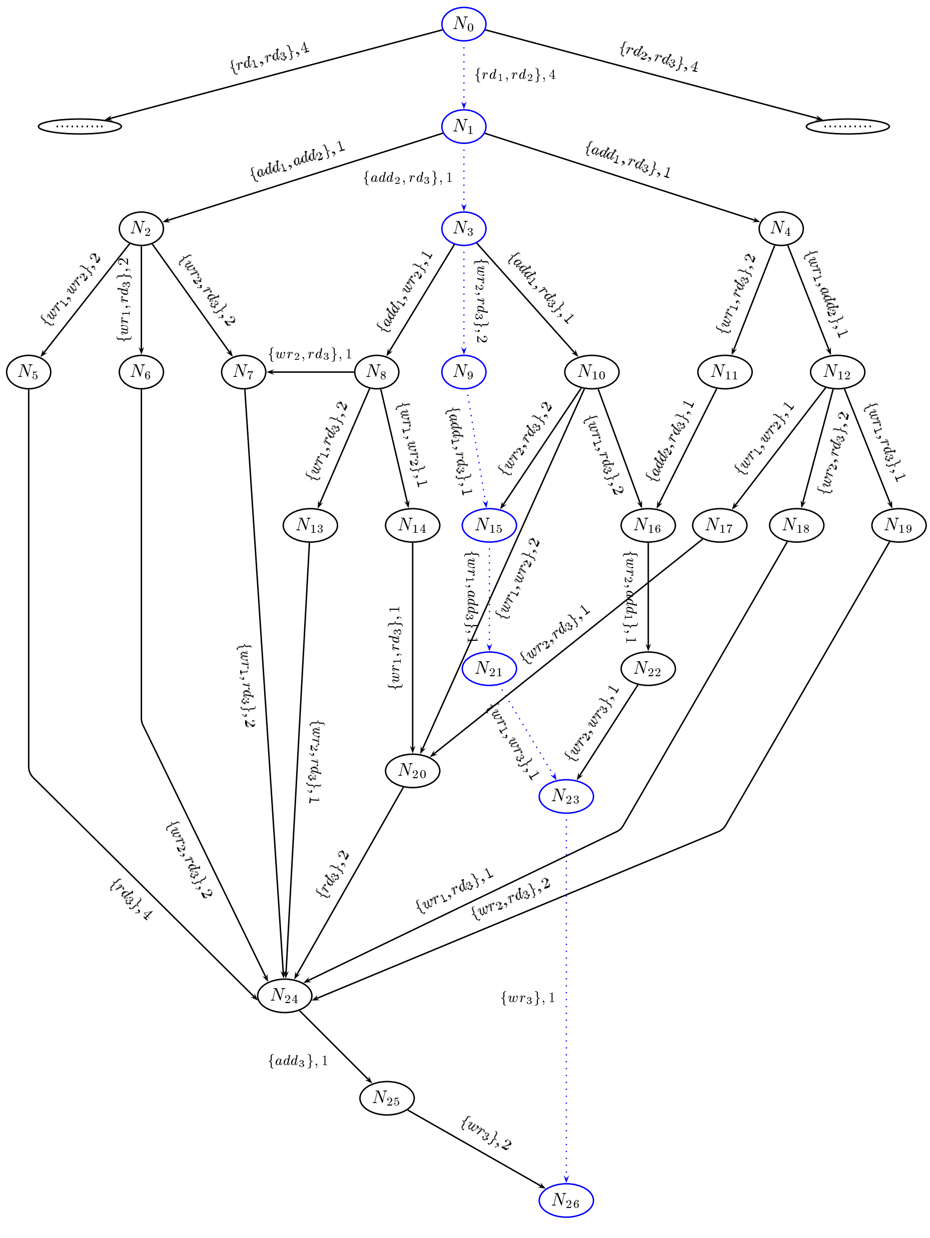

Fig. 2. Transition graph for the addition of two matrices 3x3 with $\mathcal{N} = 2$

The same example of the addition of matrices has been developed for various matrix sizes and different number of processors, obtaining the results displayed in Table 2.

Table 2. Sum of two matrices with various matrix sizes and different number of processors

Matrix size	Processors	T. Min	T. Max
3	2	11	21
3	3	7	21
4	2	14	28
4	3	10	28
4	4	7	28
6	2	21	42
6	3	14	42
6	4	11	42
6	6	7	42
10	2	35	70
10	3	24	70
10	4	18	70
10	5	14	70
10	6	12	70
10	7	11	70
10	8	11	70
10	9	11	70
10	10	7	70

6 Conclusions

In this paper we have presented a timed extension of CSP, called **BTC**, which allows us to evaluate the performance of a system, taking into account that the available resources have to be shared by all the processes. In this first approach to the model we have only considered homogeneous resources.

This algebra can be used for two main different tasks. On one hand, given a fixed number of resources, we are able to evaluate the performance of a system. On the other hand, if we start with some specification, we can find the appropriate number of resources that we need to fulfil some time requirements. We can even calculate the minimum number of resources needed to obtain the best performance of a system. This is particularly important in some applications (e.g. MPEG2 encoding compression algorithm [19]) where the performace of parallel algorithms is *measured* considering different experimental settings (2, 4, 8, ... processors) [20].

Although we have found in the literature different approaches to the problem, none of them seems to cover the main topic of our model: performance evaluation of processes that share resources.

7 Future Work

Our future work is related to extend **BTC** in order to consider heterogeneous resources. In order to define this extension we consider that some of the ideas presented in ACSR [14, 15] could be of interest.

From an application point of view, it would be interesting to link our resource-aware model to a tool for standard timed models making the analysis amenable. If it were impossible we will consider to design a new tool.

Another goal is to model (and analyse) a real system. For this aim, we are considering some parallel algorithms for image and video compression based on Wavelet methods. Compression of visual data is a computationally expensive task where large volumes of data have to be processed; therefore compression algorithms can greatly benefit from the use of parallel systems. In this setting, our algebraic language can be used to find the most appropriate number of needed processors for optimum performance.

Finally, our model suffers from the *state explosion problem*. We think that it could be useful to include some kind of reduction techniques (maybe scheduling) in our model which allows the reduction of the graphs.

Acknowledgements

The authors would like to thank the anonymous referees for their very careful reading and useful suggestions. We would like to thank also Manuel Nuñez for his always right suggestions and comments.

References

1. Costa, G., Stirling, C.: Weak and Strong Fairness in CCS. Information and Computation **73** (1987) 207–244
2. Rutten, J., Zucker, J.: A Semantics Approach to Fairness. Fundamenta Informaticae **XVI** (1992) 1–38
3. Reed, G., Roscoe, A.: A Timed Model for Communicating Sequential Processes. Theoretical Computer Science **58** (1988) 249–261
4. Moller, F., Tofts, C.: A Temporal Calculus of Communicating Systems. In: Proc. of CONCUR'90, LNCS 458, Springer (1990) 401–415
5. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
6. Hermanns, H., Herzog, U., Katoen, J.P.: Process algebra for performance evaluation. Theoretical Computer Science **274** (2002) 43–86
7. Giacalone, A., Jou, C.C., Smolka, S.: Algebraic Reasoning for Probabilistic Concurrent Systems. In: Proc. of Working Conference on Programming Concepts and Methods, IFIP TC 2. (1990)

8. Núñez, M., de Frutos, D., Llana, L.: Acceptance Trees for Probabilistic Processes. In: Proc. of CONCUR'95, LNCS 962, Springer (1995) 249–263
9. Cuartero, F., de Frutos, D., Valero, V.: A sound and complete proof system for probabilistic processes. In: Proc. of 4th International AMAST Workshop on Real-Time Systems (ARTS'97), LNCS 1231, Springer (1997)
10. Hoare, C.: Communicating Sequential Processes. Prentice Hall (1985)
11. Gruska, D.P.: Process Algebra for Limited Parallelism. In: Proc. of Concurrency, Specification and Programming (CS&P'96). (1996) 61–74
12. Gruska, D.P.: Bounded Concurrency. In Chlebus, B.S., Czaja, L., eds.: Proc. of 11th International Symposium on Fundamentals of Computation Theory (FCT'97), LNCS 1279, Springer (1997) 198–209
13. Buchholtz, M., Andersen, J., Løvengreen, H.H.: Towards a Process Algebra for Shared Processors. Electronic Notes in Theoretical Computer Science **52** (2002)
14. Lee, I., Brémond-Grégoire, P., Gerber, R.: A process algebraic approach to the specification and analysis of resource-bound real-time systems (1994)
15. Brémond-Grégoire, P., Lee, I.: A Process Algebra of Communicating Shared Resources with Dense Time and Priorities. Theoretical Computer Science **189** (1997) 179–219
16. Lee, I., Choi, J.Y., Kwak, H.H., Philippou, A., Sokolsky, O.: A Family of Resource-Bound Real-Time Process Algebras. In: Proc. of FORTE'01, Kluwer Academic Publishers (2001) 443–458
17. Núñez, M., Rodríguez, I.: `PAMR`: A Process Algebra for the Management of Resources in Concurrent Systems. In: Proc. of FORTE'01, Kluwer Academic Publishers (2001) 169–185
18. Kalyanasundaram, S., Chong, E., Shroff, N.: Optimal resource allocation in multi-class networks with user-specified utility functions. Computer Networks **38** (2002) 613–630
19. Pelayo, F.L., Cuartero, F., Valero, V., H.Macia, Pelayo, M.L.: Applying Timed-Arc Petri Nets to Improve the Performance of the MPEG-2 Encoding Algorithm. In: Proc. of the 10th IEEE International Conference on Multi-Media Modelling (MMM'2004), IEEE CS (2004) 49–56
20. Olivares, T., Quiles, F., Cuenca, P., Orozco, L., Ahmad, I.: Study of Data Distribution Techniques for the Implementation of an MPEG-2 Video Encoder. In: Proc. of the 11th IASTED Int. Conference on Parallel and Distributed Computing Systems (PDCS'99). (1999) 537–542