New Verification Approach for Reconfigurable Distributed Systems

Oussama Khlifi[1,2,3,5], Olfa Mosbahi[2], Mohamed Khalgui[3,4] and Georg Frey[1,5]
[1]*Chair of Automation, Saarland University, Saarbrücken, Germany*
[2]*LISI laboratory, INSAT, University of Carthage, Tunis, Tunisia*
[3]*Polytechnic School of Tunisia, University of Carthage, Tunis, Tunisia*
[4]*School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China*
[5]*ZeMA – Zentrum fur Mechatronik und Automatisierungstechnik gemeinnützige GmbH, Saarbrücken, Germany*

Keywords: Formal Verification, Model Checking, Adaptive Distributed Systems.

Abstract: Adaptive systems are able to modify their behaviors to cope with unpredictable significant changes at run-time such as component failures. These systems are critical for future project and other intelligent systems. Reconfiguration is often a major undertaking for systems: it might make its functions unavailable for some time and make potential harm to human life or large financial investments. Thus, updating a system with a new configuration requires the assurance that the new configuration will fully satisfy the expected requirements. Formal verification has been widely used to guarantee that a system specification satisfies a set of properties. However, applying verification techniques at run time for any potential change can be very expensive and sometimes unfeasible. In this paper, we propose a new verification approach to deal with the formal verification of these reconfiguration scenarios. New reconfigurable CTL semantics is introduced to cover the verification of reconfigurable properties. It consists of two verification steps: design time and run-time verification. A railway case study will be also presented.

1 INTRODUCTION

Adaptive discrete event control systems are dynamic and evolve according to occurrence of discrete event signals (Zhang et al., 2013). Examples of such systems are used to solve future complex mission needs in space exploration and railway train control. They include a variety of man-made systems such as flexible manufacturing systems, complex computer programs and communication systems (Li and Zhou, 2009). They are able to change their behaviors with an unpredictable way during run-time processes. Reconfigurations are qualitative changes in the structure, functionality, and algorithms of a control system. A reconfiguration scenario is also assumed to be any addition, removal or update of tasks and resources (Salem et al., 2015). This is due to qualitative changes of the controlled system or the environment within which the system behaves (Salem et al., 2015). Recently, we have seen an increase in the deployment of safety critical embedded systems in rapidly changing environments, as well as need for on-site customizations and rapid adaptation. However, since there has been no information about the behavior of the new configuration at design time, it is necessary to reason about its impact and negative side effects on the overall system behavior at run time since it might make functions of the system unavailable for some time (Sharifloo et al, 2013).

Although necessary, adaptations can cause inconsistent and unstable configurations that must be prevented for the embedded system to remain dependable and safe. Therefore, verifying such systems before they are deployed is essential because there are limited to no opportunities to effectively monitor and adjust their behavior during operation (Sharifloo et al, 2013). Formal verification has been widely used to guarantee that a system specification satisfies a set of properties (Kalita et al., 2002). The existing methods to certify reconfigurable systems mainly focus on the specification and verification of adaptation process: These approaches are based on a complete knowledge of the system and the environment behavior at design time, so they are able to reason about the properties of the whole interaction model (Bortolussi et al., 2015). However, this is not the case in many realistic examples in which the information about the behavior of some components and the environment are obtained only at run time.

Khlifi, O., Mosbahi, O., Khalgui, M. and Frey, G.
New Verification Approach for Reconfigurable Distributed Systems.
DOI: 10.5220/0006434003550362
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 355-362
ISBN: 978-989-758-262-2

This is why run-time verification techniques come into play to monitor and check that the running system does not violate the specification and the properties (Bortolussi et al., 2015). Although it is less expensive than model checking but it still not complete, and do not guarantee the satisfaction of the properties. Nevertheless, we find some limits in the temporal logic CTL for the optimal verification of adaptive properties.

To avoid any requirement violation, we have to guarantee that all the properties will be satisfied in case of applying any reconfiguration scenario (Sharifloo et al, 2013). This could be guaranteed by formally verifying the new system specification, which is obtained by integrating the specification of the new configuration, against the properties. Intuitively, it is an extra work and overhead because the major part of the specification does not change. Moreover, model checking a large specification at run-time at each reconfiguration is really difficult because of the time and resource limitations (Sharifloo et al, 2013). Thus, once it is possible to refer to the verification results of the invariant part for future verifications, this would significantly save the time and resource usage. This is why verification techniques to be proposed in this paper should verify all behaviors of the reconfigurable systems. We address run-time model checking of reconfigurable systems which are seen as systems with changing or unstable specifications. We focus on components based reconfigurable systems represented by an extension of Labelled Transition System and a model checking approach based on Reconfigurable Computation Tree Logic (CTL) (Zhang et al., 2013). More specifically, this approach allows the designer to verify the system at design time, even if some components are not fixed (unstable, can be replaced). The proposed model checking approach verifies if the requirements hold and produces a set of constraints for the unspecified components.

The paper is presented as follows: Section 2 describes the preliminaries on top of formal verification. Section 3 presents the railway network as a case study to show the problem statement. Section 4 introduces the proposed verification approach and its RCTL model checking. A discussion is presented in Section 5. The last Section concludes the paper.

2 BACKGROUND

We present in this section an overview of the temporal logic CTL. Some formal verification techniques such as model checking will be presented. The related works will be discussed in this part.

2.1 Computation Tree Logic

In CTL, all formulae specify behaviors of the system starting from an assigned state in which the formula is evaluated by taking paths (e.g. sequence of states) into account. The semantics of formulae is defined with respect to a reachability graph where states and paths are used for the evaluation (Axelsson et al., 2010). A reachability graph M consists of all global states that the system can reach from a given initial state. It is formally defined as a tuple $M = [Z, E]$ where:

- Z is a finite set of states,
- E is a finite set of transitions between states, e.g. a set of edges (z, z_0), such that z, $z_0 \in Z$ and z_0 is reachable from z.

In CTL, paths play a key role in the definition and evaluation of formulae. A path denoted by (z_i) starting from the state z_0 is a sequence of states, $(z_i) = z_0, z_1 \ldots$ such that $\forall j \in N$, there is an edge $(z_j, z_{j+1}) \in E$. The truth value of a CTL formula is evaluated with respect to a certain state of the reachability graph. Let $z_0 \in Z$ be a state of the reachability graph and ϕ be a CTL formula (Axelsson et al., 2010). The relation $z_0 \models \phi$ means that the CTL formula ϕ is satisfied in the state z_0. Then the relation $\models$ for a CTL formula is defined as follows:

- $z_0 \models EF\phi$, if there is a path (z_i) and $j > 0$ such that $z_j \models \phi$, ˆ
- $z_0 \models AF\phi$, if for all paths (z_i), there exists $j > 0$ such that $z_j \models \phi$.

2.2 Model Checking

Model checking is a technique to automatically verify the correctness properties of finite-state systems (Baier and Katoen, 2008). It is a general verification approach that is applicable to a wide range of applications such as embedded systems, software engineering, and hardware design. It also supports partial verification, i.e., properties can be checked individually, thus allowing focus on the essential properties first. It can be also easily integrated in existing development cycles since its learning curve is not very steep, and empirical studies indicate that it may lead to shorter development times (Baier and Katoen, 2008). Model checking is based on the reachability graphs of the system. SESA (Starke and Roch, 2002) is an effective software environment which analyses and 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.

2.3 Related Work

There have been a set of approaches to formally apply model checking techniques to verify the properties at design time (Schneider et al., 2006). Zhang and Cheng (Zhang and Cheng, 2006) introduce a modular verification algorithm to verify an adaptive system against the formulae expressed in A-LTL (Zhang et al., 2006). The system is represented as a state machine in which the states present the system configurations and transitions are adaptation actions. Xie and Zhe (Xie and Dang, 2004) propose a test-based approach for the verification of component-based systems, in which the behavior of some components is not specified. The system consists of a host system and a collection of unspecified components, which are represented as finite transition systems that synchronously communicate via a set of input/output symbols. Schaefer (Schaefer, 2008) has provided several approaches on verifying adaptive embedded systems specified as synchronous adaptive systems - high level representations of modelling concepts used in the MARS modelling approach (Trapp et al., 2007). The solution integrates model slicing of various granularities to reduce the complexity and enable automated model checking of the models by means of theorem proving. The technique is tested on adaptive vehicle stability control system. Goldsby et al. (Goldsby et al., 2008) provide the AMOEBA-RT model focused on run-time verification and monitoring. Wang et al. (Wang et al., 2007) have proposed usage of verification techniques to find the optimal schedule for energy constrained systems. Nevertheless, these works did not discuss how to optimize the formal verification of reconfigurable systems and their feasibility at run-time verification at each adaptation.

3 RUNNING EXAMPLE

The running example used through this paper is presented in this section. Rail transport is a means of conveyance of passengers and goods on wheeled vehicles running on rails. It is also commonly referred to as train transport. It is a complex and critical system because it deals with millions of human life every day. It is also faced to different challenges: safety from collisions and derailments and provide as maximum line capacity as possible for running many trains on the same line within the safety constraints (The Metro, 2017). These systems are considered to be reconfigurable distributed systems because the railway structure is not static: it is usually the subject of variant extension on different lines. It is also faced to numerous accident, structures breaking and natural disasters. Moreover, the number of trains is always changeable; it is possible to add extra trains to cover the increased demand and to maintain quality of service.

Similarly, rapidly increasing capacity is the biggest challenge facing all mass transit operators today. As major cities expand, so too does demand for high capacity and efficient railway network. Thus, the speed of trains is not constant for almost of the lines. Each change can be considered as an adaptation process that affects the characteristics of the system. As a real case study, the Paris Metro is a safety critical reconfigurable system. It is a large railway network with 14 main lines that cover 303 stations in the Paris area. It is mostly underground and it has 205 km of tracks. This system carried 1.5 billion passengers in 2014 (The Metro, 2017). The Metro system is an example of component-based systems whose safety properties depend on the dynamic components which are variable and change at run-time. Such systems require a continuous verification process to certify the correctness of the system at any new adaptation process.

This verification step should be as light-weight as possible to avoid intolerable overheads. The system is highly critical and its safety is the main propose of its existence. On the other side, the formal verification of the whole system at each adaptation process is considered to be unfeasible because of the resources and time limitation at run-time. We focus on the specification and verification part of the project. We present the system as a modular connected structure. It is a reconfigurable distributed system that can change its characteristics at run-time operation. Fig. 1 presents the abstract model of the system. It is a 14 module system that represents the different lines of the railway network. Each module represents one metro line with its trains and characteristics. It describes its capacity, structure and its connection to other lines. We assume that modules links represent the connections points between different lines of the railway network. The red rectangles are the system modules that represent the unstable lines: its characteristics are not fixed at

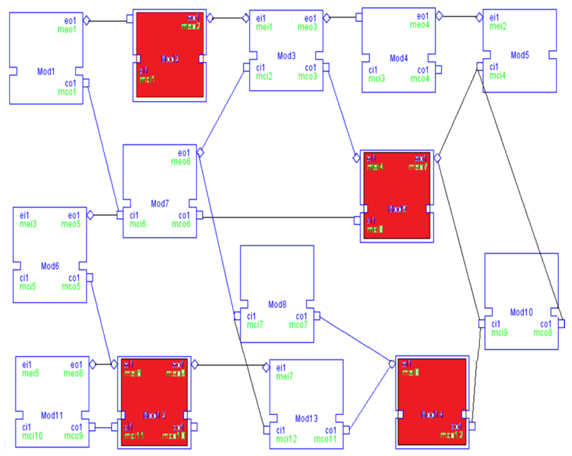


Figure 1: Reconfigurable railway network structure.

run-time. They are the object of new configurations to cope with the environment requirements at the current state of the system. These reconfigurations are due to an increase demand to enlarge the line capacity, the quality of service or to extend the line to new parts of the urban area of the city.

4 VERIFICATION APPROACH

The proposed model checking approach deals with distributed reconfigurable models, where a set of components or modules are considered to be unstable (change their behavior at run-time process) and could be also unspecified at design time and are known only at run-time. Moreover, the classical techniques enable to check the system every time the unspecified components are resolved or modified at design time. Indeed, the time and space required for the verification could be considerable and since many configurations are resolved only while the system is operating, the total overhead in resolving them has to be as small as possible. To get over this problem, we propose a two-phase verification approach that enables the designer to deal with reconfigurable scenarios and incomplete specification at design time and generate a set of constraints to be checked for the unstable parts of the system. Those constraints are verified at run-time against the new configuration of the component once it is available. A complete over view is given in Fig. 2. It presents two verification levels: at design time, the incomplete system is represented by a particular labelled transition system. It is an Incomplete Labelled Transition System dealing with specified and unspecified states. It contains two different states categories: the first are known as stable states which describe a predefined fixed part or task of the system. The second are known as unstable states to describe the reconfigurable scenarios of the system which are unknown only at design time or variable at run-time. The model is then checked against the desired Reconfigurable CTL properties "RCTL". The results of the verification process differ from the traditional model checker by an extra output namely "Conditionally True". This option generates a set of constraints to be checked against the reconfigurable module later. At run-time, only these constraints are checked against the new configuration and not the whole system specification as used before in the standard model checking.

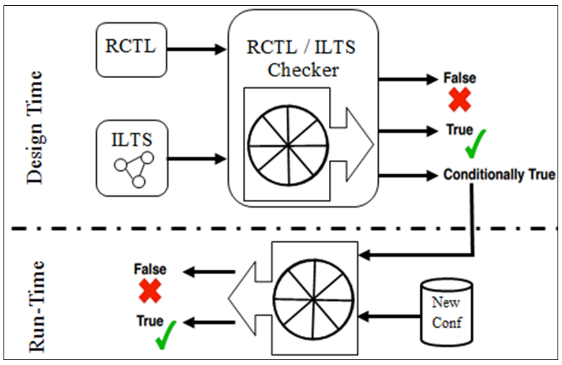


Figure 2: New verification approach.

4.1 Incomplete Labelled Transition System

An incompletely labelled transition system (ILTS) is a labelled transition system in which there are two sets of states: stable and unstable states. It can describe the unknown characteristics of the reconfigurable system at the specification step. Formally, it is a tuple (S, s_0, R, L) where:

- S is the set of stable states T and unstable states I, i.e., $S = T \cup I$ and $T \cap I = \emptyset$;
- s_0 is the initial state, the unique entering state, and it is a stable state,
- $R \subseteq S \times S$ represents the transitions between states,
- L is a labelling function that associates a subset of propositions to each stable state.

ILTS is used to specify any incomplete system later. The proposed verification approach is based on this formalism. Here, we present the ILTS of the motivating example showed in Fig. 3. It is derived from the net structure model: it is a LTS with some special unknown states. The white places represent the predefined (stable) states of the system. The red

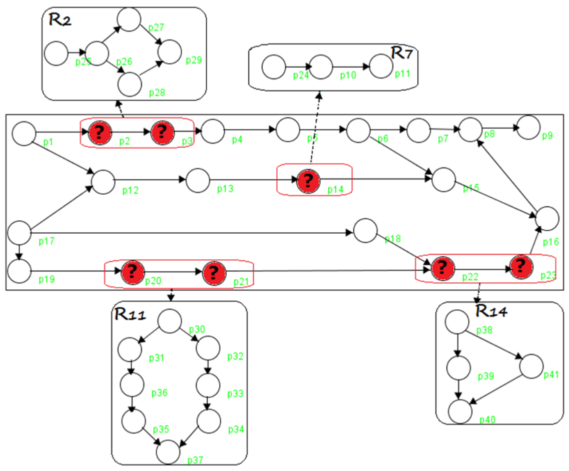


Figure 3: ILTS of the railway model.

the predefined (stable) states of the system. The red states represent the reconfigurable states of the system: its characteristics change at run-time. They are the object of new configurations to cope with the environment requirements at the current state of the system. These reconfigurations are due to an increase demand to enlarge the line capacity, the quality of service or extending the line to new parts of the area. (R2, R7, R11 and R14) are respectively new simple structures of the reconfigurable modules (2, 7, 11 and 14) at this adaptation phase. Then, once the structure is known, the constraints are applied to check these new specifications. R2 is checked against the matrix generated to satisfy the desired RCTL formula in the second module.

4.2 RCTL Model Checking

Reconfigurable CTL (RCTL) model checking is an extended version of CTL applied to adaptive systems. It has the same semantics as the standard CTL model checking for the "*True*" and "*False*" outputs with an extra definition related to the third possible output namely "*Conditionally True*". We will not recall the standard definition of CTL semantics here; we just add the new semantics related to unstable states and undefined paths. CTL is classically defined on a state of *LTS*. RCTL is defined now on states of ILTS, $M=(S, s_0, R, L)$, M, $s \models \varphi$ means that φ could hold in a state s of the ILTS M. The set of constraints that are needed to satisfy the formula φ in an unstable state s are saved in a matrix *constr*. Each element $constr(\varphi, s)$ is a set of constraints in the form $[(\varphi_1, state_1), \ldots, (\varphi_n, state_n)]$, meaning that the formula φ holds in s if the path RCTL formula φ_1 holds in $state_1$, and the path RCTL formula φ_n holds in $state_n$. We present here the semantics of RCTL:

- $M, s \models \varphi \Leftrightarrow \varphi \in L(s)$ if $s \in T$ and $s \models constr(\varphi, s)$ if $s \in I$;
- $M, s \models \neg\varphi \Leftrightarrow M, s \not\models \varphi$ if $s \in T$ and $s \not\models constr(\varphi, s)$ if $s \in I$;
- $M, s \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M, s \models \varphi_1$ and $M, s \models \varphi_2$ if $s \in T$; and $s \models constr(\varphi_1, s)$ and $s \models constr(\varphi_2, s)$ if $s \in I$;
- $M, s \models \varphi_1 \vee \varphi_2 \Leftrightarrow M, s \models \varphi_1$ or $s \models \varphi_2$ if $s \in T$; and $s \models constr(\varphi_1, s)$ or $s \models constr(\varphi_2, s)$ if $s \in I$;
- $M, s \models AX\varphi \Leftrightarrow$ ($\forall\pi$ such that $\pi_0 = s$, $M, \pi_1 \models \varphi$) for all paths starting at s, next time φ if $s \in T$ or next time $constr(\varphi, s)$ if $s \in I$;
- $M, s \models AF\varphi) \Leftrightarrow$ ($\forall\pi$ such that $\pi_0 = s$, $\exists i$ such that $M, \pi i \models \varphi$) for all paths starting at s, *eventually* φ if $s \in T$ or *eventually* $constr(\varphi, s)$ if $s \in I$;
- $M, s \models AG\varphi \Leftrightarrow$ ($\forall\pi$ such that $\pi_0 = s$, $\forall i$ $M, \pi_i \models \varphi$) for all paths starting at s, *always* φ or *always* $constr(\varphi, s)$ if $s \in I$;
- $M, s \models \varphi_1 AU \varphi_2 \Leftrightarrow$ ($\forall\pi$ such that $\pi_0 = s$, $\exists\, i$ such that ($\forall\, j < i$ ($M, \pi_j \models \varphi_1$)) $\wedge$ ($M, \pi_i \models \varphi_2$)), for all paths starting at s, φ_1*until* φ_2 if $s \in T$ or $constr(\varphi_1, s)$ until $constr(\varphi_2, s)$ if $s \in I$;
- $M, s \models EX\varphi \Leftrightarrow$ ($\exists\pi$ such that $\pi_0 = s$, $M, \pi_1 \models \varphi$) there exists a path such that *next time* φ if $s \in T$ or *next time* $constr(\varphi, s)$ if $s \in I$;
- $M, s \models EF\varphi \Leftrightarrow$ ($\exists\pi$ such that $\pi_0 = s$, $\exists\, i$ such that $M, \pi_i \models \varphi$) there exists a path such that *eventually* φ if $s \in T$ or *eventually* $constr(\varphi, s)$ if $s \in I$;
- $M, s \models E\, \varphi_1 \cup \varphi_2 \Leftrightarrow$ if there exists a path π starting from s such that $\exists s_k \in \pi \mid M, s_k \models \varphi_2$ if $s \in T$ or $s \models constr(\varphi_2, s)$ if $s \in I$ and $\forall\, s_i \in \pi$ with $i < k$, $M, s_i \models \varphi_1$ if $s \in T$ or $s \models constr(\varphi_1, s)$ if $s \in I$;
- $M, s \models EG\varphi \Leftrightarrow$ if there exists an infinite path π starting from s such that $\forall\, s_i \in \pi$, $M, s_i \models \varphi$ if $s \in T$ and $s \models constr(\varphi, s)$ if $s \in I$.

The core of the presented approach is an RCTL model checking algorithm for incomplete models, described using the ILTS formalism. It is based on the traditional explicit CTL model checking (Clarck et al., 1986) in order to deal with unstable and incomplete states. The inputs of the algorithm are an RCTL property and an ILTS model. If the ILTS is a stable LTS, it behaves as the traditional approach on predefined LTS. On the other hand, if the ILTS contains unknown states, it computes the set of path RCTL formulae that shall be guaranteed by the unspecified components later at run-time. More precisely, the algorithm operates respecting these steps. First, the RCTL formula is parsed and its parsing tree is derived. Usually, the leaves are propositions and the inner nodes are boolean and temporal operators. As CTL model checking, a bottom-up approach is applied to the tree to check if each sub-formula holds. For each node of the tree, the set of the states in which the sub-formula holds is evaluated by parsing the tree, starting from the leaves. The algorithm takes as inputs a subtree S_T of the parsing tree, the formula φ, and the ILTS M on which the original formula is evaluated. The tree S_T is a binary tree, where a node representing a unary operator has a single son, while a node representing a binary operator has two sons. We use $S_T.S$ to refer to the set of states in M that satisfy the formula represented by the current subtree, $S_T.left$ and $S_T.right$ to refer to the left and the right subtrees of the current tree (when the root is a binary operator), and $S_T.son$ to refer to the subtree of the current tree (when the root is a unary operator). The algorithm can store the elements that satisfy φ in a local set X. Moreover, the set of constraints that are needed to satisfy the formula φ in an unstable state s are saved in the matrix *constr*.

4.3 Marking Algorithms

We present here the marking algorithm of the proposed RCTL temporal logic. The inputs are: A model structure M, an RCTL formula ϕ and a subtree t. The constraint Matrix is initiated (line 2). *Mark* (ϕ, s) is a standard CTL marking function dependent on the formula ϕ. This function is applied once the visited state is a stable one (line 3). Let's assume that *Mark* (ϕ, s) ∈ {*Mark*(φ, s), *Mark*($\neg\varphi$, s), *Mark*($\varphi_1 \wedge \varphi_2$, s), *Mark*($\varphi_1 \vee \varphi_2$, s), *Mark*($AX\varphi$, s), *Mark*($AF\varphi$, s), *Mark*($AG\varphi$, s), *Mark*($\varphi_1 AU \varphi_2$, s),

```
Marking ($\phi$, $t$, $M$) {
1: for all ($s \in M.S$) {
2: constr ($\phi$, $s$) = $\emptyset$;
3:    if ($s \in M.T$) {mark ($\phi$, $s$)}
4:    elseif ($s \in M.I$) {
5:    constr($\phi$, $s$) = constr($\phi$, $s$) $\cup$ {$s$};}}}
```

Mark($EX\varphi$,s), *Mark*($EF\varphi$,s), *Mark*($E\ \varphi_1 U \varphi_2$, s), *Mark*($EG\varphi$,s)}. On the other case (line 4), a constraint is generated to be investigated at the adaptation phases. This constraint is added to the set of the existent constraints (line 5-6).

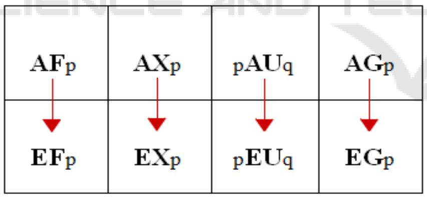
4.4 Degraded Verification Mode

Safety in critical systems is fundamental for their operation. Reconfiguration makes possible for a system to operate in different modes to be flexible as possible and adapted according the characteristics and requirements of the environment. Openness is also an inherent property, as agents may join or leave the system throughout its lifetime. The proposed verification approach is based on the generation of the constraints to be checked at each reconfiguration scenario. In case we opt to check the $AG\varphi$ formula (line 3) on the model, i.e., this property has to be satisfied by the whole system model. We generate the corresponding constraints to be respected during any adaptation. Before applying the reconfiguration tasks, the proposed algorithm makes sure that the new configuration satisfies the system requirements (line 7). Then, it is possible to check the satisfiability of the generated constraints on the new updated specification. If it is true (line 8), the system can operate safely and complete its running task. In many other cases, the properties are not respected and the system has to go forward with respect to its safety. Here, the algorithm chooses to degrade the running mode to the second level and we try to find a possible combination that should be possible to be executed by the system (line 9). Then, we move to check the validity of following formula: $EG\varphi$ (line 10) that presents the existence of a

```
 1: Verif_output R;
 2: While ( $R \neq false$) do
 3: if ($\varphi$= AGp)
 4:  {  R="Conditionally True";
 5:     constr($\varphi$, $s$);
 6:     Execute_Reconfiguration();
 7:     Verif_constr();
 8:     if (R= True) then end;
 9:     else {$\varphi$:= EGp ;
10:         Verif_constr();} }
11: if ($\varphi$= AXp) OR ($\varphi$= AFp)  OR ($\varphi$=pAUq) {
12:  if (R= "True") then end;
13:   if (R= "Conditionally True")
14:     {  constr($\varphi$, $s$);
15:       Execute_Reconfiguration();
16:       Verif_constr();
17:     if (R= True) then end;
18:       else $\varphi$:= SUBSTITUE ($\varphi$; "A"; "E"); }
19:       Verif_constr();}
20: end while
```

possible solution for the occurred deadlock state. For the following three formulas: "$AX\varphi$, $AF\varphi$, $pAUq$" (line 11). It is possible that the properties are satisfied at the stable part of the system (line 12), i.e., the reconfiguration scenario will not affect the requirement of the system. Then, the verification results should be “True”. Otherwise the corresponding constraints are checked against the updated parts of the system (line 14). In case of the non-satisfaction of the desired constraints (line 18), we can opt to the degradation. The algorithm checks if it is possible to come over the faced deadlock state. Then, we check respectively the following constraints formulas “*EXφ*, *EFφ*, *pEUq*” (line 19). The degradation strategy is presented in a summarized view in Fig. 4.

Here, the railway network is always the subject of different addition/removing of trains to various lines. As a solution for the increased demand to enlarge the system structure and the quality of service respecting to the critical safety, we can think about the existence of possibility to apply the desired property in the possible lines instead of the entire network. We opt to check the validity and existence of paths and scenarios that lead to desired target. For example, if we aim to double the speed of some trains: then, it will affect the safety distance between the components of the network. The property *p*=“double the speed”, then we check: *EFp* instead of *AFp*. Similarly, if we hope to add two extra trains in the network from certain stations to cover the large demand: *φ*= “add two extra trains”, then we check *EXφ* instead of *AXφ*. We check the formula on the predecessor state of the desired state. We look for proving the existence of safe options to improve the quality of service of the system.



$$\begin{tikzcd}
AFp \arrow[d] & AXp \arrow[d] & pAUq \arrow[d] & AGp \arrow[d] \\
EFp & EXp & pEUq & EGp
\end{tikzcd}$$

Figure 4: Degradation approach.

5 DISCUSSION

This paper highlights a double-phase approach to efficiently verify reconfigurable distributed systems, in which some components may dynamically change at run time. The idea aims to introduce an optimized formal certification method for adaptive systems: much more useful to save time and memory resources. The purpose is to optimize the verification process: the needed time and space resources at each modification of the system behavior. Based on the use of a separated modular verification approach and the results of the previous verification, we avoid the repetition of many extra unnecessary tasks during the certification of a reconfiguration scenario. To support the methodology, a new semantics of the temporal logic CTL is proposed to deal with the incomplete labelled transition systems of an adaptive system. A new marking algorithm to concretize the approach is presented. A new degraded verification algorithm is proposed as a solution for the deadlock states after applying any adaptation process. To support this built framework, correctness tests will be evaluated, we will check the validity of the results of the proposed RCTL model checking compared to the standard model checking. The gain of time and space resources will be also evaluated for huge states space of adaptive systems. Scalability of the approach will be considered in our future work. The presented case study will be checked using the introduced framework.

6 CONCLUSIONS

This paper focuses on the importance of online formal verification of reconfigurable systems. It introduces a new approach to efficiently certify adaptation scenarios at run-time process. It can avoid repetitive useless tasks that slow down the certification of adaptation scenario. Sometimes, the verification is unfeasible because of the time and resources limitation at run-time. A verification approach is proposed to cover the limits of traditional model checking method coping with large reconfigurable systems. An RCTL profile is introduced to present the semantics of properties in reconfigurable systems. An overview of the needed algorithm is also presented. Marking algorithm and degraded verification strategy are proposed here. This paper states the preliminary steps to address the runtime model checking of adaptive distributed systems. This work could be extended in many directions. At the moment, we are working on the implementation of the algorithm and to explore a new symbolic approach. In this paper, we presented RCTL, but the future work is to support the full CTL logics by adding other extensions.

REFERENCES

Axelsson, R., et al., 2010. “Extended computation tree logic,” *in Logic for Programming, Artificial Intelligence, and Reasoning, Berlin Heidelberg: Springer, pp. 67-81.*

Baier, C. and Katoen, J. P., 2008. “Principles of model checking”. Vol. 26202649. *MIT press Cambridge*.

Bortolussi. L., et al., 2015. “Verification of Complex Adaptive Systems”. [Online]. Available: http://homepage.lnu.se/staff/daweaa/papers/2015CASVerification.pdf.

Clarck, E. A., Emerson, A. P., and Sistla, A. P., 1986. “Automatic Verification of Finite-State Systems Using Temporal Logic Specification: A Practical Approach,” *ACM Transaction on Programming Languages and Systems, vol. 8, no. 2, pp. 244-263.*

Goldsby, H. J. et al., 2008. “AMOEBA-RT: Run-Time Verification of Adaptive Software”. In: *Giese H. (eds) Models in Software Engineering. MODELS 2007. Lecture Notes in Computer Science, vol 5002. Springer, Berlin, Heidelberg.*

Kalita, D., and Khargonekar, P. P., 2002. “Formal verification for analysis and design of logic controllers for reconfigurable machining systems,” *IEEE Trans. Robot. Autom., vol. 18, no. 4,* pp. 463–474.

Li, Z. W., and Zhou, M. C., 2009. “Deadlock Resolution in *Automated Manufacturing Systems: A Novel Petri Net Approach,” London, U.K.: Springer,* pp. 20-28.

Salem, M. O. B., Mosbahi, O., Khalgui, M., and Frey, G., 2015. “ZiZo: Modeling, simulation and verification of reconfigurable real-time control tasks sharing adaptive resources: Application to the medical project BROS”. *Proceedings of the Int. Conf. on Health Informatics, Portugal,* pp. 20-31.

Schaefer. I., 2008. “Integrating Formal Verification into the Model-Based Development of Adaptive Embedded Systems”. PhD thesis, *TU Kaiserslautern, Kaiserslautern, Germany*, ISBN 978-3-89963-862-2.

Sharifloo, A.M., and Spoletini, P., 2013. “LOVER: Light-weight fOrmal Verification of adaptivE systems at Run time” Formal Aspects of Component Software, pp 170-177.

Schneider, K., et al., 2006. “Verifying the adaptation behavior of embedded systems”. *SEAMS ’06,* pp. 16–22.

Starke, P. H., and Roch, S., 2002. “Analysing signal-net systems”. Professoren des Inst. für Informatik.

The Metro: a Parisian institution [online]. [Accessed 20 February 2017]. Available from: http://www.ratp.fr/en/ratp/r_108503/the-metro-a-parisian-institution/.

Trapp, M. et al., 2007. “Runtime adaptation in safety-critical automotive systems*”. In Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering, pages 308–315, Anaheim, CA, USA, ACTA Press.*

Wang, W., et al., 2007. “Reachability analysis of cost-reward timed automata for energy efficiency scheduling”. *In Proceedings of Programming Models and Applications on Multicores and Manycores, PMAM’14, pages 140:140– 140:148, New York, NY, USA, ACM.*

Xie, G., and Dang. Z., 2004. “Ctl model-checking for systems with unspecified finite state components”. SAVCBS.

Zhang, J., and Cheng, B. H. C., 2006. “Using temporal logic to specify adaptive program semantics*”. Journal of Systems and Software, vol. 79, no. 10, pp. 1361 – 1369.*

Zhang, J., Khalgui, M., Li, Z. W., Mosbahi, O. and Al-Ahmari, A. M., 2013. “R-TNCES: A novel formalism for reconfigurable discrete event control systems”. *IEEE Trans. Systems, Man, and Cybernetics: Systems, vol. 43, no. 4, pp. 757-772.*

Zhang, J., et al., 2006. “Model-based development of dynamically adaptive software*”. ICSE ’06, pp. 371–380, New York, NY, USA.*