On Improvement of Formal Verification of Reconfigurable Real-Time Systems Using TCTL and CTL-Based Properties on IaaS Cloud Environment

Chams Eddine Choucha[1(✉)], Mohamed Ramdani[1], Moahmed Khalgui[1,2], and Laid Kahloul[3]

[1] LISI Laboratory, National Institute of Applied Sciences and Technology (INSAT), University of Carthage, 1080 Tunis, Tunisia

[2] School of Electrical and Information Engineering, Jinan University (Zhuhai Campus), Zhuhai 519070, China

[3] LINFI Laboratory, Computer Science Department, Biskra University, Biskra, Algeria

Abstract. The verification of reconfigurable real-time systems that dynamically change their structures due to external changes in environment or user requirements continues to challenge experts which have to face new challenges such as fault tolerance, response in time, flexibility, modularity, etc. Moreover, such systems face constraints as real-time requirements, their generated state spaces are much bigger, consequently, properties to be verified are more complex, which makes the formal verification more complex. For modeling systems, in this paper, we use Reconfigurable Timed Net Condition/Event Systems (R-TNCESs) for the optimal functional and temporal specification. To control the complexity and to reduce the verification time, a new method of properties verification in a cloud-based architecture is proposed. The novelty consists of a new method for state space generation and the decomposition of the complex properties for running an efficient verification. Moreover, An algorithm is proposed for the incremental state space generation. An application of the paper's contribution is carried out on a case study to illustrate the impact of using this technique. The current results show the benefits of the paper's contribution.

Keywords: Discrete-event system · Reconfiguration · R-TNCES · Computation Tree Logic · CTL · Cloud computing · Formal verification

1 Introduction

Reconfigurable discrete event control systems (RDECSs) such as manufacturing systems [11], real time systems and intelligent control systems [10,12] are complex. RDECSs satisfy several conditions such as concurrency, control and communication. In fact, RECESs are the trend of future systems. However, ensuring the safety of these systems is crucial especially when dealing with critical situations. Formal verification is, therefore, imperative. RDECSs have flexible configurations that allow them to switch from a configuration to another due to user requirements or to prevent system malfunctions [6]. This verification consists of two major steps: state-space generation and state-space analysis. Mentioned steps applications are usually expensive in terms of computation time and memory occupation (i.e., huge accessibility graph to be generated and complex properties to be verified) [19]. The authors in [17] proposed to classify properties automatically and to introduce a priority order during RDECSs verification to control the high number of properties to be verified. The mentioned method improves verification by reducing the number of properties to be verified by exploiting relationships among properties (equivalence, composition and dominance). However, when the property relationship rate is low which is frequent while verifying complex RDECSs, the said method is equivalent to the classic ones. The authors in [6] proposed a method for accessibility graph generation with less computing time and less required memory, while preserving the graph semantics. They start by computing the initial TNCES accessibility graph classically, then making updates on it to compute the remaining TNCESs accessibility graphs, while considering similarities between them. Previous methods improve classical ones. However, with large scale systems, their application using a unique machine (i.e., a centralized system) may be expensive in terms of time and calculation [13]. Authors in [2] initiate the cloud-based solution for formal method problems. Authors have proposed a distributed fixed-point algorithm to check CTL properties with basic operators. The said algorithm can analyze DECS efficiently. However, RDECSs complexity forced us to move forward with big data solutions for formal method problems. To cope with RDECSs, Petri nets has been extended and developed by several works [14]. Reconfigurable Timed Net Condition/Event System (R-TNCES) is a novel formalism proposed in [18], where reconfiguration and time properties with modular specification are provided in the same formalism. This Paper deals with RDECSs modeled by R-TNCES. Authors in [2] developed a CTL Model checker in the cloud using map-reduce. The basic idea is to increase computation power and data availability to reduce time execution. They perform distributed fixed-point algorithm. However, the authors do not consider the system model similarities, which involves redundant calculations during verification. Moreover, this verification method support only simple CTL properties expressed with a restricted number of operator-quantifier combinations. Both of layer-by-layer verification proposed in [20] and the formal verification method proposed in [6] focused on the improvement of the state space generation phase, thus, they neglect state space analysis. The authors in [17] proposed automatic properties classification and introduced a priority order during RDECSs verification to control the high number of properties. The said method improves verification by reducing the number of properties by proposing an approach for exploiting relationships among them (equivalence, composition and dominance). In [16], the authors proposed Reconfigurable Computation Tree Logic R-CTL as an extension of CTL. This logic adds properties relationships management to deals redundancy caused by relationships (dominance, composition, and equivalence). RCTL improves version of CTL in terms of expressiveness, However processing RCTL properties verification on the generated space in a sequential way remains hard. Authors in [7] proposed a new method for state space generation, which extends classical accessibility graphs (AGs) to timed accessibility graphs (TAGs). The said method is efficient when dealing with reconfigurable real-time systems, it allows us to control complexity in the analysis step during verification. In our previous works, we propose a method for state space generation which, considers similarities that an R-TNCES can contain, thanks to an ontology-based history. Also, we proposed in [5] to perform CTL properties verification in a parallel way on a cloud-based architecture while considering relationships among properties. The said methods are efficient; However, the first work is only focused on state space generation, and the second one presents limits when properties are complex and properties relationship rate is low. Therefore, we propose in this paper a new work that comes to fill the limits of precedent ones. Hence, we proposed a new method that aims to improve R-TNCES formal verification. Reconfigurable Real-time systems formal verification may be expensive in terms of computation power and memory occupation, therefore, we resort to a cloud-based solution to increase computation power (resp. memory occupation) thank to the Infrastructure as a service IaaS (reps. Simple Storage Service S3) proposed by Amzon [8]. To control systems formal verification complexity we propose the following contributions:

1. Incremental timed state space generation to facilitate the access to different parts of the accessibility graph; Certain properties do not require the entire exploration of the accessibility graph in order to be validated or not, therefore, a partial exploration of the accessibility graph is sufficient. Indeed, we introduce the modularity and the time concepts to the state-space generation step, which allows us to access different parts of the accessibility graph (modules) and help us to face time constraints. This contribution allows us to proceed to a targeted verification.
2. Decomposition of CTL properties to control complexity during the state-space analysis. Due to the systems complexity, properties to be verified in order to ensure the correctness of the system behavior are more complex. Thereby, increasing the complexity of the analysis step. In order to fix the mentioned issue, we check the possibility of decomposition of the complex properties into several simple or less complex properties that can be verified in less computation time using fewer resources.
3. Development of a distributed cloud-based architecture to perform parallel computations during formal verification and to store large scale data. The huge generated state spaces, the high number of properties to be verified, and time constraints forced us to opt for a big data solution to control the complexity of reconfigurable real-time systems formal verification. Computation tasks are ensured by the master and the workers via virtual machines allocated thanks to the EC2 product proposed by amazon. Data storage is ensured by S3.

This paper is an extended version of our previous paper [3], presented at the 'IC-SOFT 2020' conference. The method improves by

– Replacing classical accessibility graphs by the Timed accessibility graphs proposed in [7].
– Using temporal logic TCTL in addition to CTL in order to respond to the real-time constraints.
– Updating the proposed cloud-based architecture to deal with the verification of reconfigurable real-time constraints.

The main objective of this paper is to propose a new formal verification method that improves the classical ones by controlling complexity. As a running example, a formal case study is provided to demonstrate the relevance of our contributions. The obtained results are compared with different works. The comparison shows that the verification is improved in terms of execution time (i.e., less complexity to perform systems formal verification).

The remainder of the paper is organized as follows. Section 2 presents some required concepts. The distributed formal verification is presented in Sect. 3. Section 4 presents the evaluation of the proposed method. Finally, Sect. 5 concludes this paper and gives an overview about our future work.

2 Background

In this section, we present basic concepts which are required to follow the rest of the paper.

2.1 Reconfigurable Timed Net Condition/Event System

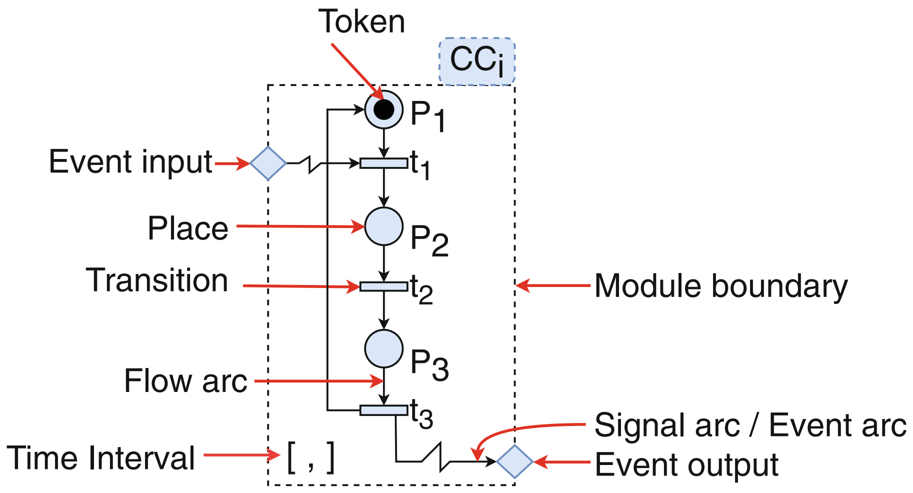

Fig. 1. Graphical model of a generic control component modeled by TNCES [3].

R-TNCES is a modeling formalism used to specify and verify reconfigurable Real Time Systems. R-TNCES is based on Petri nets and control components CCs. A control component (CC) is defined as a software unit. Control components are applied as a formal model of the controller of a physical process and are modeled by TNCES as shown in Fig. 1. Each CC resumes the physical process in three actions: Activation, working and termination. An R-TNCES RTN is defined in [20] as a couple $RTN = (B, R)$, where R is the control module and B is the behavior module. B is a union of multi TNCES-based CC modules, represented by

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

where,

a) P (resp, T) is a superset of places (resp, transitions),
b) F is a superset of flow arcs,
c) 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 $x, y \in P \cup T$,
d) CN (resp, EN) is a superset of condition signals (resp, event signals),
e) DC is a superset of time constraints on input arcs of transitions,
f) $V : T \rightarrow \wedge, \vee$ maps an event-processing mode for every transition;
g) $Z_0 = (M_0, D_0)$, where M_0 is the initial marking, and D_0 is the initial clock position.

R is a set of reconfiguration functions $R = \{r_1, ..., r_n\}$. r is structured as follow: $r = (Cond, s, x)$ such that:

1. $Cond \rightarrow$ {true, false} is the pre-condition of r, which means specific external instructions, gusty component failures, or the arrival of certain states.
2. s: $TN(^*r) \rightarrow TN(r^*)$ such that $TN(^*r)$(resp. $TN(r^*)$) be the original (resp. target) TNCES before (resp. after) r application is the structure modification instruction.
3. x: $last_{state}(TN(^*r)) \rightarrow initial_{state}(r^*)$ is the state processing function, where $last_{state}(TN(^*r))$ (resp. $initial_{state}(TN(r^*))$) is the last (resp. the initial) state of $TN(^*r)$ (resp. $TN(r^*)$).

2.2 Timed Accessibility Graph

Timed accessibility graphs is an extension on accessibility graphs proposed in [7], during model-checking it allows us to control verification complexity thank its time property. Timed accessibility graph (TAG) of a TNCES TNS is a structure tAG given by

$$tAG(St, Ed, S_O)$$

where,

– St denotes the set of reachable states;
– Ed: $St \rightarrow St$ denotes the set of edges that defines state-transitions such that each edge is labeled by the executed step;
– s_0 denotes the initial state.

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

$$State(M_p, Pclocks, D)$$

where,

– M_p is the set of marked places;
– $Pclockst$ s is a vector of integers representing places clock positions;
– D is the delay of the state which denotes the minimal number of time units after which at least one step becomes enabled.

2.3 Computation Tree Logic CTL

Computational tree logic CTL is a temporal logic for branching-time based on propositional logic used by [1] for model checking. CTL can describe the context and branching of the system state, it models system evaluation as a tree-like structure where each state can evolve in several ways (i.e., specify behavior systems from an assigned state in which the formula is evaluated by taking paths). CTL has a two-stage syntax where formulae in CTL are classified into state and path formulae. The former is formed according to the following grammar:

$$\Phi ::= true|AP|\Phi_1 \wedge \Phi_2|\Phi_1 \vee \Phi_2|\neg\Phi|E\varphi|A\varphi$$

While path formulae which express temporal properties of paths are formed according to the following grammar:

$$\varphi ::= X\Phi|F\Phi|G\Phi|\Phi_1 U \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae. AP is the set of atomic propositions. The CTL syntax include several operators for describing temporal properties of systems: A (for all paths), E (there is a path), X(at the next state), F (in future), G (always) and U (until).

Definition 1. *Equivalence of CTL Formulae: CTL formulae σ_1 and σ_2 (over AP) are called equivalent, denoted $\sigma_1 \equiv \sigma_2$ whenever they are semantically identical. Therefore, $\sigma_1 \equiv \sigma_2$ if $Sat(\sigma_1) = Sat(\sigma_2)$ for all transition systems TS over AP such that Sat(σ) = $\{s \in S|s \models \sigma\}$. Table 1 presents an important set of equivalences rules (expansion and distributive laws).*

Table 1. Some equivalence rules for CTL.

Expansion laws
$EG\phi \equiv \phi \wedge EXEG\phi$
$AF\phi \equiv \phi \vee AXAF\phi$
$EF\phi \equiv \phi \vee EXEF\phi$
$A[\phi U \psi] \equiv \psi \vee (\phi \wedge AXA[\phi U \psi])$
$E[\phi U \psi] \equiv \psi \vee (\phi \wedge EXE[\phi U \psi])$
Distributive laws
$AG(\sigma_1 \wedge \sigma_2) \equiv AG\sigma_1 \wedge AG\sigma_2$
$EF(\sigma_1 \vee \sigma_2) \equiv EF\sigma_1 \vee EF\sigma_2$

2.4 Infrastructure as a Service IaaS

Cloud computing is an increasingly popular paradigm for ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. In practice, cloud service providers tend to offer services that can be grouped into three categories as follows:

(i) software as a service,
(ii) platform as a service, and
(iii) infrastructure as a service presented in Fig. 2.

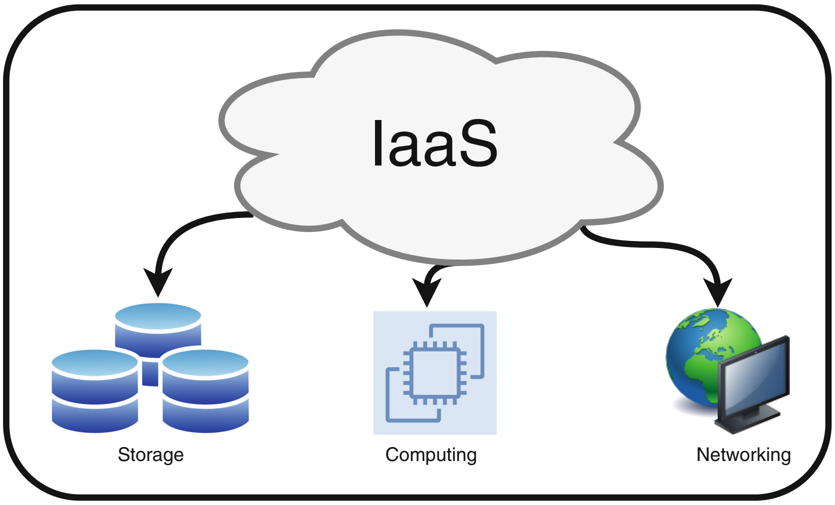

Fig. 2. Infrastructure as a Service.

IaaS is defined by [8] as web service that provides provision processing, storage, networks, administrative services needed to store applications and a platform for running applications [8]. It is designed to make web-scale cloud computing easier for developers. Amazon Web Services Elastic Compute Cloud (EC2) and Secure Storage Service (S3) are examples of IaaS offerings as shown in Fig. 2.

3 Distributed Cloud Based Formal Verification

We present in this section the proposed distributed cloud-based formal verification of R-TNCESs.

3.1 Motivation

R-TNCES is an expressive formalism, which allows considering different aspects of Reconfigurable real-time systems (time, probability, reconfigurability and concurrency) [9]. The correctness of systems modeled by R-TNCES can be ensured by formal verification. However, such a formalism makes the verification process complex, due to the combinatorial growth of the state space according to the model size, and due to the high number and complexity of the properties that the designer wants to verify. Thus, we aim to make model checking more efficient by reducing the time validation of properties to be verified. Therefore, we propose a new method for R-TNCES verification, which facilitates both generation and analysis of state space. To ensure our objective, we implement different tasks that can be presented in two parts as follows:

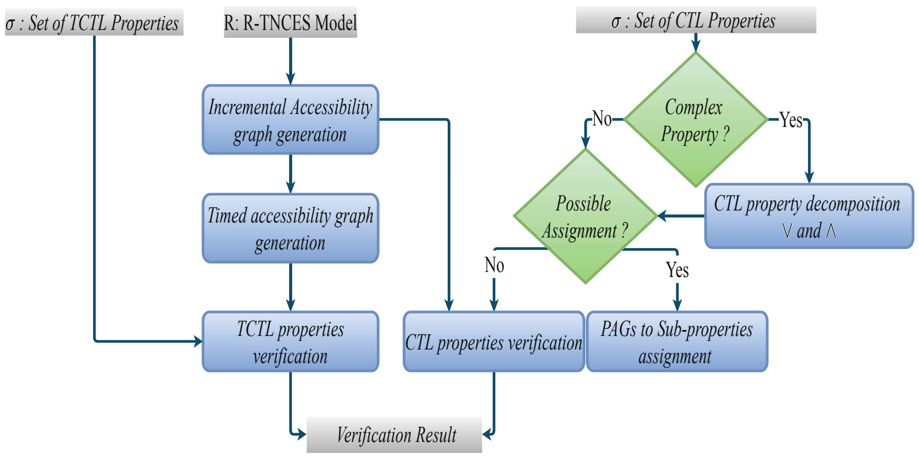
σ : Set of TCTL Properties
R: R-TNCES Model
σ : Set of CTL Properties
Incremental Accessibility graph generation
Complex Property ?
No
Yes
Timed accessibility graph generation
Possible Assignment ?
CTL property decomposition ∨ and ∧
No
Yes
TCTL properties verification
CTL properties verification
PAGs to Sub-properties assignment
Verification Result

Fig. 3. Global idea for the formal verification according to the distributed cloud based verification.

- *Part 1*: CTL properties verification:
  a) Incremental state space generation, which is to construct the state space by part,
  b) the complex properties are decomposed to simple or less complex ones, then
  c) if possible, we assign a partial graph to the property to be verified.
  d) Finally, we proceed to CTL properties verification.
- *Part 2*: TCTL properties verification:
  a) Timed state space generation, which is generated from accessibility graphs computed during part 1, then
  b) we proceed to TCTL properties verification.

Figure 3 presents scheduling of the presented tasks.

3.2 Formalization

In this section, we present formal verification steps according to the distributed cloud-based formal verification of R-TNCESs.

Incremental State Space Generation. Incremental state space generation consists of generating accessibility graphs by part, while preserving models semantics. Let $RTN(R, B)$ be an R-TNCES model, this task consists in two steps:

(i) Basic accessibility graph generation BAG, which consists of generating accessibility graphs for each $CC_i \in TNCES_j$ where, $i \in 0 \ldots NumberCC(TNCES_j)$ and $j \in 0 \ldots NumberTN(B)$. This step is implemented in Algorithm 1. It takes an R-TNCESs as input and proceed to BAGs generation through several function including $Generate\_State\_Space(CC)$ which, take a CC modeled by TNCES and return its accessibility graph using SESA tool [15].

Algorithm 1. Timed Basic accessibility generation.

```
Input: $RTN$: R-TNCES; $TN_0$: TNCES;
Output: $S\_TBAG$: Set of elementary accessibility graphs;
for int $i = 0$ to $| \sum TN |$ do
    for each $CC \in TN$ do
        if ( !Tagged (CC)) then
            $Insert(S\_BAG, Generate\_State\_Space(CC))$;
            tag(CC);
        end
    end
    for each $BAG \in S\_BAG$ do
        if ( !Tagged (BAG)) then
            $Insert(S\_BAG, Generate\_TBAG(BAG))$;
            tag(BAG);
        end
    end
end
return $S\_TBAG$
```

Algorithm 2. Accessibility graph construction.

```
Input: $S\_TBAG$: Set of Times basic accessibility graphs ; $\sum CChain$: Set of Cchains;
Output: $S\_AG$: Set Accessibility graphs;
for int $i = 0$ to $| \sum CChain |$ do
    $AG \leftarrow TBAG_{CC_i^0}$;
    for int $j = 0$ to $| \sum CC_i |$ do
        $AG \leftarrow Compose(AG, TBAG_{CC_i^j})$;
    end
    $Insert(S\_AG, AG)$
end
return $S\_AG$
```

(ii) Basic Timed Accessibility Graph BTAG Generation from a Graph BAG, which consists on generating a new graphs that consider time properties from another graph, we adopt the algorithm proposed in [7] to proceed to the TBAGs generation as shown in Fig. 4

(iii) Partial accessibility graphs (PAGs) composition: This step is implemented in Algorithm 2. It consists of composing pair of graphs computed during the first step (BAGs) and throughout iterations of the second step (PAGs), mainly by using the function $Compose(AG, AG)$ that takes two graphs and composes them and returns a new composed graph.

Complex CTL Properties Decomposition. We assume that properties that contain the operators ($\wedge$) or ($\vee$) are complex. Two kinds of complex properties are distinguished as follows:

– Decomposable: The operators ($\wedge$) or ($\vee$) are not linked to factors (State operators or path quantifiers). This kind of properties are directly splited into a set of sub-properties (e.g., $\Phi = P_1 \wedge P_2$ gives $\sigma_1 = P_1$ and $\sigma_2 = P_2$) (Fig. 5).

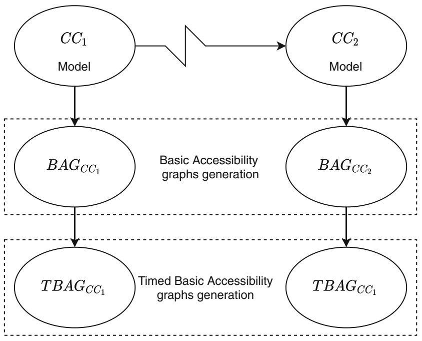
CC_1
Model
CC_2
Model
BAG_{CC_1}
Basic Accessibility graphs generation
BAG_{CC_2}
$TBAG_{CC_1}$
Timed Basic Accessibility graphs generation
$TBAG_{CC_1}$

Fig. 4. Timed Basic accessibility graph generation.

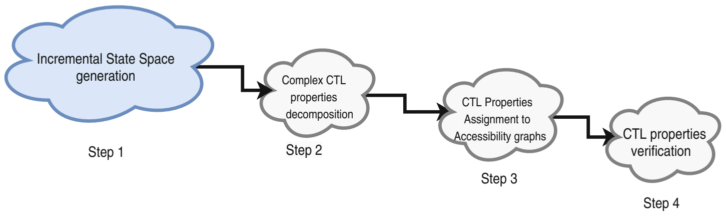
Incremental State Space generation
Step 1
Complex CTL properties decomposition
Step 2
CTL Properties Assignment to Accessibility graphs
Step 3
CTL properties verification
Step 4

Fig. 5. First step of reconfigurable real time systems verification.

– Non-decomposable: The operators $(\wedge)$ or $(\vee)$ are linked to factors. For this kind of properties, we firstly applied expansion or distribution laws and then re-check if they are decomposable or not (Fig. 6).

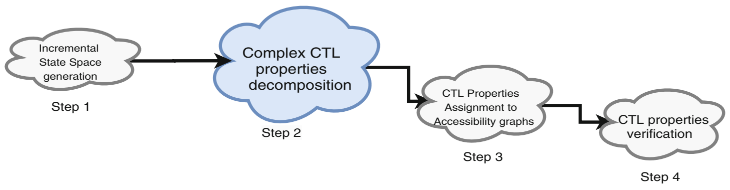
Incremental State Space generation
Step 1
Complex CTL properties decomposition
Step 2
CTL Properties Assignment to Accessibility graphs
Step 3
CTL properties verification
Step 4

Fig. 6. Second step of reconfigurable real time systems verification.

Figure 7 shows the majors steps of complex CTL properties decomposition task.

CTL Properties Assignment to PAGs. We assign to each property one or several state spaces computed during incremental state space generation. The assignment is done based on two criteria:

(i) Path quantifier and state operators, and
(ii) places concerned by the property such that we assign the smallest state space that contains the concerned places.

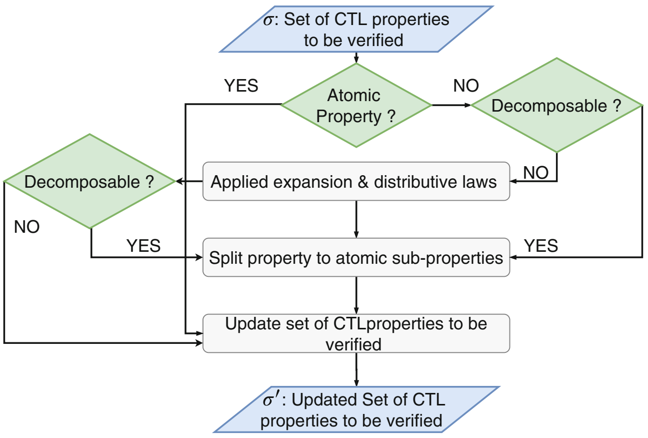
σ: Set of CTL properties to be verified
Atomic Property ?
YES
NO
Decomposable ?
NO
Applied expansion & distributive laws
Decomposable ?
NO
YES
Split property to atomic sub-properties
YES
Update set of CTLproperties to be verified
σ′: Updated Set of CTL properties to be verified

Fig. 7. Complex CTL properties decomposition.

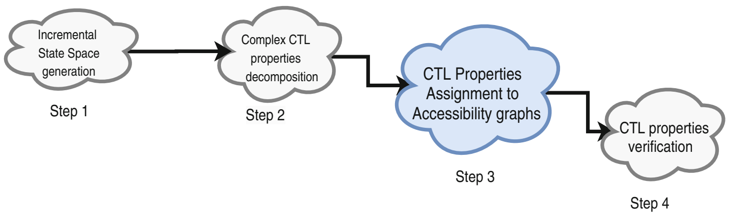
Incremental State Space generation
Step 1
Complex CTL properties decomposition
Step 2
CTL Properties Assignment to Accessibility graphs
Step 3
CTL properties verification
Step 4

Fig. 8. Third step of reconfigurable real time systems verification.

CTL Properties Verification. In short-term we integrate CTL properties verification method inspired from methods proposed in [5]. This method consider relationships which exist among properties to be verified (Equivalence, dominance and composition) and processes the verification in parallel way (Figs. 8 and 9).

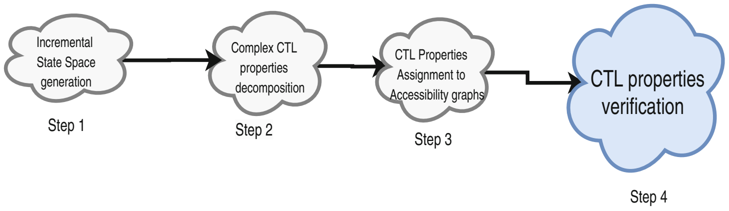
Incremental State Space generation
Step 1
Complex CTL properties decomposition
Step 2
CTL Properties Assignment to Accessibility graphs
Step 3
CTL properties verification
Step 4

Fig. 9. Fourth step of reconfigurable real time systems verification.

3.3 Distributed Architecture for Formal Verification

In this subsection, we present the proposed distributed cloud-based architectures shown in Fig. 10. The idea that motivates the development of this architecture is to increase

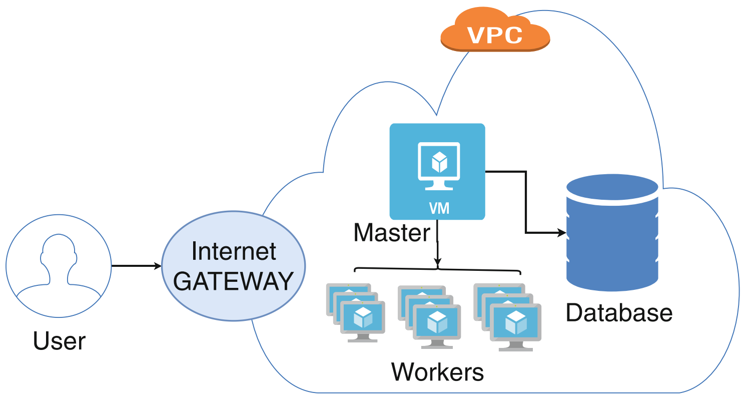

Fig. 10. Distributed cloud-based architecture.

computation power and storage availability. It is composed of computational and storage resources. To develop the architecture shown in Fig. 10, we use IaaS to allocate the following resources:

- Computation resources: which represent the master that coordinates the executed tasks, and the workers that execute the presented tasks above.
- Storage resources: represents the allocated cloud database that stores accessibility graphs computed during verification.

3.4 Reconfigurable Real-Time System Verification in a Distributed Cloud-Based Architecture

The Reconfigurable real-time system verification is performed on the proposed architecture as follow

- $Master$: has the coordinator role it:
  - Receives the verification request;
  - Sends to each worker the task to perform (Accessibility graph generation, Properties decomposition and assignment, and CTL or TCTL properties verification);
  - Stores and retrieves data from storage unit.
- $workers$: perform different tasks received from the Master and return the results.

Note that the TCTL properties are considered as non-decomposable, thus the Master distributes them to the works for parallel verification by considering the time constraints.

4 Experimentation

In this section, to validate and demonstrate the gain of our proposed contributions, we use a formal case study.

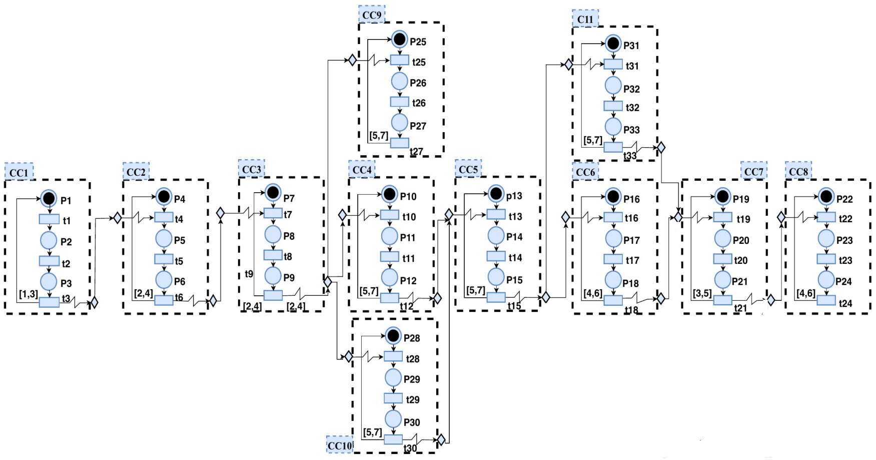

Fig. 11. Behavior model with three configurations process.

4.1 Case Study

To demonstrate the performance and the gain of the proposed contribution, we use R-TNCES formalism to model a sequential system S_{01}, used in the original conference paper [3], which is denoted by $RTN_{S_{01}}(B_{S_{01}}, R_{S_{01}})$. S_{01} is composed of 11 physical processes modeled by 11 CCs. The behavior module of the system ($B_{S_{01}}$) is modeled graphically as shown in Fig. 11. This model covers three configurations (C_1, C_2, C_3). It is assumed that every configuration has one control chain (C_{Chain_i}) as follows.

- C_{Chain_1}: CC_1, CC_2, CC_3, CC_9.
- C_{Chain_2}: $CC_1, CC_2, CC_3, CC_4, CC_5, CC_{11}, CC_7, CC_8$.
- C_{Chain_3}: $CC_1, CC_2, CC_3, CC_{10}, CC_5, CC_6, CC_7, CC_8$.

This behavior module can be reconfigured automatically and timely between the three configurations ($C_i, i = 1, ..., 3$), according to the environment changes or to the user requirements. $RTN_{S_{01}}$ can apply six different reconfiguration scenarios according to the control module $R_{S_{01}}$, which are described as follows: $R_{S_{01}} = (C_1, C_2)$; (C_1, C_3);(C_2, C_1); (C_2, C_3); (C_3, C_1); (C_3, C_2).

4.2 Application

In this section, we present the application of the formal verification of $RTN_{S_{01}}$ according to the cloud-based formal verification.

Incremental State Space Generation. In order to generates $RTN_{S_{01}}$ accessibility graph $AG_{RTN_{S_{01}}}$, we apply Algorithms 1 and 2. First, we generates accessibility graphs for each physical process, which are denoted by ($TBAG_i$, $i =_{1, ..., 11}$). Then, we proceed to successive pair graphs compositions until we constitute $AG_{RTN_{S_{01}}}$. Table 2

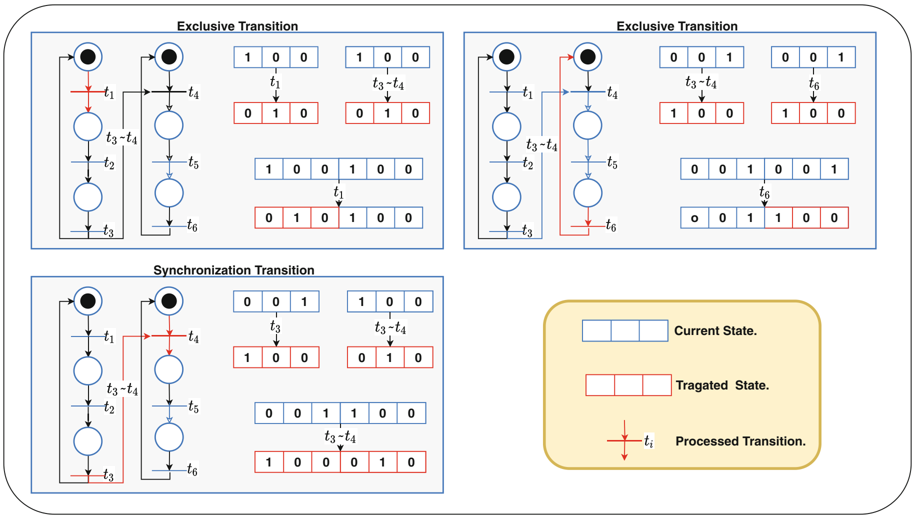

Fig. 12. Example of accessibility graphs composition according to the transition type.

Table 2. Incremental state space generation [3].

R-TNCES model	Control chains	PAGs
$RTN_{S_{01}}$	$CChain_1$	$(CC_1, CC_2); (CC_1, CC_2, CC_3); (CC_1, CC_2, CC_3, CC_9)$
	$CChain_2$	(CC_1, CC_2, CC_3, CC_4), $(CC_1, CC_2, CC_3, CC_4, CC_5)$, $(CC_1, CC_2, CC_3, CC_4, CC_5, CC_{11})$, $(CC_1, CC_2, CC_3, CC_4, CC_5, CC_{11}, CC_7)$, $(CC_1, CC_2, CC_3, CC_4, CC_5, CC_{11}, CC_7, CC_8)$
	$CChain_3$	$(CC_1, CC_2, CC_3, CC_{10})$, $(CC_1, CC_2, CC_3, CC_{10}, CC_5)$, $(CC_1, CC_2, CC_3, CC_{10}, CC_5, CC_6)$, $(CC_1, CC_2, CC_3, CC_4, CC_5, CC_6, CC_7)$, $(CC_1, CC_2, CC_3, CC_4, CC_5, CC_6, CC_7, CC_8)$

shows PAGs computed during the first step of the system verification. Note that each computed PAG is stored in the cloud database. Moreover, Fig. 12 shows an example of a pair graphs composition.

Decomposition of CTL Properties. In order to validate the basic behavior of the system and to guarantee that system model satisfies the good requirements, we must ensure the CTL functional properties. In particular, to ensure: a) The safety, the system allows only one process to be executed at any time, i.e., no activation of two CCs from two different configurations at the same time, b) the liveness, whenever any process wants to change the configuration, it will eventually be allowed to do so, and c) the non-blocking, any active CC is eventually ended. Table 3 presents the above mentioned properties specified by CTL.

Table 3. Set of CTL properties to be verified [3].

σ: Set of CTL Properties
P_1: $EF(p_3)$, P_2: $AF(p_9)$,
P_3: $AF(p_{15})$, P_4: $AF(p_{21})$,
P_5: $AF(p_{24})$, P_6: $AF(p_{17})$,
P_7: $AF(p_{32})$, P_8: $AF(p_{35})$
P_9: $EF(p_{12} \wedge EG(p_{24}))$,
P_{11}: $EG(p_{12} \wedge EGp_{35}))$,
P_{12}: $EG(p_{12} \wedge EG(p_{33}))$,
P_{13}: $\neg EF(p_{27} \wedge EG(p_{24}))$,
P_{14}: $EF(p_{12}) \wedge EF(p_{18})$,
P_{15}: $AF(p_{12}) \wedge EG(p_{33}) \wedge EG(p_{21}) \wedge AF(p_{24})$,
P_{16}: $AF(p_{12}) \wedge EG(p_{33}) \wedge EG(p_{21}) \wedge AF(p_{24})$

Table 4. CTL properties decomposition and assignment [3].

σ: Set of CTL properties	Decomposition	Assignment
P_1: $EF(p_3)$, P_2: $AF(p_9)$, P_3: $AF(p_{15})$, P_4: $AF(p_{21})$, P_5: $AF(p_{24})$, P_6: $AF(p_{17})$, P_7: $AF(p_{32})$, P_8: $AF(p_{35})$	Non-decomposable	P_1: $TBAG_1$, P_2: CC_1, CC_2, CC_3, P_3: $CC_1, \ldots, CC_5$, P_4: $CC_1, \ldots, CC_7$, P_5: $CC_1, \ldots, CC_8$, P_6: $CC_, \ldots, CC_6$, P_7: $CC_, \ldots, CC_{11}$
P_9: $EF(p_{12} \wedge EG(p_{24}))$, P_{12}: $EG(p_{12} \wedge EG(p_{33}))$, P_{13}: $\neg EF(p_{27} \wedge EG(p_{24}))$,	Non-decomposable	P_9: $CC_1, \ldots, CC_8$, P_{12}: $CC_1, \ldots, CC_{11}$, P_{13}: $CC_1, \ldots, CC_8$
P_{14}: $EF(p_{12}) \wedge EF(p_{18})$	P'_{14}: $EF(p_{12})$ P''_{14}: $EF(p_{18})$	P'_{14}: $CC_1, \ldots, CC_4$ P''_{14}: $CC_1, \ldots, CC_6$
P_{15}: $AF((p_{12}) \wedge EG((p_{33})) \wedge EG(p_{21})) \wedge AF(p_{24})$	P'_{15}: $AF(p_{12})$ P''_{15}: $EG((p_{33}) \wedge EG(p_{21}))$ P'''_{15}: $AF(p_{24}$	P'_{15}: $CC_1, \ldots, CC_4$, P''_{15}: $CC_1, \ldots, CC_7$, P'''_{15}: $CC_1, \ldots, CC_8$
P_{16}: $AF(p_{12}) \wedge EG(p_{33}) \wedge EG(p_{21}) \wedge AF(p_{24})$	P'_{16}: $AF(P_{12})$ P''_{16}: $\neg AF(p_{30})$	P'_{16}: $CC_1, \ldots, CC_4$, P''_{16}: $CC_1, \ldots, CC_{10}$

Assignment of CTL Properties to PAGs. We apply the possible decomposition to the CTL properties in σ, then we assign each property to be verified to the correspondent accessibility graph (BAG or PAG). The results are shown in Table 4.

CTL and TCTL Properties Verification. CTL properties are distributed according to their assignment one by one on workers by the Master, then workers proceed to their verification using the SESA tool [15]. Where, TCTL properties are non-decomposable thus their distribution depends on time constraints. Table 5 shows a set of TCTL properties. The order of verification of the mentioned properties is: $P_1, P_2 > P_3, P_4, P_6 > P_7 > P_5$.

Table 5. TCTL properties to be verified.

$\sigma_{F_{MPS}}$: Set of TCTL properties
P_1: $EF[1,3]p_3 = 1$,
P_2: $EF[10,18]p_9 = 1$,
P_3: $EF[10,28]p_{12} = 1$,
P_4: $EF[9,41]p_{18} = 1$,
P_5: $EF[26,43]p_{24} = 1$,
P_6: $EF[10,18]p_{27} = 1$,
P_7: $EF[20,42]p_{33} = 1$,

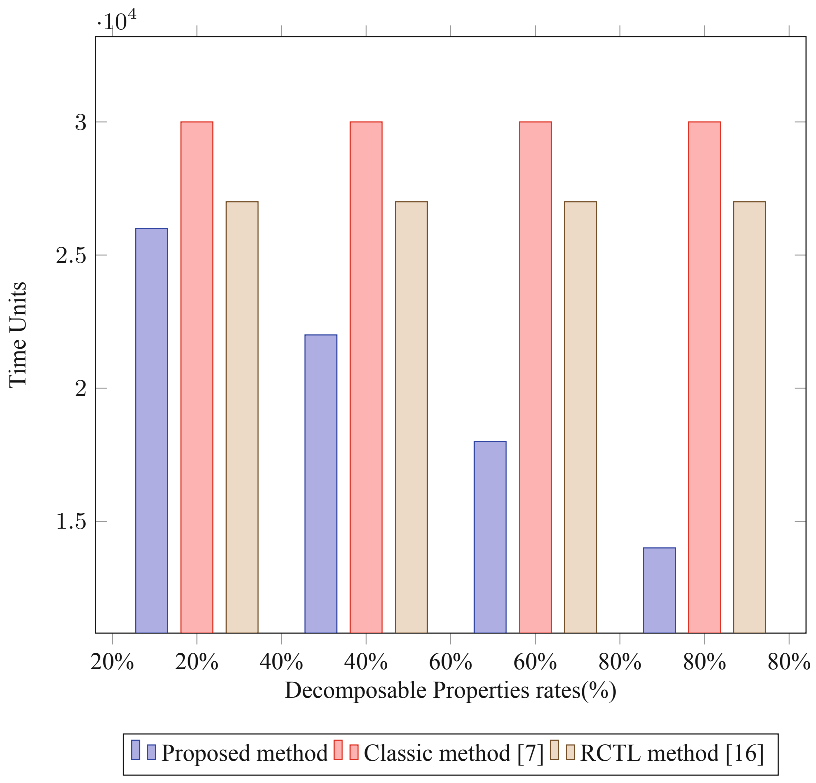

Fig. 13. Classic methods VS Proposed method.

4.3 Evaluation

In this subsubsection, the evaluation of the proposed method is presented considering two factors: The decomposable rate and the complex CTL properties rate.

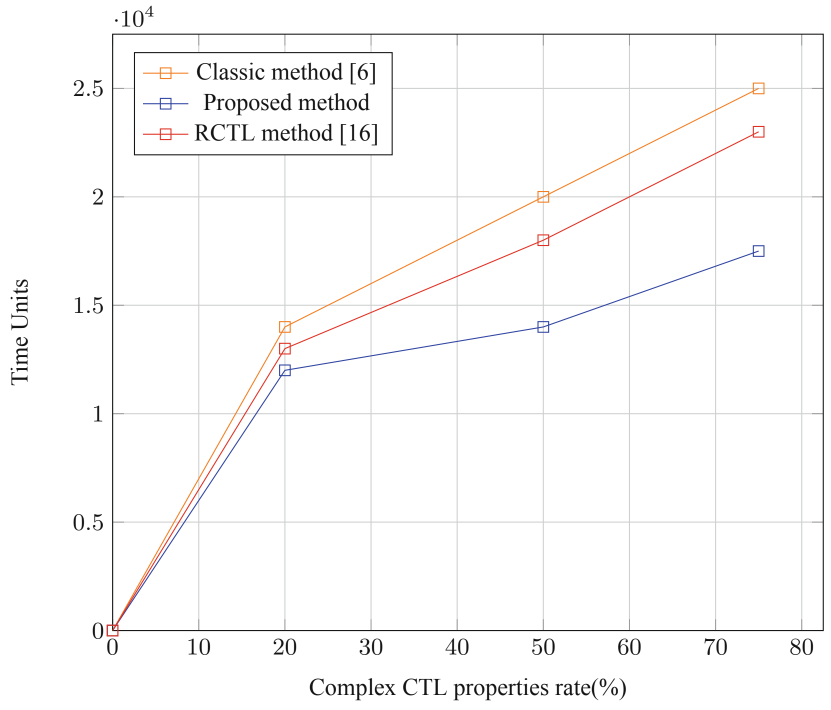

Fig. 14. Improved performance of proposed method verification.

Evaluation of CTL Properties Verification Method Considering Different Decomposable Ratex. Let assume we have to verify a system model with 2500 TNCESs. In order to ensure the well-behave of the system we have to verify at least 4 properties for each TNCES. Thus, we need to verify 10000 CTL properties. We assume that the properties to be verified are complex and the rate of decomposable one can be:

(i) Low in $0, 20\%$,
(ii) $Medium$ in $20, 60\%$, or
(iii) $High$ when more than 60%.

The results show in Fig. 13 that the gain increases proportionally to decomposable properties rate. Thus, the gain is clearly shown when similarity rate is 'High'. This is explained by the fact that, when properties are decomposed their verification is less complex [4].

Evaluation of CTL Properties Verification Method Considering Complex CTL Properties Rate. Figure 14 we can observe an important gain when performing parallel verification thanks to the proposed architecture. This gain is explained by the fact that the proposed architecture allows us to reduce considerably times execution by

(i) Avoiding redundant calculation,
(ii) Avoiding wait time execution,
(iii) Performing several properties verification at the same time.

5 Conclusion

This work deals with the formal verification of reconfigurable real-time systems modeled by R-TNCES using CTL and TCTL specifications. In this paper, we present a cloud-based solution for the formal verification problem. A distributed cloud-based architecture is developed with two hierarchical levels (Master and worker) where, data storage is ensured by Amazon Simple Storage S3 (Murty, 2008)). It allows us to increase computational power, data availability, and to perform parallel execution. The proposed method aims to improve state space analysis by using a hybrid distributed cloud-based architecture for computation tasks. Developed architecture is composed of:

1. A local workstation, where simple computation tasks are executed. First, a classification algorithm is applied in order to distinguish between simple and composed properties. Then, we compute a matrix relationships that mention any eventual relationship between each couple of properties. Finally, we generate a parallelization tree that we explore to extract a suitable execution order for each property to be verified.
2. A virtual workstation, where complex tasks are computed. Virtual machines use SESA tool to perform CTL properties verification and stores results in the shared memory.

We introduce the modularity and the timed concept to the generated state spaces which allow us to execute the generation step in a parallel way via several workers (virtual machines) and to deal with time constraints. We detect the complex CTL properties and decompose them into several simple or less complex properties, then proceed to their verification via workers using the SESA tool [15]. The TCTL properties are them verified following the order established by the master according to the time constraints. Incremental Timed state space generation and the decomposition of CTL properties allow us to run a targeted verification, which is less complex and more efficient in terms of execution time. This work opens several perspectives; first, we plan to apply our approach in the verification of real-case with complex properties to check the functional and the temporal specifications. Then, automatize the detection of complex properties by using the IA thanks to ontologies. Also, we plan to introduce a deep learning method to detect similar behavior of systems, which will allow us to reduces complexity during verification. Besides, To apply our methodology in the verification process of many research fields in (i) smart systems like smart grids, (ii) robotics, (iii) vehicular technologies, and other more evaluations of the proposed contributions.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
2. Camilli, M., Bellettini, C., Capra, L., Monga, M.: CTL model checking in the cloud using mapreduce. In: 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 333–340. IEEE (2014)
3. Choucha, C.E., Ramdani, M., Khalgui, M., Kahloul, L.: On decomposing formal verification of CTL-based properties on IaaS cloud environment. In: Proceedings of the 15th International Conference on Software Technologies, Volume 1: ICSOFT, pp. 544–551. INSTICC, SciTePress (2020). https://doi.org/10.5220/0009972605440551
4. Choucha, C.E., Salem, M.B., Khalgui, M., Kahloul, L., Ougouti, N.S.: On the improvement of R-TNCESs verification using distributed cloud-based architecture. In: Proceedings of the 15th International Conference on Software Technologies, Volume 1: ICSOFT, pp. 339–349. INSTICC, SciTePress (2020). https://doi.org/10.5220/0009836103390349
5. Choucha, C.E., Ougouti, N.S., Khalgui, M., Kahloul., L.: R-TNCES verification: distributed state space analysis performed in a cloud-based architecture. In: Proceedings of the the 33rd Annual European Simulation and Modelling Conference, pp. 96–101. ETI, EUROSIS (2019)
6. Hafidi, Y., Kahloul, L., Khalgui, M., Li, Z., Alnowibet, K., Qu, T.: On methodology for the verification of reconfigurable timed net condition/event systems. IEEE Trans. Syst. Man Cybern. Syst. **99**, 1–15 (2018)
7. Hafidi, Y., Kahloul, L., Khalgui, M., Ramdani, M.: New method to reduce verification time of reconfigurable real-time systems using R-TNCESs formalism. In: Damiani, E., Spanoudakis, G., Maciaszek, L.A. (eds.) ENASE 2019. CCIS, vol. 1172, pp. 246–266. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40223-5_12
8. Hayes, B.: Cloud computing. Commun. ACM **51**(7), 9–11 (2008)
9. Housseyni, W., Mosbahi, O., Khalgui, M., Li, Z., Yin, L., Chetto, M.: Multiagent architecture for distributed adaptive scheduling of reconfigurable real-time tasks with energy harvesting constraints. IEEE Access **6**, 2068–2084 (2017)
10. Järvensivu, M., Saari, K., Jämsä-Jounela, S.L.: Intelligent control system of an industrial lime kiln process. Control. Eng. Pract. **9**(6), 589–606 (2001)
11. Khalgui, M., Hanisch, H.M.: Reconfiguration protocol for multi-agent control software architectures. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **41**(1), 70–80 (2011)
12. Khalgui, M., Mosbahi, O., Li, Z., Hanisch, H.M.: Reconfiguration of distributed embedded-control systems. IEEE/ASME Trans. Mechatron. **16**(4), 684–694 (2011)
13. Koubâa, A., Qureshi, B., Sriti, M.F., Javed, Y., Tovar, E.: A service-oriented cloud-based management system for the internet-of-drones. In: 2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 329–335. IEEE (2017)
14. Padberg, J., Kahloul, L.: Overview of reconfigurable Petri nets. In: Heckel, R., Taentzer, G. (eds.) Graph Transformation, Specifications, and Nets. LNCS, vol. 10800, pp. 201–222. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75396-6_11
15. Patil, S., Vyatkin, V., Sorouri, M.: Formal verification of intelligent mechatronic systems with decentralized control logic. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), pp. 1–7. IEEE (2012)
16. Ramdani, M., Kahloul, L., Khalgui, M., Li, Z., Zhou, M.: RCTL: new temporal logic for improved formal verification of reconfigurable discrete-event systems. IEEE Trans. Autom. Sci. Eng. 1–14 (2020). https://doi.org/10.1109/TASE.2020.3006435
17. Ramdani, M., Kahloul, L., Khalgui, M.: Automatic properties classification approach for guiding the verification of complex reconfigurable systems. In: ICSOFT, pp. 625–632 (2018)
18. Zhang, J., et al.: R-TNCES: a novel formalism for reconfigurable discrete event control systems. IEEE Trans. Syst. Man Cybern. Syst. **43**(4), 757–772 (2013)

19. Zhang, J., et al.: Modeling and verification of reconfigurable and energy-efficient manufacturing systems. Discret. Dyn. Nat. Soc. **2015** (2015)
20. Zhang, J., Khalgui, M., Li, Z., Mosbahi, O., Al-Ahmari, A.M.: R-TNCES: a novel formalism for reconfigurable discrete event control systems. IEEE Trans. Syst. Man Cybern. Syst. **43**(4), 757–772 (2013)