On Improved Verification of Reconfigurable Real-Time Systems

Yousra Hafidi[1,2,3,4], Laid Kahloul[3], Mohamed Khalgui[1,2] and Mohamed Ramdani[1,2,3,4]
[1]*LISI Laboratory, National Institute of Applied Sciences and Technology, University of Carthage, Tunis 1080, Tunisia*
[2]*School of Electrical and Information Engineering, Jinan University, China*
[3]*LINFI Laboratory, Computer Science Department, Biskra University, Algeria*
[4]*University of Tunis El Manar, Tunis, Tunisia*

Keywords: Real-time System, Reconfiguration, Formal verification, Model-checking, CTL.

Abstract: This paper deals with formal modeling and verification of reconfigurable real-time systems under reconfigurability and real-time constraints. To deal with the modeling, we enrich the formalism, named reconfigurable timed net condition event systems (R-TNCESs), with new reconfiguration forms such as the ability to update the earliest/latest firing time on the intervals which are associated to flow arcs. To handle the verification of the new extended formalism, an algorithm is defined to generate a timed accessibility graph for timed net condition event systems (TNCESs). We control the verification complexity of R-TNCESs using a new method which accelerate the generation of accessibility graphs, where redundancies, repetitions, and unnecessary computations are avoided as much as possible. An experimentation is carried out and a performance evaluation is achieved to demonstrate the advantages of the proposed contribution compared with related works.

1 INTRODUCTION

Several critical systems such as medical, aerospace and manufacturing systems are facing challenges like dealing with the technology process updating, fault-tolerance, response in time, flexibility, modularity, etc. Systems had to comprise new abilities in order to efficiently answer today's requirements. Actually, any problem that a critical system may face during its execution can cause serious consequences. Most problems of critical systems are due to a faulty and an unreliable design. To deal with those problems, many academic researchers as well as industrial companies tackle systems reliability by formal verification (Ramdani et al., 2018).

Formal verification methods exploit techniques based on mathematical and logical proofs to check whether a system meets the requirements of its initial specification. Indeed, system requirements are usually specified in a temporal logic like computational tree logic (CTL), and/or its extensions: extended CTL (eCTL), timed CTL (TCTL), etc. The system design is carried out using one of the existing formal languages such as *Petri Nets* and their extensions. Many system properties including safety, deadlock-freedome and liveness that are specified by a temporal logic can be verified using model-checking (Baier et al., 2008). Model-checking is the process that takes as inputs a model (typically a state/transition system) and a property (typically written in a temporal logic), then proves that the system satisfies the given property or provides a counterexample of the execution that falsifies it.

Reconfigurability is the ability of systems to transform their selves and their working process in order to adapt to a changed inner/outer environment, respond to user requirements, prevent malfunctions when hardware failures occur during the process, etc. Reconfigurable real-time systems are systems that encompass reconfigurability constraints (Lyke et al., 2015) and they subject to real-time requirements (Lakhdhar et al., 2018).

By the inclusion of some new skills, reconfigurable real-time systems become more complex, i.e., their design includes more details, and their verification becomes more expensive in terms of computation time and memory. Researchers have tried to deal with the formal modeling and verification of discrete event systems using *Petri nets* and their extensions. (Badouel and Oliver, 1998) proposed reconfigurable Petri nets which are considered as high level Petri nets with special abilities of self reconfiguration. (Biermann and Modica, 2008) proposed reconfigurable object Petri nets (RONs) that are used to design recon-

Hafidi, Y., Kahloul, L., Khalgui, M. and Ramdani, M.
On Improved Verification of Reconfigurable Real-Time Systems.
DOI: 10.5220/0007736603940401
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 394-401
ISBN: 978-989-758-375-9

figurable manufacturing systems. RONs formalism has two types of places (1) net places that contain ordinary Petri nets as tokens, and (2) rule places that contain rules as tokens. Also, two types of transitions (1) firing transitions that model the simple firing of Petri nets, and (2) transform transitions that model the reconfiguration of the system. (Rausch and Hanisch, 1995) proposed net condition/event systems (NCESs) formalism which is a modular Petri nets extension enriched with event/condition signals that models interactions among system modules. NCESs are developed through the last years to timed net condition/event systems (TNCESs) (Hanisch et al., 1997) involving time constraints on arcs. (Zhang et al., 2017) proposed reconfigurable timed net condition/event systems (R-TNCESs) which is an enriched extension of Petri nets formalism that supports reconfiguration constraints. In R-TNCESs formalism, the system is represented by a couple $Sys(B_{sys}, R_{sys})$ such that (1) B_{sys} is a set of TNCESs that represent the behavior module, and (2) R_{sys} is a set of reconfiguration rules that represent the control module. All of those research works are important because they are building convenient formal models. However, these models face important problems when they are used to verify complex reconfigurable real-time systems. The formal verification of reconfigurable real-time systems is a hard computationally problem that requires so much time and memory, and it is identified as a very expensive task. Consequently, proposing a new methodology for ensuring the safety of these systems as well as controlling the complexity of their verification is an important research area. In this paper, we model reconfigurable real-time systems using R-TNCESs formalism. In fact, R-TNCESs formalism is like the well-known formalism timed net condition/event systems (TNCESs) (Hanisch et al., 1997) such that R-TNCESs formalism does not change the semantic of TNCESs but it just gives functional structure and a pattern for reconfigurable systems in terms of (B_{sys}, R_{sys}). R-TNCES is a suitable model because it provides modularity, time and reconfiguration abilities. However, many computations and redundancies can be encountered during R-TNCESs verification process. To deal with the complexity problem, we propose a method that benefits from the similarities between the system's configurations to avoid unnecessary and repetitive calculations. Indeed the paper proposes a method that generates an accessibility graph from another one according to the system's reconfiguration. Given an R-TNCES $Sys(B_{sys}, R_{sys})$, where (1) $B_{sys} = \{C_1, C_2\}$ is the set of system configurations, (2) $R_{sys} = \{rule_{C_1C_2}\}$ is the set of possible reconfiguration rules such that $rule_{C_1C_2}$ transforms the configuration C_1 to C_2, and (3) $tAG(C_0)$ is the timed accessibility graph of the configuration C_0. The proposed method, in this paper, shows how to generate $tAG(C_2)$ from $tAG(C_1)$ according to $rule_{C_1C_2}$, (i.e., rather than computing the whole accessibility graph $tAG(C_2)$ from zero, the new method applies the corresponding graph modifications such as adding/removing a state/arc in $tAG(C_1)$ in order to obtain $tAG(C_2)$).

(Hafidi et al., 2018) propose a methodology that improves the modeling and the verification of reconfigurable discrete event control systems using R-TNCESs formalism. The authors main contribution is efficient for the verification of functional properties in R-TNCESs. However, the suggested methodology cannot be used for systems under reconfigurability and real-time constraints. The main difference between the paper's methodology and the one presented in (Hafidi et al., 2018) is that it shows how to generate an accessibility graph from another one when a reconfiguration on real-time constraints occurs which is not considered in other works. The paper's contribution will complete the work presented in (Hafidi et al., 2018). Therefore, we assume that functional properties are already verified in the system, we focus on real-time properties, reconfiguration properties, their modeling in R-TNCES formalism and their efficient verification.

The main contributions of this paper are summarized as:

- The enrichment of R-TNCESs with new real-time reconfiguration forms such that modifying the earliest/latest firing times on the timed arcs are included, i.e., new structure modification instruction for the new reconfiguration forms;
- The proposition of new rewriting rules that generate a new graph from a given one, according to the reconfiguration on time applied by the system, i.e., this is used to control the complexity of the verification task.

The originality of this research work can be founded from two general parts, i.e., the formal modeling and the improved verification of reconfigurable real-time systems using R-TNCESs. To the best of our knowledge, this is the first study that deals with the enrichment of R-TNCESs modeling by the new reconfiguration form of real-time systems, i.e., the modification of time constraints on timed arcs. In addition, no previous research works have tackled with the complexity control and optimization of the verification task. The performance evaluation proves that the complexity of the verification task increases exponentially if it is not controlled such as in the blind method which constructs the whole accessibility graph of the system

after each reconfiguration step. However, by using the proposed method in this paper, significant gains in computation time are achieved for the same verification result as in the classical algorithm. The experimentation and the performance evaluation results are compared using the model checker SESA (Patil et al., 2015) which analyses TNCESs models and computes their accessibility graphs.

The remainder of the paper is organized as follows. Section 2 outlines the definition of R-TNCES formalism and explains its enrichment with the new time reconfiguration forms. Section 3 defines the proposed method for improving the verification of real-time and reconfiguration properties in R-TNCESs. Section 4 shows the performance of the proposed method on a case study. Section 5 concludes the paper with the limitations of the current work and the perspectives for future works.

2 PRELIMINARIES: RECONFIGURABLE TIMED NET CONDITION/EVENT SYSTEMS (R-TNCESs)

Reconfigurable timed net condition/event systems (R-TNCESs) are an extension of Petri nets (Zhang et al., 2017), used for formal specification of reconfigurable discrete event control systems (RDECSs). An RDECS may encompass a set of configurations, where each one is modeled by a TNCES. A TNCES is a set of modules graphically represented as depicted in Fig. 2. To model an RDECS, we use the concept of control components (CCs) introduced in (Khalgui et al., 2011), i.e., the interconnected modules communicating with signals that compose each TNCES are called control components (CCs). The syntax and semantics of the previous structures are explained in this subsection.

2.1 Syntax

R-TNCESs are formally defined in as a couple $RTN = (B, R)$ where B (respectively, R) is the behavior (respectively, the control) module of a reconfigurable discrete event control system (RDECS). B is a union of multi-TNCESs represented by

$$B = (P, T, F, W, CN, EN, DC, V, Z_0)$$

where, (1) P (respectively, T) is a finite set of places (respectively,transitions), (2) $F \subseteq (P \times T) \cup (T \times P)$ is a superset of flow arcs, (3) $W : (P \times T) \cup (T \times P) \longrightarrow \{0, 1\}$ maps a weight to a flow arc, (4) $W(x, y) > 0$ if $(x, y) \in F$, and $W(x, y) = 0$ otherwise, where $x, y \in P \cup T$, (5) $CN \subseteq (P \times T)$ (respectively, $EN \subseteq (T \times T)$) is a superset of condition signals (respectively, event signals), (6) $DC : F \cap (P \times T) \rightarrow \{[l_1, h_1], \ldots, [l_{|F\cap(P\times T)|}, h_{|F\cap(P\times T)|}]\}$ is a superset of time constraints on transition's input flow arcs, where $\forall i \in [1, |F \cap (P \times T)|]$, $l_i, h_i \in \mathbb{N}$ and $l_i < h_i$, (7) $V : T \longrightarrow \{\vee, \wedge\}$ maps an event-processing mode (AND or OR) for every transition, and (8) $Z_0 = (M_0, D_0)$, where $M_0 : P \longrightarrow \{0, 1\}$ is the initial marking, and $D_0 : P \longrightarrow \{0\}$ is the initial clock position.

R is a set of reconfiguration rules such that rule r is a structure represented by

$$r = (Cond, s, x)$$

where, (1) $Cond \rightarrow \{True, False\}$ is the pre-condition of r, i.e., r is executable only if $Cond = True$, (2) $s : TN({}^\bullet r) \rightarrow TN(r^\bullet)$ is the structure-modification instruction such that $TN({}^\bullet r)$ (respectively, $TN(r^\bullet)$) represents the structure before (respectively, after) applying the reconfiguration r, and (3) $x : last_{state}({}^\bullet r) \rightarrow initial_{state}(r^\bullet)$ is the state processing function.

In this paper, we denote by r_{ij} the reconfiguration rule that transforms $TNCES_i$ to $TNCES_j$.

As reported in (Zhang et al., 2017), the basic possible structure-modification instructions for R-TNCESs are summarized by adding/removing signals (i.e., condition signals or event signals) between or among modules. However, other possible reconfiguration forms should be considered in this paper to express the transformation of time constraints. Therefore, we present in Table. 1 A new time structure-modification instructions for R-TNCESs. We denote by p a place, t a transition, eft the earliest firing time, lft the latest firing time, $\mathbb{N}^+ = \{1, 2, ...\}$ the set of positive natural numbers, and $\mathbb{N} = \mathbb{N}^+ \cup \{0\}$ the set of all natural numbers.

Table 1: Time structure-modification instructions.

Instruction	Symbol
Modify the earliest or/and the latest firing time value in the time interval of the flow arc (p, t). $eft \in \mathbb{N}^+ \wedge lft$, $\omega \in \mathbb{N} \wedge eft < lft < \omega$	$mtime((p, t), [eft, lft])$

mtime: symbol of the instruction that modifies time constraints.

2.2 Semantics

An R-TNCES $RTN(B_{RTN}, R_{RTN})$ behavior is described by the dynamism of tokens inside of each $TNCES \in B_{RTN}$, and the transformations applied by each reconfiguration rule $rule \in R_{RTN}$. There exist two types of transitions in TNCESs formalism: (1) forced transitions have at least an incoming signal arc (2) spontaneous transitions have no incoming signal arcs. Enabled forcing transitions with input flow arcs associated by time interval $[eft, lft]$ should fire after a duration d since it became enabled such that $eft \leq d \leq lft$.

A reconfiguration rule $rule(cond, s, x)$ has the priority to be applied first when its condition is verified, i.e., $cond = True$. In this case, the enability of transitions falls down and only the reconfiguration rule is applied. A reconfiguration rule r_{st} transforms a TNCES source TNS_s to a TNCES target TNS_t. $last_{state}(^{\bullet}r_{st})$ denotes the last state where the simulation among TNS_s ends (i.e., the dynamism of tokens), it also denotes the source state where the reconfiguration rule is applied. $initial_{state}(r_{st}^{\bullet})$ denotes the initial state where the simulation among TNS_t starts, it also denotes the target state after applying the reconfiguration rule.

3 VERIFICATION OF TIME CONSTRAINTS IN RECONFIGURABLE SYSTEMS USING TAG

This section deals with the checking whether the modeled system (R-TNCES) meets the temporal requirements. In this task, we specify system properties using TCTL, we compute the accessibility graphs, and we use model-checking to check whether temporal properties are satisfied or not. Classical accessibility graphs (AGs) are extended to timed accessibility graphs (TAGs) and a new method is proposed to optimize the calculation of these last ones.

3.1 Timed Accessibility Graph

Timed accessibility graph (TAG) of a TNCES TNS is a structure tAG given by

$$tAG(St, Ed, s_0)$$

where, (1) St denotes the set of reachable states, (2) $Ed : St \rightarrow St$ denotes the set of edges that defines state-transitions such that each edge is labeled by the executed step, and (3) s_0 denotes the initial state.

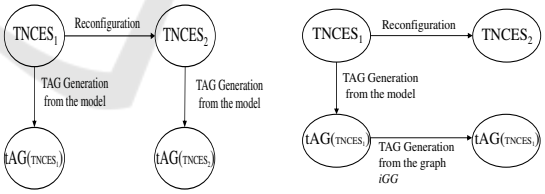
A state $s \in St$ is a structure given by

$$State(Mp, Pclocks, D)$$

where, (1) Mp is the set of marked places in TNS, (2) $Pclocks$ is a vector of integers representing places clock positions, and (3) D is the delay of the state which denotes the minimal number of time units after which at least one step becomes enabled.

3.2 TAG Generation from a Graph (Contribution)

Given two TNCESs $TNCES_1$ and $TNCES_2$ such that $TNCES_2$ is obtained from $TNCES_1$ by applying a time modification instruction. Classically, $tAG(TNCES_1)$ (respectively, $tAG(TNCES_2)$) the timed accessibility graph of $TNCES_1$ (respectively, $TNCES_2$) is computed using the classical algorithm explained in (Murata, 1989), where the whole accessibility graph of each structure is computed from zero (Fig. 1(a)). Actually, $tAG(TNCES_1)$ and $tAG(TNCES_2)$ share some similar parts (sub-graphs) that should not be recomputed again while generating $tAG(TNCES_2)$. Consequently, the complexity of the accessibility graphs generation can be optimized if these repetitive computations are avoided. In this paper as depicted in Fig. 1(b), we propose an improved graph-generation method *iGG* that computes $tAG(TNCES_2)$ from the graph $tAG(TNCES_1)$ rather than computing $tAG(TNCES_2)$ from the model. The proposed *iGG* method then, considers the already computed parts and does not recalculate them.



(a) TAG generation from model..

(b) TAG generation from graph.

Figure 1: TAG generation from graph.

3.3 The Improved Graph-generation Method *iGG* (Contribution)

In order to verify system properties in an R-TNCES model $RTN(B, R)$, the timed accessibility graph of each TNCES $TS \in B$ should be generated using the classical method described in (Murata, 1989), i.e., the algorithm is therefore executed $\mid B \mid$ times. Consequently, the operation requires more time and memory.

Table 2: Rewriting rules on TAG.

SMI	Rewriting rules on TAGs	Comments
$mtime((p,t),[eft,lft])$	a) $\forall e \in Ed, t \subset Label(e) ::= St \leftarrow St \setminus \{sc(e)\}$; b) $\forall s \in St ::= SimulationFrom(s)$.	a) Remove all source states of edges labeled by t in TAG ; b) Continue the simulation from each state.

Given two TNCESs structures $TNCES_a$, $TNCES_b$, where tAG_a is the timed accessibility graph of $TNCES_a$. The TNCES $TNCES_b$ is obtained from $TNCES_a$ after applying a transformation of time constraints which is described by the structure modification instruction *SMI* in Tables 1 and 2. The new proposed method *iGG* (improved graph generation) takes as an input tAG_a then transforms it into a new graph tAG_b by adding/removing some states/edges from tAG_a. Therefore, the complexity of *iGG* is $O(1)$ in its best case where $O(1)$ is the complexity of each instruction of modification on the graph. The complexity in the worst case is $O(e^m)$, such that $O(e^m)$ is the complexity of accessibility graphs computations as reported in (Murata, 1989). The resulting graph is exactly the timed accessibility graph of $TNCES_b$ except that by using *iGG* method, there are less computed states, i.e., no repetitive calculations for the similar parts.

Table 2 introduces the proposed rewriting rules on timed accessibility graphs (TAGs) related to the new time structure modification instructions (SMI) proposed in this paper. We denote by e a TAG edge, t a transition, Ed the set of edges in a TAG, s a state in a TAG, St the set of states in a TAG, $sc(e)$ the function that returns the source state of an edge e in a TAG, and $Label(e)$ the function that returns the label of the edge e in a TAG. *iGG* method is applied in the case of having n SMIs to get tAG_b as follows.

$Step_0$ Copy tAG_a to tAG_b, i.e., initially, tAG_b is a copy of tAG_a;

$Step_1$ For every structure modification instruction SMI apply the indicated rewriting rules (Table 2) on tAG_b;

$Step_2$ Delete all unreachable states in tAG_b.

3.4 $iGG_{generalized}$: *iGG* for R-TNCESs (Contribution)

Algorithm 1 deals with the application of *iGG* in the case when having n TNCESs. The proposed algorithm is recursive and composed of a parallel part that computes the TAGs of reachable TNCESs in the same time when possible. The algorithm stops in two cases: (1) if the behavior of a configuration is not validated by the verification, or (2) if it reaches a configuration that has been already verified before, i.e., to avoid redundant computations. In Algorithm 1, we denote by: (1) $NextConfigs(tAG, R)$ the function that from the TAG tAG of the current TNCES and a set of possible reconfiguration rules R returns the set of reachable TNCESs resulted from reconfigurations (2) $newTN(TNCES_i)$ the Boolean function that returns $True$ if $TNCES_i$ has not been verified before, otherwise it returns $False$, (3) $iGG(tAG, TNCES_i)$ the function that generates and returns the TAG of $TNCES_i$ from the TAG tAG (already explained in previous subsections), and (4) $verifyPropertiesIn(tAG_i, p)$ the Boolean function that returns $True$ if the system indicated properties p are verified on tAG_i, otherwise it returns $False$.

Algorithm 1: $iGG_{Generalized}$.

```
Input: RTN(B : Set of TNCESs, R : Set of
       Reconfiguration Rules): R-TNCES;
       tAG_current: TAG; p: Set of Properties;
Variables : tAG: TAG; isCorrect: Boolean;
            ToVerify: Set of TNCESs;
1  tAG ← tAG_current;
2  ToVerify ← NextConfigs(tAG, R);
3  foreach TNCES_i ∈ ToVerify in parallel do
4      if newTN(TNCES_i) /* New TNCES: if
          its TAG has not been computed
          yet                          */
5      then
6          tAG_i ← iGG(tAG, TNCES_i);
7          isCorrect ←
            verifyPropertiesIn(tAG_i, p);
8          if isCorrect == True then
9              iGG_Generalized(RTN(B, R), tAG_i,
               p);
10         end
11     end
12 end
```

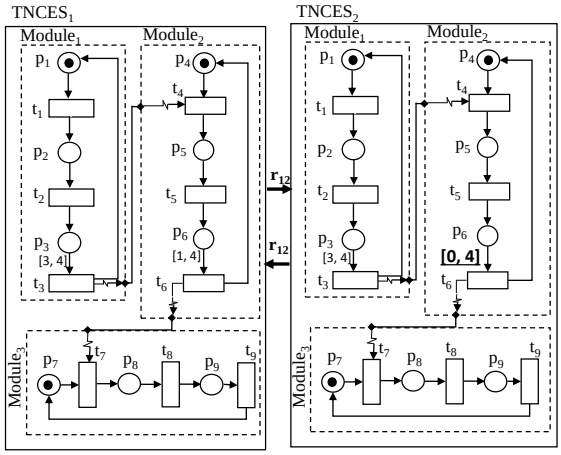
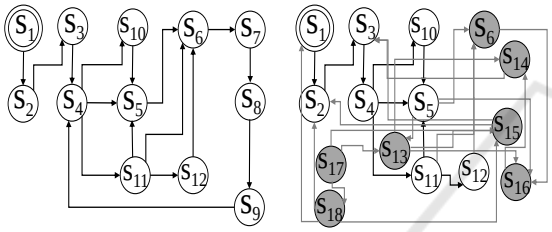


Figure 2: RTN Graphical model.



(a) $tAG(TNCES_1)$. (b) $tAG(TNCES_2)$.

Figure 3: *RTN* timed accessibility graphs.

4 EXPERIMENTATION

In this section, we experiment the proposed approach on a running example, and we evaluate its performance on large scale systems.

4.1 Running Example

Given an R-TNCES $RTN(B_{RTN}, R_{RTN})$ such that $B_{RTN} = \{TNCES_1, TNCES_2\}$ is the behavior module containing all possible configurations and $R_{RTN} = \{r_{12}, r_{21}\}$ is the control module containing all possible reconfigurations. The graphical model of *RTN* is depicted in Fig. 2.

The reconfiguration rules in R_{RTN} are described in Table 3.

Reconfigurations run spontaneously when their conditions are fulfilled, i.e., there are no time constraints on reconfiguration scenarios. Therefore, the control module R_{RTN} is not considered by the verification of real-time properties. Contrarily, the behavior module B_{RTN} contains a set of TNCESs which are timed and should be validated by checking real-time properties. Note that in the behavior module B_{RTN}, there exist similar parts between both configurations $TNCES_1$ and $TNCES_2$, e.g., $module_1$ in $TNCES_1$ is similar to $module_1$ in $TNCES_2$ and $module_2$ in $TNCES_1$ is similar to $module_2$ in $TNCES_2$. The repetitive calculations on those similar parts are considered as redundancies that make of the verification a complex task.

In this subsection, we try to apply the proposed method to verify the R-TNCES $RTN(B_{RTN}, R_{RTN})$ efficiently by avoiding as much as possible necessary computations. The timed accessibility graph of the initial structure $TNCES_1$ is computed classically using SESA tool. The resulted graph $tAG(TNCES_1)$ is depicted in Fig.3(a). To compute $tAG(TNCES_2)$ from $tAG(TNCES_1)$ we use the improved graph-generation method *iGG* as following:

$Step_0$	Copy $tAG(TNCES_1)$ to $tAG(TNCES_2)$;
$Step_1$	Apply the rewriting rules (Table 2) on tAG_b as in $Step_{11}$ and $Step_{12}$;
$Step_{1.1}$	$(\forall e \in Ed_2, t_6 \subset Label(e) ::= St_2 \leftarrow St_2 \setminus \{s_6\}) \Rightarrow St_2 \leftarrow St_2 \setminus \{s_6\}$. Ed_2 (respectively, St_2) represents the set of edges (respectively, states) in $tAG(TNCES_2)$;
$Step_{1.2}$	$\forall s \in St ::= SimulationFrom(s)$. By the simulation, new states are created: $St_2 \leftarrow St_2 \cup \{S_6, S_{13}\, S_{14}\, S_{15}\, S_{16}\, S_{17}\, S_{18}\}$;
$Step_2$	Delete all unreachable states in tAG_b: $St_2 \leftarrow St_2 \setminus \{S_7, S_8\, S_9\, S_{15}\}$.

After following the previous steps, $tAG(TNCES_2)$ the new accessibility graph of $TNCES_2$ is achieved. $tAG(TNCES_2)$ is depicted in Fig. 3(b) where the colored states among it denote the new computed ones.

Note that the studied system *RTN* has 12 states in the configuration $TNCES_1$, and 15 states in the configuration $TNCES_2$ where only 7 states are computed using the improved graph-generation method *iGG*, i.e., the other states are kept from the first TAG $tAG(TNCES_1)$. Therefore, *iGG* has avoided the unnecessary repetitive computations and optimized *RTN* accessibility graphs generation by more than 50% calculations.

4.2 Concept of Redundancies

We define the function $RRedun(TNCES_a, TNCES_b)$ that takes two TNCESs $TNCES_a$, $TNCES_b$ and gives the redundancy rate between them.
$RRedun(TNCES_a, TNCES_b)$ is computed as follows

$$RRedun(TNCES_a, TNCES_b) = \frac{\#similarStates}{\#States}$$

where (1) $\#similarStates$ is the number of similar states that appear in both graphs $tAG(TNCES_a)$ and

Table 3: *RTN* reconfiguration rules.

Rule	Condition	s	x
r_{12}	$True$	$\{mtime((p_6, t_6), [0, 4])\}$	$\{(S_9, TNCES_1), (S_1, TNCES_2)\}$
r_{21}	$True$	$\{mtime((p_6, t_6), [1, 4])\}$	$\{(S_{16}, TNCES_2), (S_1, TNCES_1)\}$

Table 4: States marking and clocks positions.

State		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	D
S_1	Mp	1	0	0	1	0	0	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_2	Mp	0	1	0	1	0	0	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_3	Mp	0	0	1	1	0	0	1	0	0	2
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_4	Mp	1	0	0	0	1	0	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_5	Mp	0	1	0	0	0	1	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_6	Mp	0	0	1	0	0	1	1	0	0	1
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_7	Mp	0	0	1	1	0	0	0	1	0	0
	$Pclocks$	0	0	1	0	0	0	0	0	0	
S_8	Mp	0	0	1	1	0	0	0	0	1	0
	$Pclocks$	0	0	1	0	0	0	0	0	0	
S_9	Mp	0	0	1	1	0	0	1	0	0	2
	$Pclocks$	0	0	1	0	0	0	0	0	0	
S_{10}	Mp	1	0	0	0	0	1	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{11}	Mp	0	1	0	0	1	0	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{12}	Mp	0	0	1	0	1	0	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{13}	Mp	0	1	0	1	0	0	0	1	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{14}	Mp	0	0	1	1	0	0	0	0	1	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{15}	Mp	0	1	0	1	0	0	0	0	1	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{16}	Mp	0	0	1	1	0	0	0	1	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{17}	Mp	0	0	1	0	0	1	1	0	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	
S_{18}	Mp	1	0	0	1	0	0	0	1	0	0
	$Pclocks$	0	0	0	0	0	0	0	0	0	

$tAG(TNCES_b)$, and (2) $\#States$ is the total number of states in $tAG(TNCES_b)$.

We denote by: low redundancy rate *LRR* the systems with $RRedun \leq 30\%$, medium redundancy rate *MRR* the systems with $30\% < RRedun < 80\%$, and high redundancy rate *HRR* the systems with $RRedun \geq 80\%$, e.g. in the example of the experimentation part $RRedun_{TNCES_1 TNCES_2} = \frac{8}{15} = 53\%$ is the redundancy rate of *RTN* which denotes that *RTN* is in *MRR* systems.

4.3 Evaluation of Performance

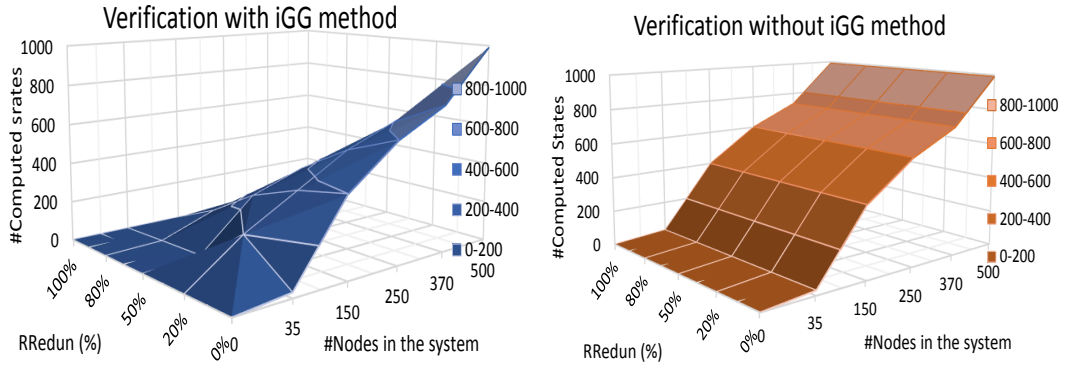
The proposed method has proven benefits in the experimentation of the previous running example. We want to study its efficiency in larger systems by measuring computed states where a parameter namely redundancy rate is used as a factor to held the performance evaluation in different kinds of problems.

The 3D surfaces in Fig. 4 depict the number of computed states resulted from the analyzes held on R-TNCESs with: (1) different redundancy rates, and (2) different numbers of nodes. The study is performed in two cases, the former by using the proposed *iGG* method while computing the TAGs, the latter using the classical method and without any improvement. The surfaces show that the number of computed states using *iGG* reduces when the *RRedun* is higher, i.e., in *HRR* systems. Therefore, *iGG* method performs best for *HRR* systems verification regardless the number of nodes. The surface presented in Fig. 4(a) matches to the surface depicted in Fig. 4(b) when $RRedun = 0$. Thus, *iGG* method turns to classical verification methods when the redundancy rate is very low, i.e., the method is in its worst case.

5 CONCLUSION

This research work deals with the formal modeling and verification of reconfigurable real-time systems such as reconfigurable discrete event control systems RDECSs. The modeling is enriched with new possible reconfiguration forms: the modification of the earliest/latest firing time in the intervals associated to flow arcs. The proposed method *iGG* showed how to generate a TAG from another one to avoid repetitive computations when the two TAGs have some similar parts.

According to the formal running example and the performance evaluation results, it is shown that the verification task of temporal properties has been improved in terms of computing time and memory. In addition, it is proved that the proposed method performs best for *HRR* systems. The proposed method is less beneficial in *LRR* systems. Actu-



(a) *IGG* method. (b) Classical method.

Figure 4: *iGG* efficiency.

ally in RDECSs reconfigurations, the transformation includes only some modules and others will still be identical as those in the source model, which gives a high similarity between models and makes most of them *HRR* systems. Therefore, the proposed methodology is suitable for RDECSs improved verification. Compared with the previous related works, this work presents a new reconfiguration form to the R-TNCES formalism, a method to verify real-time properties where the correctness of the system is considered and also the complexity of its verification is controlled.

Future works will (1) provide a formal proof of correctness proving that information on the system's behavior are not lost or corrupted after applying the proposed improvement method, (2) consider probabilistic constraints in the verification task, and (3) involve new techniques to reduce the system properties and TAGs in order to improve the model-checking on R-TNCESs, and (4) include the proposed improvement method in a model-checker in order to automatize it and profit from its gain. Finally the proposed techniques will be generalized to be considered in other formalisms like reconfigurable Petri nets (Padberg and Kahloul, 2018).

REFERENCES

Badouel, E. and Oliver, J. (1998). *Reconfigurable nets, a class of high level Petri nets supporting dynamic changes within workflow systems*. PhD thesis, INRIA.

Baier, C., Katoen, J., and Larsen, K. (2008). *Principles of Model Checking*. MIT Press.

Biermann, E. and Modica, T. (2008). Independence analysis of firing and rule-based net transformations in reconfigurable object nets. *Electronic Communications of the EASST*, 10.

Hafidi, Y., Kahloul, L., Khalgui, M., Li, Z., Alnowibet, K., and Qu, T. (2018). On methodology for the verification of reconfigurable timed net condition/event systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, (99):1–15.

Hanisch, H.-M., Thieme, J., Luder, A., and Wienhold, O. (1997). Modeling of PLC behavior by means of timed net condition/event systems. In *Proc. 6th International Conference on Emerging Technologies and Factory Automation Proceedings*, pages 391–396. IEEE.

Khalgui, M., Mosbahi, O., Li, Z., and Hanisch, H. M. (2011). Reconfigurable multiagent embedded control systems: From modeling to implementation. *IEEE Transactions on Computers*, 60(4):538–551.

Lakhdhar, W., Mzid, R., Khalgui, M., Li, Z., Frey, G., and Al-Ahmari, A. (2018). Multiobjective optimization approach for a portable development of reconfigurable real-time systems: From specification to implementation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Lyke, J. C., Christodoulou, C. G., Vera, G. A., and Edwards, A. H. (2015). An introduction to reconfigurable systems. *Proceedings of the IEEE*, 103(3):291–317.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Padberg, J. and Kahloul, L. (2018). Overview of reconfigurable petri nets. In *Graph Transformation, Specifications, and Nets*, pages 201–222. Springer.

Patil, S., Vyatkin, V., and Pang, C. (2015). Counterexample-guided simulation framework for formal verification of flexible automation systems. In *Proc. IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1192–1197.

Ramdani, M., Kahloul, L., and Khalgui, M. (2018). Automatic properties classification approach for guiding the verification of complex reconfigurable systems. In *Proc. Proceedings of the 13th International Conference on Software Technologies - Volume 1: ICSOFT,*, pages 591–598. INSTICC, SciTePress.

Rausch, M. and Hanisch, H.-M. (1995). Net condition/event systems with multiple condition outputs. In *Proc. Emerging Technologies and Factory Automation*, volume 1, pages 592–600. IEEE.

Zhang, J., Frey, G., Al-Ahmari, A., Qu, T., Wu, N., and Li, Z. (2017). Analysis and control of dynamic reconfiguration processes of manufacturing systems. *IEEE Access*.