PCP-based Solution for Resource Sharing in Reconfigurable Timed Net Condition/Event Systems

Mohamed Oussama Ben Salem[1,2,3], Olfa Mosbahi[1,3], Mohamed Khalgui[1,3]

[1]LISI Laboratory, INSAT, Tunis, Tunisia
[2]Tunisia Polytechnic School, Tunis, Tunisia
[3]University of Carthage, Tunisia
bensalem.oussama@hotmail.com, olfamosbahi@gmail.com,
khalgui.mohamed@gmail.com

Abstract. This research paper deals with reconfigurable discrete-event control systems which are distributed on networked devices to execute control software tasks with shared resources. A reconfiguration scenario is also assumed to be any addition, removal or update of tasks and shared resources. Nevertheless, the addition of resources at run-time can bring the system to a blocking problem that is sometimes unsafe. We propose to enrich the formalism Reconfigurable Timed Net Condition/Event Systems (R-TNCES) with the well-known protocol Priority Ceiling Protocol (PCP) in order to model and verify safety reconfiguration scenarios. The paper is applied to a study case illustrating our contribution.

Keywords: Distributed Control System, Reconfiguration, Shared Resource, PCP, Verification.

1 Introduction

Reconfiguration is the qualitative change in the structure, functionality, and algorithms of the control systems. This is due to qualitative changes of goals of control, the controlled system or of the environment the system behaves within. Partial failures, breakdowns, or even human intervention may cause such changes. Thus, the development of Reconfigurable Distributed Control Systems (RDCS) is not an easy activity to perform since they should be adapted to their environment under functional and temporal constraints. These systems are distributed on networked devices which execute control tasks with shared resources. The RDCS's devices may actually share physical or logical resources (robots, material movement systems, machining centers, etc.) which may be added or removed according to users' requirements with specific evolutions in the environment.

A reconfiguration scenario is assumed, in [1], to be any addition, removal or update of control tasks to insure the required flexibility of the system and according to well-defined users' strategies. Nevertheless, this definition is incomplete since it does not consider the reconfiguration of resources. We propose, then, a new definition of reconfiguration to deal with the addition and removal of resources shared by tasks.

We note that no one in our community proposed a same. To model and verify reconfigurable systems, the authors in [2] propose a new formalism named Reconfigurable Timed Net Condition/Event Systems (R-TNCES) extending the formalism TNCES [3] and Petri nets to model the different behaviors of the system to be executed after different reconfiguration scenarios. Even though this formalism is original and useful, the authors do not consider the case of shared resources to be adapted at run time. Nowadays, several research works are proposed to manage the problems of shared resources [4] [5]. Two well-known protocols have been selected in our community to encode the management of resources: Priority Inheritance Protocol (PIP) [6] and Priority Ceiling Protocol (PCP) [7]. The second protocol is better than the first since it avoids deadlock problems. Today, several research works are based on these two protocols, but no one consider them for the case of reconfigurable resources in DRCS. Since it is an expressive formalism for adaptive systems, we propose in this paper to enrich RTNCES with the protocol PCP in order to model both tasks and resources. Our goal is to check that any reconfiguration of resources or tasks does not bring the whole architecture to a blocking situation. We propose three modules to specify the behavior of each resource such that the first represents the control of the resource, the second its state and the third its ceiling which is a priority equal to the highest priority of any task that may lock the resource. We also propose two sub-modules for each control task: the first for its control and the second for its state. We use SESA [8] as a model-checker to check the safety of each reconfiguration scenario that can possibly be applied to the system. The Computation Tree Logic (CTL) [9] is used to specify the deadlock property which will be verified on the system. We consider in this paper a case study which illustrates the relevance of our contribution.

The current paper in organized as follows: the next section describes useful preliminaries for the reader to understand our contribution. Section 3 discuses the state of the art. We expose, in Section 4, the case study and detail our problem. Section 5 proposes our contribution, before finishing the paper in Section 6 with a conclusion and an exposition of our future works.

2 Preliminaries

We present in this section the formalism R-TNCES which extends Petri nets and TNCES [3] for the modeling of adaptive control systems. We expose, thereafter, two well-known protocols: PIP and PCP, for the management of resource sharing.

2.1 Modeling formalisms

Timed Net Condition/Event Systems:

A Timed Net Condition/Event System (TNCES) [3] has a modular structure which may be basic or composite. A basic TNCES is an elementary module extending Petri nets by proposing the new concepts of event and condition signals. A composite TNCES can be composed of basic/composite interconnected modules in a hierarchical form. A TNCES, as shown in Fig. 1, is formalized as a tuple in [13] as follows:

TNCES = {PTN; CN; WCN; I; WI; EN; em}

where: (i) *PTN = (P; T; F; K; WF)* is a classic Place/Transition Net, (ii) *CN* $\subseteq$ *(P* $\times$ *T)* is a set of condition arcs, (iii) *WCN: CN* $\rightarrow N^{+}$ defines a weight for each condition arc, (iv) *I* $\subseteq$ *(P* $\times$ *T)* is a set of inhibitor arcs, (v) *WI: I* $\rightarrow N^{+}$ defines a weight for each inhibitor arc, (vi) *EN* $\subseteq$ *(T* $\times$ *T)* is a set of event arcs free of circles, (vii) *em* : *T* $\rightarrow$ *{*$\vee$,$\wedge$*}* is an event mode for every transition (i.e. if $\wedge$ then the corresponding events have to occur simultaneously before the transition firing, else if $\vee$ then one of them is enough).

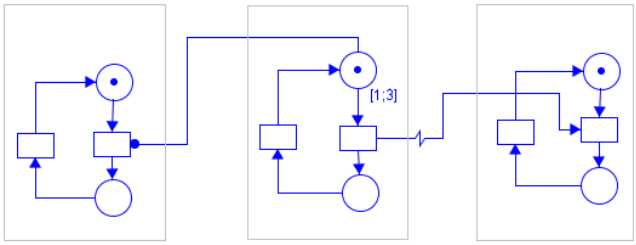

Fig. 1. Example of an TNCES.

Reconfigurable Timed Net/Condition Event Systems:

Reconfigurable Timed Net/Condition Event System (abbr. R-TNCES) is an extension of the formalism TNCES with a specific function of self-reconfiguration. A R-TNCES is formalized in [2] as a structure :

RTN=(B, Rec)

where *B* is the behavior module which is a super set of TNCESs which model different behaviors of the system. The authors in [2] formalize *B* as follows:

B = (P, T, F, W, V , Z, CN, EN, DC)

where: (i) P (resp. T) is a superset of places (resp. transitions), (ii) F $\subseteq$ (P$\times$T) $\cup$ (T$\times$P) is a superset of flow arcs, (iii) W : (P$\times$T) $\cup$ (T$\times$P) $\rightarrow$ {0, 1} maps a weight to a flow arc, W(x, y)>0 if (x, y) $\in$ F, and W(x, y) = 0 otherwise, where x, y $\in$ P $\cup$ T, (iv) CN $\subseteq$ (P $\times$ T) (resp. EN $\subseteq$ (T $\times$ T)) is a superset of condition signals (resp. event signals), (v) DC : F $\cap$ (P $\times$ T) $\rightarrow$ {[l1,h1], ..., [l| F $\cap$ (P $\times$ T) |, h| F $\cap$ (P $\times$ T) |]} is a superset of time constraints on output arcs, where $\forall$ i $\in$[1, | F $\cap$ (P $\times$ T)]|, li, hi $\in$ $\mathbb{N}$, and li < hi, (vi) V : T $\rightarrow$ {Λ, $\bigvee$} maps an event processing mode (AND or OR) for every transition, (vii) Z0= (M0, D0), where M0 : P $\rightarrow$ {0, 1} is the initial marking, and D0: P $\rightarrow$ {0} is the initial clock position.

The module Rec = {r_1, ..., r_m} is defined in [2] as a control module which is a set of reconfiguration functions dealing with the automatic transformation from a TNCES to another. It was proposed for the optimal modeling of reconfigurable discrete event

systems. We propose in this paper to use R-TNCES as a formalism to model reconfigurable DRCS since it expressively allows to add or remove any place or transition.

2.2 Resource sharing protocols

Priority Inheritance Protocol:

Priority Inheritance Protocol (PIP) in real-time computing is a solution to avoid the problems of priority inversion. As introduced in [6], the basic PIP prevents any blocking of higher priority tasks by lower ones. In fact, if a lower priority task blocks a higher one, then it should execute its critical section with the priority level of the higher priority task that it blocks. In this case, we say that the lower priority task inherits the priority of the higher priority task. In PIP, the maximum blocking time (due to a lower priority task) is equal to the length of one critical section and the blocking can occur at most one time for each lock. A PIP operation is summed up in [6] as follows: Let us assume a system to be composed of a set of tasks {T1...Tn} such that (i) Ti and Tk share a resource R ($i \in (1,n)$ and $k \in (1,n)$), (ii) Priority of Ti is lower than Tk's. Then, if Tk is blocked on a semaphore that corresponds to a resource in use by Ti, then Ti immediately inherits the priority of Tk in order to unblock it as soon as possible.

Priority Ceiling Protocol

The Priority Ceiling Protocol (PCP) [7] in real-time computing is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections. In this protocol, each resource R is assigned a priority ceiling Cl(R), which is equal to the highest priority of the tasks that may lock it. A task can acquire a resource only if the resource is free and has a higher priority than the priority ceiling of the rest resources in lock by other tasks. Let us assume a system to be composed of the tasks T1, T2, T3 and T4 (having respectively the increasing priorities 1, 2, 3 and 4) and two resources R and Q: R can be used by T1 and T2 and Q by T1 and T4. Then, C(R)=2 and C(Q)=4. Thus, T2 is blocked if it tries to block R which is free when Q is locked. PCP brought improvements from PIP since it gives guarantees that there is no deadlock and each task in blocked at most the duration of one critical section. However, there is a downside when using PCP; there is more run-time overhead than PIP.

3 State of the art

Nowadays, several research works are proposed to deal with shared resources in control systems. Actually, when dealing with concurrent systems, very complex behaviors may sometimes be faced because of the interaction among the concurrent processes that share resources. Previous studies showed that Petri nets are useful tools to model such systems [10] [11] [12], but they did not deal with the shared resources issue. The work [4] supposed that each sub-system has only one shared resource. Authors of [5] proposed a special class of shared-resource systems with restrictions

by using structural and behavioral conditions on Petri net models. Since we are dealing with reconfigurable resources in DRCS, we affirm that no one in our large community treats this problem. We propose in this paper an original contribution to extend and enrich R-TNCES with PCP to model and verify reconfigurable tasks and resources for the safety of any reconfiguration scenario to be applied on the system.

4 Case study and problem

Let us assume a reconfigurable discrete event system to be composed of two tasks $\mathcal{A}$ and $\mathcal{B}$. We suppose that these two tasks share initially the resources $\mathcal{Q}$ and $\mathcal{R}$ (as shown in Fig. 2) before applying a reconfiguration scenario which will add a new resource $\mathcal{S}$ (to be used by both $\mathcal{A}$ and $\mathcal{B}$). This case was not treated in any related work and forms a new problem dealing with reconfigurable resources. We suppose that $\mathcal{B}$ has the highest priority ($\mathcal{B} > \mathcal{A}$). We suppose that the system is safe before the reconfiguration scenarios. But, once the reconfiguration is applied, a deadlock certainly occurs according to Fig. 3. In fact, $\mathcal{A}$ starts by using $\mathcal{R}$ before being interrupted by $\mathcal{B}$ due to the latter's higher priority. $\mathcal{B}$ is then blocked because it tries to lock $\mathcal{R}$ (P($\mathcal{R}$)) which is hold by $\mathcal{A}$. $\mathcal{A}$ continues progressing until it frees $\mathcal{R}$ (V($\mathcal{R}$)) and $\mathcal{B}$ interrupts it. When $\mathcal{B}$ asks for $\mathcal{Q}$ (P($\mathcal{Q}$)), it is interrupted because $\mathcal{Q}$ is hold by $\mathcal{A}$. $\mathcal{A}$ is lately blocked when it asks for $\mathcal{S}$ (hold by $\mathcal{B}$). A deadlock occurs thus, because $\mathcal{A}$ is waiting for $\mathcal{S}$ while $\mathcal{B}$ for $\mathcal{Q}$. Regarding to situation, we affirm that reconfiguration scenarios can bring the system to blocking situations. The latter can be met if we apply R-TNCES according to [2] to model the tasks A and B as illustrated in Fig. 4. This is proven by applying the CTL formula *AG EX TRUE* which turned out to be false .

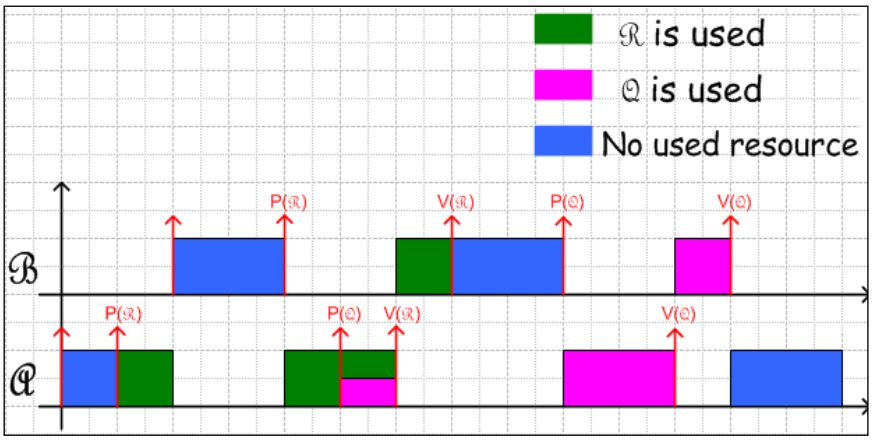

Fig. 2. Behavior of $\mathcal{A}$ and $\mathcal{B}$ before a reconfiguration scenario.

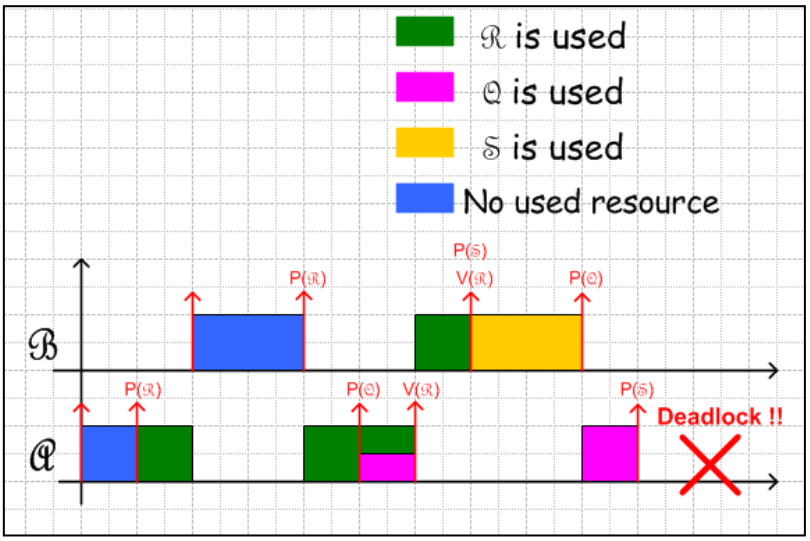

Fig. 3. Behavior of $\mathscr{A}$ and $\mathscr{B}$ after a reconfiguration scenario.

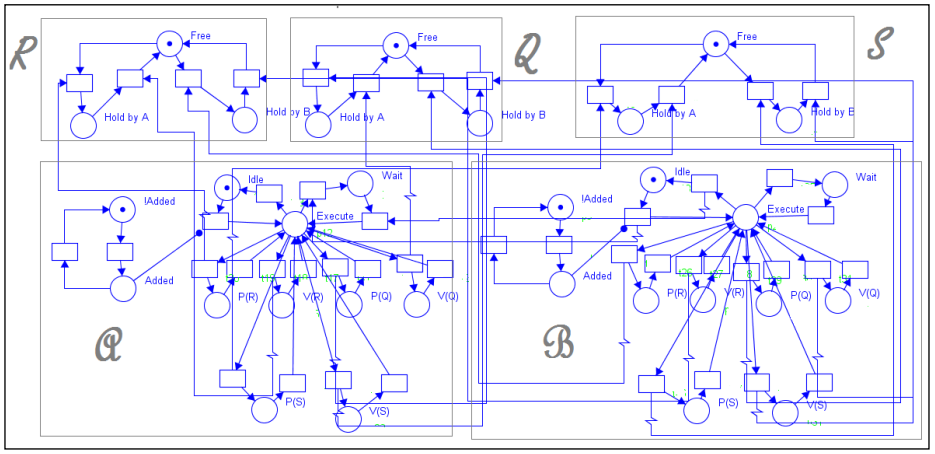

Fig. 4. The illustrative example's R-TNCES.

```
Parsing formula:
AG EX TRUE

The formula is FALSE.
```

Fig. 5. Screenshot from SESA.

This is why we aim in this paper to check the safety of each scenario by enriching R-TNCES with the PCP protocol. We propose in the following sections new patterns to model reconfigurable discrete event systems according to R-TNCES by using PCP.

This contribution is original since R-TNCES is an original formalism for reconfigurable systems, but lacks of useful mechanisms to manage shared resources. Let us remember the reader that in [2] the authors do not consider the eventual addition or removal of resources in the system.

5 Contribution

Since R-TNCES is a useful formalism to model reconfigurable systems, we aim in this paper to propose a new PCP-based extension to this formalism in order to check the system's safety after any reconfiguration scenario dealing with the addition or removal of resources.

5.1 Formalization

We present in this section the formalization of Distributed Reconfigurable Control Systems sharing resources.

DRCS

We assume in the current paper that a DRCS D to be composed of n_1 networked reconfigurable sub-systems sharing n_2 resources. Then we extend the formalization of DRCS in [2] by adding the new set of resources as follows:

$$D = (\sum \text{R-TNCES}, \varpi, \sum M, \sum R)$$

where: (i) $\sum$R-TNCES is a set of n_1 R-TNCES, (ii) ϖ a virtual coordinator handling $\sum M$, a set of Judgment Matrices, (iii) $\sum R$, a set of n_2 shared resources. The virtual coordinator ϖ and $\sum M$ are detailed in [2]. The original parameter here is $\sum R$ which refers to a set of reconfigurable resources.

Shared resources

On the basis of PCP's definition and the flexibility expected from the DRCS, we define a resource R as follows :

$$R = (\text{Rec}, S, \text{Cl})$$

where: (i) Rec (Reconfiguration) indicates whether R is added to the system / $Rec \in \{added, !added\}$. Actually, since DRCSs are reconfigurable, a resource may be activated or deactivated when switching from one mode to another, (ii) S indicates the state of R / $S \in \{free, hold\ by\ a\ task_i\}$, (iii) Cl is used for the ceiling of R.

Control tasks

Based on the expected reconfiguration of the system, we define a task T by :

$$T = (\text{Rec}, S)$$

where: (i) *Rec* (Reconfiguration) indicates whether *T* is added to the system (depending on the selected mode) / R ∈ { *added, !added* }, (ii) *S* indicates the state of *T* / *S* ∈ { *idle, execute, wait, P(R_i), V(R_i)* } and *P(R_i)* means locking R_i and *V(R_i)* unlocking it.

5.2 Modeling

We are interested in this research work in the reconfiguration of both tasks and shared resources. We propose, then, new solutions to introduce PCP in R-TNCES to avoid any blocking problem after reconfiguration scenarios. We propose an R-TNCES model for each resource of ∑R and task of ∑R-TNCES.

Shared resources

We model each shared resource by a R-TNCES as shown in Fig. 6. The latter is composed of three TNCES modeling the resource's reconfiguration (Rec), state (S) and ceiling (Cl). Here is the modeling of a resource R:

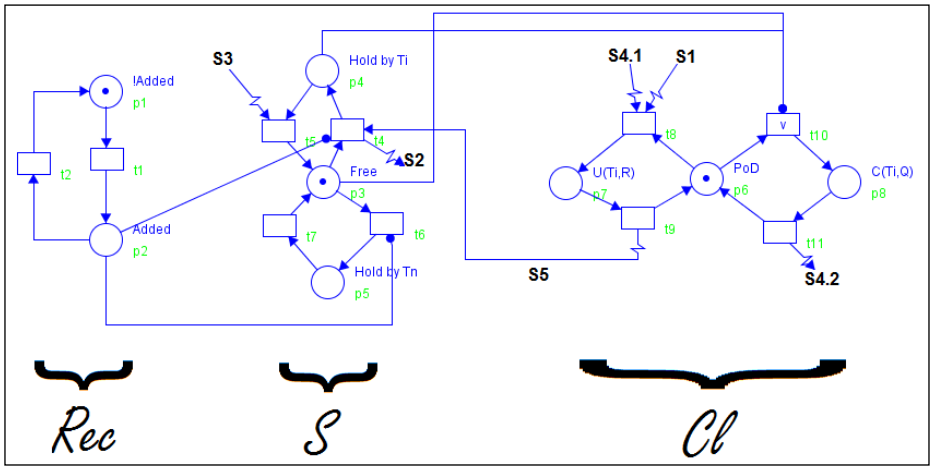

Fig. 6. A shared resource's modeling.

Let's, now, dissect this R-TNCES into 3 TNCESs to analyze the structure of each one thereof and understand the role of each arc and place.

The first TNCES, named Rec and represented in Fig. 7, models the reconfiguration of the resource R. Actually, a DRCS, as its name suggests, is reconfigurable and may work under different modes. It may need, depending on the activated mode, the use of a set of resources. Thus, a resource may be used under some modes but not others. We suggest, therefore, that a resource may be added to a system or removed from it as required from the activated mode. That explains the fact that we have two places in the reconfigurable TNCES: "Added" and "!Added" (not added) respectively represented by p1 and p2. Two conditions leave the place "Added" to join transitions

in the TNCES S (State). This is explained by the fact that a resource can not change status if it is not added to the system.

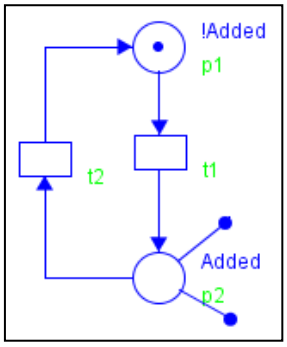

Fig. 7. Shared resource's reconfiguration TNCES.

The second TNCES (illustrated in Fig. 8), S, models the state of the resource R. A resource may be free or hold by a task Ti. Thus, we suggested the places "Free", "Hold by Ti",... and "Hold by Tn" (respectively represented by p3, p4 and p5) where R may be exclusively hold by a task from a set of n different tasks (n is an integer $\in (1,+\infty)$). The conditions leaving the different places are due to the fact that, according to PCP, a task T may hold a given resource if the latter is free and the resources hold by other tasks have a ceiling lower than T's dynamic priority. We will see where those conditions are entering in the next TNCES. The different signals indicated on the figure above stand for: (i) S2: a signal going to Ti and confirming the lock of R by it, (ii) S3: a signal from Ti asking to unlock R, (iii) S5: A signal confirming that all the conditions required to lock R are met (based on PCP).

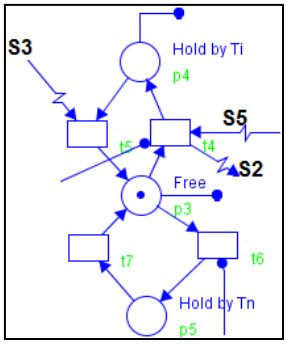

Fig. 8. State TNCES.

The last TNCES (represented in Fig. 9), Cl, models R's ceiling. It contains a place named "PoD" (Point of Decision) and several places type U(task, resource) and type C(task, resource). The two last types stand for:

- *U(T_i, R)*: this place will allow the use of the resource *R* by the task T_i. To switch from "*PoD*" to "*U(T_i, R)*", we need the two events "S1" and "S4.1". S1 is an event coming from the task T_i and asking to lock the resource *R*. S4.1 is actually the S4.2 leaving the transition following *C(T_i, R)*. The latter place will be drawn on the R-TNCES of the resource *Q*. S5 is the signal which we explained in the previous TNCES.
- *C(T_i, Q)*: this place controls the use of *Q* by T_i by checking the PCP's requirements. To reach the place *C(T_i, Q)*, we need, according to PCP, one of the two following conditions: the other resources, whose ceiling are higher than T_i's priority, are free or hold by T_i (hence the OR sign in the transition preceding *C(T_i, R)*). S4.2 on the opposite figure is the S4.1 which will enter the transition preceding *U(T_i, Q)* (drawn of the R-TNCES of the resource *Q*).

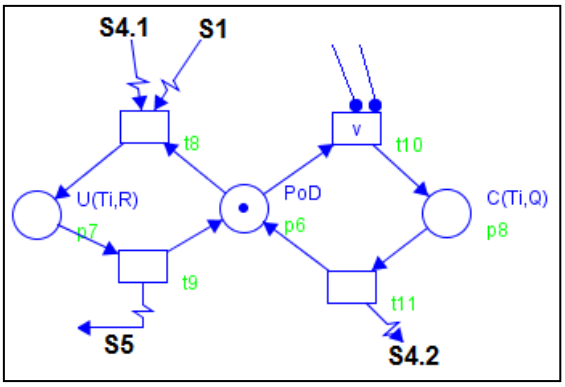

Fig. 9. Ceiling TNCES.

Running example 1 in case study

In our case study, the different resources of our illustrative example (Section IV) have the same modeling since they have the same ceiling and are used by the same tasks. We choose to model the resource Q as shown in Fig. 10.

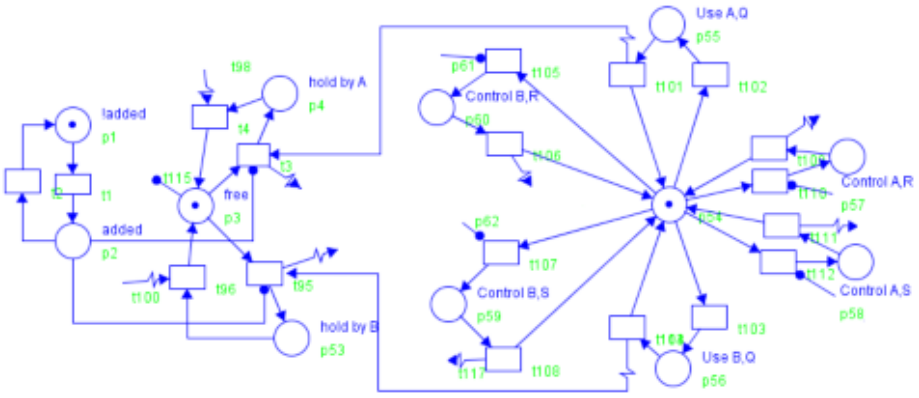

Fig. 10. The resource *Q*'s modeling

The conditions entering the transition preceding C(Task, Resource) are used to guarantee that, when a task T tries to lock a resource, all the other resources -whose ceilings are not lower than the task's priority- are free or hold by T. Thus, we avoid any eventual deadlock and see the relevance of the PCP.

Control tasks

Each task T is modeled by a R-TNCES to be composed of two TNCESs as shown in Fig. 11: the first one is illustrating its reconfiguration (Rec), the second its state (S).

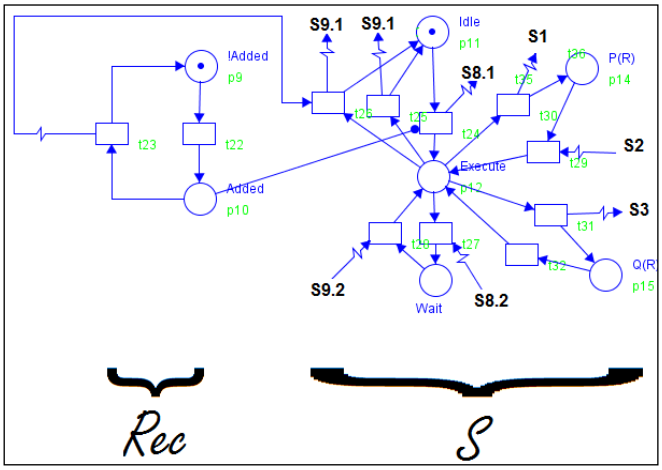

Fig. 11. A task's modeling.

We unpack the R-TNCES in Fig. 11 and analyze the functioning of its two TNCES. Due to the reconfigurable aspect of DRCS, a task may be added or removed to the system according to the activated mode. That's why the T's reconfiguration

TNCES (Rec), represented in Fig. 12, contains two places: "Added" and "!Added" (respectively the places p9 and p10). Once T is removed, the event leaving the transition following "Added" will force the task T to switch from the state "Execute" to the state "Idle". A condition leaving the place "Added" will allow the switching from "Idle" to "Execute".

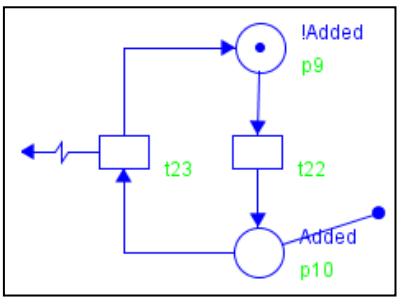

Fig. 12. Task's reconfiguration TNCES.

The second TNCES (Fig. 13) models the state of the task T. It contains the following places: (i) Idle: as its names suggests, the task is idle, (ii) Execute: the task is running, (iii) Wait: the task was interrupted by another one, so it is waiting, (iv) P(R): the task T is asking to lock a resource R, (v) Q(R): the task T is unlocking the resource R. The R-TNCES of a task T should include as many P(R) and Q(R) as the resources it may lock, but, in this illustrative example, we decided that T will use just one resource (R).The different signals indicated on the figure above stand for:(i) S8.1: when T switches from "Idle" to "Execute", the event S8.1 will force the running tasks with lower priorities to switch from "Execute" to "Wait". T's S8.1 is actually the S8.2 of tasks with lower priorities, (ii) S9.1: when T switches from "Execute" to "Idle", the events S9.1 will force the waiting tasks with lower priorities to switch from "Wait" to "Execute". T's S9.1 are actually the S9.2 of tasks with lower priorities, (iii) S1, S2 and S3: refer to the last paragraph.

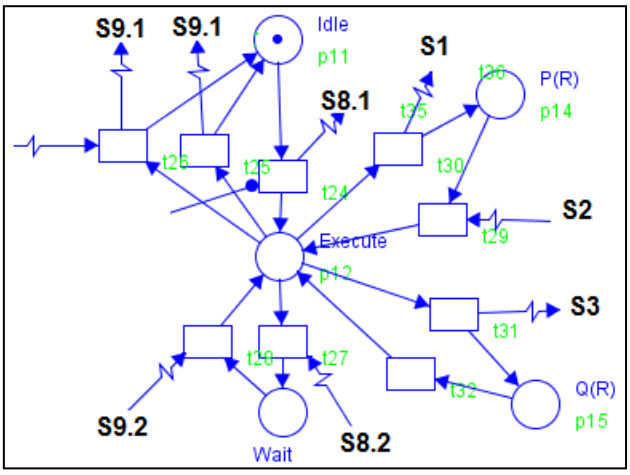

Fig. 13. Task's state R-TNCES.

Running example 2 in case study

We continue our case study by modeling, here, the case study using R-TNCES and tasks and shared resources' modeling we previously proposed. Let's remember that B has a higher priority than A (priority(A)=1 and priority(B)=2), and Q, R and S are all shared by the two tasks (Cl (Q) = Cl (R) = Cl (S) = priority (B) = 2). We start by modeling the two tasks as following in Fig. 14:

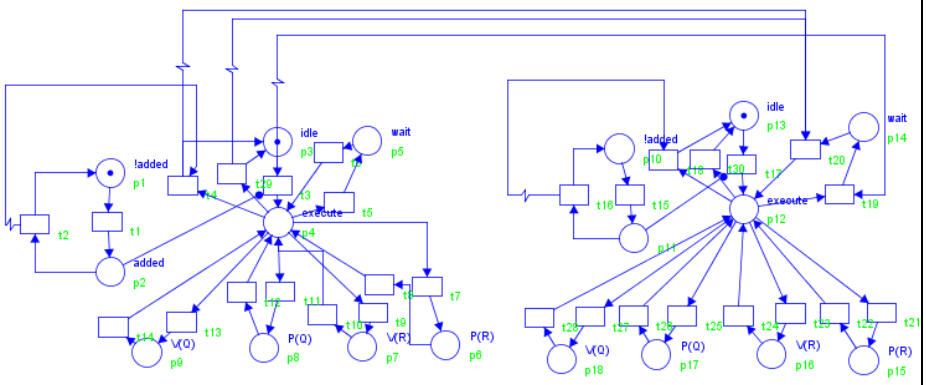

Fig. 14. $\mathcal{A}$ and $\mathcal{B}$'s modeling.

Now that PCP is used, A and B's behaviors in time become as following in Fig. 15. We note that the deadlock issue is now resolved.

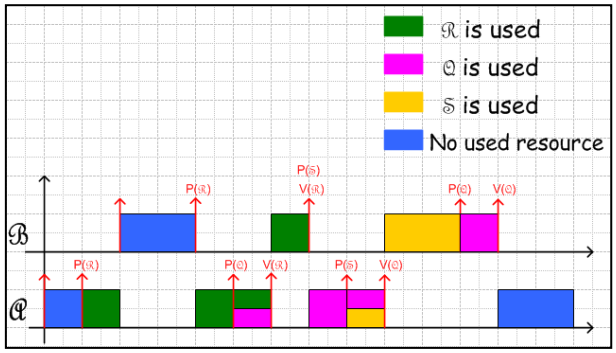

Fig. 15. $\mathcal{A}$ **and** $\mathcal{B}$ **behaviors after using PCP.**

5.3 Verification

Once the R-TNCES model of the DRCS is enriched with PCP, the next step is to verify whether the models meet users' requirements. So, any reconfiguration scenario dealing with adding/removal or resources does not lead to a blocking situation. Model-checking is a technique for automatically verifying the correctness properties of finite-state systems. Model checking for TNCES and R-TNCES is based on their reachability graphs. SESA [8] is an effective software environment for the analysis of TNCES, which computes the set of reachable states exactly. Typical properties which can be verified are boundedness of places, liveness of transitions, and reachability of states. In addition, temporal/functional properties based on Computation Tree Logic (CTL) specified by users can be checked manually. The following e-CTL formula is applied:

AG EX true

This formula is proven to be true by SESA as shown in the screenshot in Fig. 16, so there no deadlocks in our R-TNCES.

```
Parsing formula:
AG EX TRUE

The formula is TRUE.
```

Fig. 16. Screenshot from SESA.

We also check the safety property by checking if a given resource may be simultaneously locked by two different tasks. The following CTL formula is checked:

EF p22 AND p23

where: (i) p22 is the place translating that the resource $\mathcal{R}$ is locked by the task $\mathcal{A}$, (ii) p22 means that $\mathcal{B}$ locks $\mathcal{R}$. This formula is proven to be false as illustrated in Fig. 17.

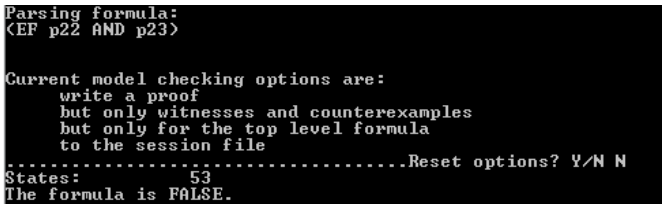

Fig. 17. Screenshot from SESA.

6 Conclusion

This research paper deals with the verification of distributed reconfigurable control system sharing resources. These systems are composed of several control tasks. A reconfiguration scenario is assumed to be in this paper as any dynamic operation allowing the addition, removal, update of tasks and resources. This is an original definition since no one in related works treats the case of changeable resources for reconfigurable systems. Although the modification of tasks is easy to do, the addition of resources may lead the system to unpredictable situations where blocking problems can occur. We show in this paper as a running example a case study which blocks the system after the addition of resources. Since R-TNCES is a new and original formalism for reconfigurable systems, we aim to enrich it with PCP in order to model DRCS. We extend the paper in [2] and propose models for resources and tasks to specify PCP. These models are applied in our case study which is proven to be safe thanks to our contribution. The contribution of this paper is well-useful for any real industrial case study since the resource sharing is often applied, but critical if run-time reconfiguration can be possible applied. We plan in future work to apply this contribution to the BROS project at Children Hospital of Tunis where the applicability of the paper's contribution is relevant.

Acknowledgements

This research work is carried out within a MOBIDOC PhD of the PASRI program, EU-funded and administered by ANPR.

References

1. Khalgui, Mohamed, et al. "Reconfiguration of distributed embedded-control systems." Mechatronics, IEEE/ASME Transactions on 16.4 (2011): 684-694.
2. Zhang, Jiafeng, et al. "R-TNCES: A Novel Formalism for Reconfigurable Discrete Event Control Systems." Systems, Man, and Cybernetics: Systems, IEEE Transactions on 43.4 (2013): 757-772.
3. Hanisch, H-M., et al. "Modeling of PLC behavior by means of timed net condition/event systems." Emerging Technologies and Factory Automation Proceedings, 1997. ETFA'97., 1997 6th International Conference on. IEEE, 1997.
4. Koh, I., and F. DiCesare. "Checking liveness in Petri nets using synchronic concepts." Proceedings of the Korean Automatic Control Conference. 1991.

5. Zhou, MengChu, and Frank DiCesare. "Parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources." Robotics and Automation, IEEE Transactions on 7.4 (1991): 515-527.
6. Sha, Lui, Ragunathan Rajkumar, and John P. Lehoczky. "Priority inheritance protocols: An approach to real-time synchronization." IEEE Transactions on computers 39.9 (1990): 1175-1185.
7. Goodenough, John B., and Lui Sha. The priority ceiling protocol: A method for minimizing the blocking of high priority Ada tasks. Vol. 8. No. 7. ACM, 1988.
8. Starke, Peter H., and Stephan Roch. Analysing signal-net systems. Professoren des Inst. für Informatik, 2002.
9. Emerson, E. Allen, and Joseph Y. Halpern. "Decision procedures and expressiveness in the temporal logic of branching time." Journal of computer and system sciences 30.1 (1985): 1-24.
10. Viswanadham, N., and Y. Narahari. Performance modeling of automated manufacturing systems. Englewood Cliffs, NJ: Prentice Hall, 1992.
11. Koh, Inseon, and Frank DiCesare. "Transformation methods for generalized Petri nets and their applications to flexible manufacturing systems." Computer Integrated Manufacturing, 1990., Proceedings of Rensselaer's Second International Conference on. IEEE, 1990.
12. Krogh, B. H., and C. L. Beck. "Synthesis of place/transition nets for simulation and control of manufacturing systems." (1986).
13. Zhang, Jia Feng, Olfa Mosbahi, Mohamed Khalgui and Atef Gharbi. "Feasible Dynamic Reconfigurations of Petri Nets." Formal Methods in Manufacturing Systems: Recent Advances. IGI Global, 2013. 247-267. Web. 3 May. 2014. doi:10.4018/978-1-4666-4034-4.ch010.