A Bounded True Concurrent Process Algebra and Flexible Manufacturing Systems *

M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, Hermenegilda Macia

Departamento de Sistemas Informáticos
Univ. de Castilla-La Mancha
Campus Univ. s/n. 02071. Albacete, Spain.
{carmenr, dcazorla, fernando}@dsi.uclm.es, hmacia@pol-ab.uclm.es

Abstract. This paper deals with the specification and analysis of Flexible Manufacturing Systems (FMS). We use a **timed** process algebra called **BTC** (for Bounded True Concurrency) that we have developed from CSP and which takes into account that the available *resources* in a system have to be shared by all the processes. It is able to consider heterogeneous resources of any type (preemptable and non-preemptable) which makes it suitable for specifying FMS. We show by means of an example that the specifications obtained suit the real systems quite well, are straightforward and deadlock-free. Due to the great interest (and need) in obtaining shorter manufacturing lead times, we also carry out a performance evaluation using an algorithm which allows us to estimate the time required to evolve between states. To the best of our knowledge, this is the first attempt to do this with process algebras, and we think that it is a good way to be able to split up the system into subsystems, which will be easily analyzed without the deadlock problems that researchers in Petri Nets have encountered.

1 Introduction

Globalization has created new demands for manufacturers to produce a wide range of products, which have to be of higher quality at lower prices, and shorten their manufacturing lead times. In an environment of fierce competition, industry can not afford to waste time, material or production capacity. Cost, quality and responsiveness are three requirements that any industry must be aware of if it hopes to survive. This requires a new type of manufacturing system, the flexible manufacturing system (FMS).

Flexible manufacturing systems have been developed with the hope that they will tackle these new challenges - better quality, lower cost and shorter times - by integrating machine tools, robots, material handing, storage systems and computers. Control of these integrated systems results in a new set of problems and challenges, which have been receiving considerable attention from both the academy community and industrial system users.

* This work has been partially supported by CICYT project TIC2003-07848-C02-02, and JCCM project PAC06-0008-6995

In the last few decades, the modelling and the analysis of flexible manufacturing systems has been closely studied by control theorists and engineers. More specifically, in dealing with behaviour analysis, various behaviour models are used that specify the dynamic relationships among the occurrence of events, operations and states of the system. These behaviour-modelling techniques include the classical finite state machines [Gil62], Petri nets [DAJ95, EC97] and rule models [LTP91]. But several challenges are still left in the air because complex FMSs are hard to model and analyze owing to the complexity and the dimension of real FMS.

We propose a new point of view. We specify FMSs by means of a timed process algebra. As far as we know, this is the first attempt to do so but we firmly believe that process algebras are very suitable for specifying complex FMSs for several reasons. First, process algebras allow us to work in a *compositional* way, that is to say, a complex FMS can be split into subsystems which will be easier to analyze, and we can be sure that each property checked in each subsystem appears throughout the whole system. For example, if we join two deadlock-free subsystems we can guarantee that the system obtained is deadlock-free too. It should be mentioned in passing that Petri nets (which are the most widely used approach in formal methods for specifying FMSs) have encountered serious difficulties in avoiding deadlocks. In fact, there exists ample literature offering ways of avoiding deadlocks in complex FMSs, among the most recent papers of which we can find interesting works in [Uza04, LZ04]. This is the main feature which has led us to the conclusion that process algebras are suitable for specifying FMSs but there is also another reasons. The specifications obtained are quite easy to perform because it is fairly straightforward to represent concurrency, source sharing, conflicts, mutual exclusion and non-determinism. We can apply top-down and bottom-up design methodologies and we can have different levels of abstraction in the system. There is a great number of tools which allow qualitative and quantitative analysis from the specification made with a process algebra.

Therefore, our main goal is to make specifications and performance evaluation of Flexible Manufacturing Systems using a Bounded True Concurrent Process Algebra called **BTC** which takes into account that the resources in a system must be shared by all the processes. It was presented in [RCC$^+$04] but it should be pointed out that we extended CSP syntax in order to consider the duration of actions by means of a timed prefix operator. The operational semantics was also extended to consider the context (resources) in which processes are executed. Recently, we enriched **BTC** with a view to considering heterogeneous resources. This means that the system has different kinds of shared resources and, depending on the action to be executed, one or another shared resource must be used. Moreover, we are able to distinguish between preemptable and non-preemptable resources (a preemptable resource can be preempted from a process and reallocated without side effects while with a non-preemptable one problems can arise).

We can also make performance evaluation using the performance algorithm shown in [RCC+04]. This can be used in two different ways. On the one hand, if we have a fixed number of resources (machines, robots, material handing, storage systems or computers) we can ascertain the time needed to evolve from the initial state to the final one (or between states), so we can check different configurations for a flexible manufacturing system before using. On the other hand, if we start with some specification, we can find the appropriate number of resources of any type 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 which is very important if we remember that in industrial systems any waste of time or resources is a loss of money. We will try to show this by means of an example in this paper.

The structure of the rest of this paper is as follows. First, we present an outline of the Flexible Manufacturing Systems in order to situate our work in context. In Section 3 we present some new features in **BTC** and in section 4 the performance algorithm we will use. Then, we specify an example of an assembly cell in FMSs and carry out performance evaluation. We finish with some concluding remarks and our future work.

2 Overview of FMS

In this section, we just try to explain where our work is located within the subject.

Usually, as it is explained in [GV03], manufacturing transformation processes are classified into *continuous* (chemical and oil industries, for instance) and *discrete* (consumer goods and computer industries, for example). According to the type of transformations to be carried out during the manufacturing process, discrete manufacturing systems are classified into *assembly* and *non-assembly* processing. The assembly processes combine several components to obtain a different product, while the non-assembly processes concern the transformation (machining, moulding, painting, etc.) of raw materials.

In this paper, we focus on the specification and performance evaluation of **discrete non-assembly** FMSs.

An FMS is composed of two parts: The physical one, which is composed of the physical resources, and the control part which determines the work system to organize and optimize the production process. As summarized in [SV89], the control part can be split into several levels: *Planning*, *Scheduling*, *Global coordination* , *Subsystem coordination*, *Local control*.

We focus on the problems that arise at the **global coordination level**. This level must have an updated state of the workshop and must also make real-time decisions taking into consideration the state of each resource and the state of the parts being processed. The fact that we begin working in this level does not mean that it is the only stage where we can apply our language. Indeed, we have in mind to broaden the field of our future research, but this stage seems to be a good beginning.

3 The Language BTC

In this section we present the newest approach to **BTC**. The first version can be found in [RCC+04] but now a great number of improvements have been made. We begin by saying that we deal here with three kinds of action: *timed actions* ($\mathcal{A}ct_T$) which use time (and resources if they need); *untimed actions* ($\mathcal{A}ct_U$) which use neither time nor resources (actions for synchronization); and *special actions* ($\mathcal{A}ct_S$), which use resources but do not use time.

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

$$P ::= stop \mid a.P \mid \langle b, \alpha \rangle.P \mid P \ \oplus \ P \mid$$
$$P \ + \ P \mid P \parallel_A P \mid recX.P$$

where $A \subseteq \mathcal{A}ct_U$, $a \in (\mathcal{A}ct_U \cup \mathcal{A}ct_S)$, $b \in \mathcal{A}ct_T$, and $\alpha \in \mathbb{N}$, $\mathbb{N}$ 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 $\mathcal{A}ct = \mathcal{A}ct_U \cup \mathcal{A}ct_T \cup \mathcal{A}ct_S$.

Let $c \in \mathcal{A}ct_S$ be a *special action*, which is used to deal with non-preemptable resources which must be requested and released. So, for each non-preemptable resource we have an action c to request the resource and the corresponding *conjugate* action $\widehat{c} \in \mathcal{A}ct_S$ which is executed when the resource is released.

Furthermore, we extend this syntax with a view to representing both the actions that use the shared resources and amount of resources the system has at its disposal.

In the system we can find $m \in \mathbb{N}$ different types of shared resources. For each of these types we define a set Z_i consisting of all the actions which require for their execution at least one of the shared resources of this type i. If the resources are preemptable, the actions in Z_i will be timed actions and we denote with the number of different types of shared resources available in the system and $x_i \in \mathbb{N}$ the number of actions in the set Z_i. So, each set Z_i for preemptable resources is defined as following:

$$Z_i = \{b_1, \ b_2, \ ..., \ b_{x_i}\}$$

On the other hand, if the resources are non-preemptable, we can find just two actions in its set Z_i. The action c to request the resource and the conjugate action $\widehat{c}$ to release it. For the sake of clarity, we have preferred to include in Z_i just actions c and assumed that $\widehat{c}$ takes part too. Therefore, for non-preemptable resources Z_i is:

$$Z_i = \{c_i\}$$

Now, we put together all these Z_i sets and define the set Z as following:

$$Z = \{Z_1, \ Z_2, \ ..., \ Z_m\}$$

where Z is composed of all the set Z_i generated for each different type of shared resource. Next, we consider

$$\mathcal{N} = \{N_1, \ N_2, \ ..., \ N_m\}$$

where $N_i \in \mathbb{N}$ represents the amount of shared resources of type i available in the system.

By these means, we have extended our syntax in order to recollect all this information, that is, what actions need which shared resource and the number of resources of each type in the system. Now, a process is denoted as:

$$\{\![P]\!\}_{Z,\mathcal{N}}$$

By incorporating these changes, our time process algebra **BTC** is able to model processes which need to use different types of resources in their execution (heterogeneous resources), deal with preemptable and non-preemptable resources and take into account the number of all resources available in the system at any time. This is an important step since we are able to model any kind of delay which can appear in a system, i.e., we are able to deal with delays related to the synchronization of processes and delays related to the allocation of resources. As a result, we come closer to our goal of modelling more complex FMSs.

3.1 Operational Semantics

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

We will take *transitions* to be triples $((P,\mathcal{N}),(Q,\mathcal{N}),(B,\alpha))$. We will usually denote transitions by $(P,\mathcal{N}) \xrightarrow{B,\alpha} (Q,\mathcal{N}')$.

It can be seen that any process comes with $\mathcal{N}$ which, as we have shown in the previous section, is the set representing the amount of shared resources of any type at our disposal in the system. We need this information because we are dealing with non-preemptable resources, and we need to know the availability of a resource when it is requested.

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

The values in $\mathcal{N}$ do not change unless a non-preemptable resource has been requested or released. Here one might ask why they do not change when preemptable resources are involved. This will become clear in the operational semantics, but a brief explanation can help. When an action needs a preemptable resource for its execution, this is used for just as long as the action lasts, so, after the execution, the values in $\mathcal{N}$ are the same. It works even if the action has been split, since preemptable resources can be preempted from a process and reallocated without side effects.

In order to consider internal (non observable) evolutions of a process, we will 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 has a duration 0.

In Table 1 we find the rules of the operational semantics. We assume that $\alpha, \alpha' \in \mathbb{N}$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, $c \in \mathcal{A}ct_S$, $B_i \in \mathcal{B}(\mathcal{A}ct_T \cup \mathcal{A}ct_U\})$ and $B' \in \mathcal{B}(\mathcal{A}ct_U)$.

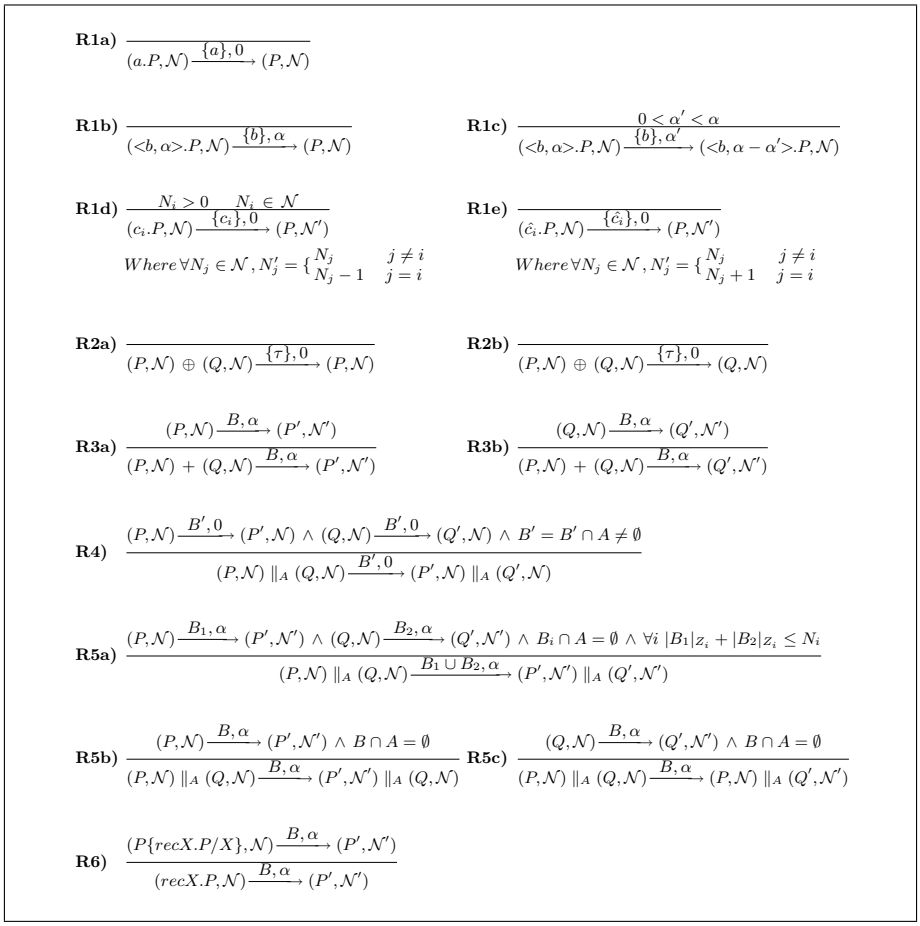

Table 1. Operational Semantics

The rule $R1$ represents the prefix operator, which we have split in order to consider the different types of actions. The rule $R1a$ is the classic one for the prefix operator when the action neither takes time nor resources (in our language *untimed actions*) and here we include extra information about the availability of resources in the system ($\mathcal{N}$). The rule $R1b$ states that, given a process P and a *timed action* b with an execution time α, b is first 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. Note that $\alpha \in \mathbb{N}$, so it is important to underline that this rule does not generate infinite transitions. 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 [BGL97] with the very important difference that, in the latter 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.

Finally, the prefix operator rules for *special actions* are $R1d$ y $R1e$; the first for requesting a non-preemptable resource and the second for releasing it. In rule $R1d$ a *special action* which needs a resource of type i (c_i) for its execution will be executed only if there is some resource of this type at its disposal ($N_i > 0$). The action represents the fact of allocating a resource, so after this the set $\mathcal{N}$ must change and show that there is one less resource of the specified type. When the resource is released the rule $R1e$ is used and modifies the value of N_i, adding one unit.

The rules $R2$ and $R3$ are the classic ones for the internal and external choice operators with additional information about the time an action needs for its execution and the number of resources at disposal.

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 no synchronization actions with the restriction that, for each type of resource, N_i actions, at most, can be executed simultaneously. These are actions which need a shared resource of this type for their execution.

Although we have taken as premises that both processes may perform different timed bags with the same time α, this does not represent a loss of generality because, by 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.

4 Performance Evaluation

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 making performance evaluation: we want to be able 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 weight-directed graph, where weights are always positive numbers; and the problem we want to solve is finding the shortest path form the initial node. We solve this problem by using a Dijkstra algorithm which can found in [RCC+04].

5 Example: an assembly cell

We begin dealing with *flexible cells*. Two or more elements are considered a flexible cell and two or more cells are considered a flexible manufacturing system.

In this section we specify a typical flexible cell (an assembly cell) by means of **BTC** and try to prove the utility of our approach and more future possibilities of use.

As we have said, this example deals with a very useful type of cell: an assembly cell which is included in almost any FMS. Our cell consists of three machines and a robot. Machine *M1* produces parts of type "A" and *M2* produces parts of type "B". Later, machine *M3* assembles one part of each type to obtain the final product that leaves the assembly cell. The movement of parts inside the cell is carried out by the robot which, obviously, works in mutual exclusion (Figure 1).

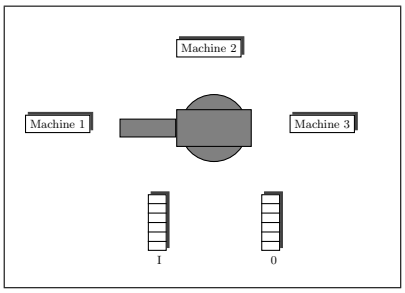

Fig. 1. Example: an assembly cell

In a first approach, each machine in the system can produce/deal with just one part at a time, although we will show later on that it is not a restriction but a simplification for a better understanding of this first example. In the same way, we have described a system with just one machine of any type, but this restriction will not be necessary in future.

In this example, we can find several important characteristics that are common to almost all FMSs. Let us now look at it. It has sequential relations because some events must occur in a sequential way. For example, a part in machine B cannot be unloaded until the machine has finished its work. But it also has concurrency, the processing in machine $M1$ can be made concurrently or in parallel with the processing in machine $M2$. Note that we have said concurrently or in parallel, because our model is able to easily model the two alternatives, something which is worth taking into consideration.

In our example-system, some events can occur in an asynchronous way. Obviously, the end of the processing of a part in machine $M1$ is asynchronous with the end of the processing in machine $M2$. But, at the same time, we may find that synchronization is necessary before machine $M3$ begins its processing.

In this first system, we have a robot which moves parts among all the machines and the input/output conveyors. At any moment, a conflict can arise if more that one machine/conveyor needs this robot at the same time, so a decision must be taken which is made using a decision strategy. For example a decision strategy may be the following one: The demands to the robot are served following a First Come-First Serve (FCFS) strategy.

And the last characteristic, although no less important, that we can find in this example is mutual exclusion. The robot will be considered as a resource that can be used for just one process at a time. This must be used in mutual exclusion.

Now, we have seen that this toy-example encloses many important characteristics of FMSs. Its analysis is representative enough to show the power of **BTC** algebra. Let us see our example in detail.

Our algebra can deal with heterogeneous resources (different types of resources) and in this system we have four types of them, i.e., three machines ($M1$, $M2$ and $M3$) and a *robot*. The resources that modelled the machines are preemptable. For example, if machine $M2$ has received more than one part, it can process them in concurrency. As it is assumed (just for the sake of clarity) that any machine can process just one part at a time, this means that the execution of different concurrent processes will be made in a interleaving way. This means that we can preempt a resource of type $machineM2$ and reallocate it later on without side effects.

In contrast, the resource *robot* is non-preemptable. It is clear that, once the robot has played a part and begun the movement in one direction, it cannot be preempted without problems.

Therefore,in the system we have four resources ($M1$, $M2$, $M3$ and *robot*) and for each one we define a set consisting of the actions that need (at least) one resource of this type for their execution. For preemptable resources ($M1$, $M2$ and $M3$) we define as follows:

$Z_1 = \{process\_m1\}$ $\quad Z_2 = \{process\_m2\}$
$Z_3 = \{process\_m3\}$

which include the actions executed in any machine. However for the non-preemptable resource, *robot*, its set is a bit different. We need to keep the robot during the whole time the movement lasts. So, we need to request and release it with the intention of working in mutua exclusion. Then, the actions that need the robot are *request_robot* and *release_robot* which we named as r_robot and its conjugate $\widehat{r\_robot}$. But, as we have said previously, we include just one of them in the set because the conjugate is taken for granted. From this:

$$Z_4 = \{r\_robot\}$$

We have mentioned that, for this example, we consider that we have just one resource of any type, so $\mathcal{N} = \{1,1,1,1\}$. On such a basis, we can model the system considered as seen in Figure 2.

It is assumed that there are as many raw parts as necessary in the input conveyor, so the system starts moving a raw part to a machine M1 and another one to machine M2. After this, the $ASSEMBLE$ process starts. This specification is clear enough and we think that no more explication is required.

5.1 Performance evaluation of an assembly cell

With the formal resource-aware model we have defined, the timed characteristics of the system have been captured. Now, we try to solve a performance optimization problem relevant to the minimization of the maximum completion time. We

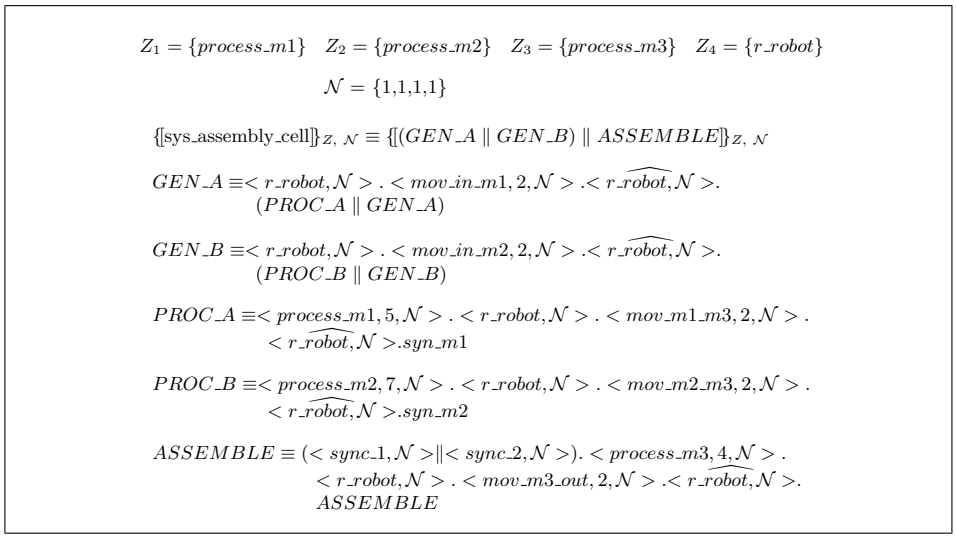

Fig. 2. Specification for the assembly cell example

have chosen this item because it is well know that in industry any waste of time is a waste of money.

Using our performance algorithm, we can find the minimum number of resources needed to obtain a required performance of a system or evaluate the performance of an established system to research any improvement. In this example, we already have an established system, which we are going to study for the purpose of finding out if it can be improved, i.e., if a better completion time can be achieved.

Starting from the operational semantics, we are able to make a state transition graph. Then, we use the performance algorithm shown in section 4 to estimate the time required to evolve between states. More preciously, we calculate the time required to evolve from the initial state to the final one, so we can ascertain the number of completion parts per unit of time (throughput).

Previously, we have said that conflicts are presented in the system which leads to non-determinism. This means, the resource *robot* is requested by different processes and it is necessary to decide which one is served first. To avoid, as far as possible, this non-determinism we use some decision strategies. We employ an FCFS strategy in the use of the robot, but it can happen that in the same unit of time two different processes request the robot, so that some kind of *priority* must be settled. Let us begin by giving to the process B (process which deals with parts of type B) more priority to enter in the FCFS queue.

For such a premise and by means of the performance algorithm, the result is that the system needs 13.06 units of time to process 100 raw parts, that is, it has a throughput of 7.65. But with any change in the physical requirements of the system, if we change just the priority so that processes for parts A now have

higher priority the result is that the system takes 11.16 units of time in processing 100 raw parts with a throughput of 8.96. And when the highest priority is given to process in charge of the assembly, then the time needed is 11.08 units and the throughput is 9.02. We can observe that by just changing the decision strategy we can obtain very different results in this little cell. This gives us a glimpse of the huge repercussion on an entire system.

But if we have a look at the system we may think that the shared resource *robot* seems to cause a bottleneck and that, if we had two robots the system throughput would be better. We use our specification and performance algorithm and check what happens in this supposition. The results are clear: With two shared resources *robot* the system takes 11.06 units of time to process 100 raw parts with a throughput of 9.04. Now, we will turn to the system designer. With two robots the system will have a similar throughput to that with only one if the decision policy is correct. Is it worth using a new robot? In this case it seems clear, but sometimes more parameters have to be taken into account.

Let us see more examples. With the same specification we now suppose that the robot is the slowest resource and takes 9 units of time in its movements. In this case, the results obtained with the performance algorithm seem to be more relevant for taking a decision. With one shared resource *robot*, the system needs 45.18 units of time to process 100 raw parts (throughput 2.21) while with two robots it uses just 24.97 units (throughput 4.00) for the same number of parts. Here the number of parts processed per unit of time is nearly double if the system has two robots instead of one. Taking into account economic consideration, this result would strongly suggest which decision is the best one to take.

6 Conclusion and Future Work

In this paper, we have presented a new point of view in the use of formal methods to specify and analyze FMSs in a formal framework. We have presented a timed process algebra (**BTC**) that considers the context (resources) in which processes are executed, that is, it takes into account that the number of resources available in a system is limited. **BTC** is able to deal with different types of resources at the same time (heterogeneous) and makes a clear distinction between preemptable and non-preemptable ones. This language has become powerful enough to model nearly any type of system and we have decided to apply it in a field with a great numbers of challenges: FMSs.

The complexity and the dimension of real FMSs makes them very difficult to cope with. We have adopted a bottom-up approach and our work uses flexible cells composed of two or more elements.

Here, we specify a very useful type of cell, namely, an assembly cell, which is included in almost any FMS, and which solves a performance optimization problem relevant to the minimization of the maximum completion time. We make it using our performance algorithm over the state transition graph obtained by applying the rules of the operational semantics.

We obtain important results and prove that it is very useful to be able to check the throughput of different compositions before being settled or for future improvements.

In the example we have modelled, every machine is supposed to have input/output buffers with capacity for infinite parts. In a real FMS, every machine usually has input/output buffers with a limited capacity so, at present, we are working to include new resources of type $buffer$ and checking their effects in the throughput of the system.

The complexity of real FMSs and, as a result, the explosion of the number of states in the transition graph due to the necessity of representing all possible states seems to discourage the application of this approach to the modelling and analysis of FMS as a whole. But a promising research direction seems to be its use in a decentralized fashion, that is, we can decompose an FMS into flexible manufacturing cells and specify and make performance evaluations on these subsystems.

References

[BGL97] P. Brémond-Grégoire and I. Lee. A Process Algebra of Communicating Shared Resources with Dense Time and Priorities. In *Theoretical Computer Science* ***189***, pages 179–219, 1997.

[DAJ95] Alan A. Desrochers and Robert Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems. Modeling, Control and Performance Analysis.* IEEE Press, 1995.

[EC97] J. Ezpeleta and J.M. Colom. Automatic synthesis of Colored Petri Nets for the control of FMS. In *IEEE Transactions on Robotics and Automation*, pages 13(3):327–337, June 1997.

[Gil62] A. Gill. *Introduction to the Theory of Finite-state Machines.* McGraw-Hill New York, 1962.

[GV03] Claude Girault and Rdinger Valk. *Petri Nets for Systems Engineering. A Guide to Modelling, Verification, and Applications.* Springer, 2003.

[LTP91] P. Loucopoulos, B. Theodoulidis, and D. Pantazis. Business rules modelling: conceptual modelling and object-oriented specifications. In *Object-Oriented Approach in Information Systems*, pages 323–42, Amsterdam, 1991. F. Assche, B. Moulin and C. Rolland.

[LZ04] Z. Li and M.C. Zhou. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Trans. Sys. Man and Cyber. Part A*, 34:38–51, 2004.

[RCC+04] M.C. Ruiz, D. Cazorla, F. Cuartero, J.J. Pardo, and H. Macia. A Bounded True Concurrency Process Algebra for Performance Evaluation. In *Proc. of the 1st European Performance Engineering Workshop (EPEW'04), LNCS 3236*, pages 143–155, Toledo, Spain, October 2004. Springer.

[SV89] M. Silva and R. Valette. Petri Nets and Flexible Manufacturing. *Advances in Petri Nets*, 424:374–417, 1989.

[Uza04] M. Uzam. The use of Petri net reduction approach for an optimal deadlock prevention policy for flexible manufacturing systems. *Int. F. Adv. Manuf. Tech.*, 23:204–219, 2004.