A Novel Approach for Repairing Reconfigurable Hierarchical Timed Automata

Roufaida Bettira[1], Laid Kahloul[2] and Mohamed Khalgui[1,3]
[1]*National Institute of Applied Sciences and Technology (INSAT), University of Carthage, Tunis 1080, Tunisia*
[2]*LINFI Laboratory, Computer Science Department, University of Mohamed Khider, Biskra, Algeria*
[3]*School of Electrical and Information Engineering, Jinan University (Zhuhai Campus), Zhuhai 519070, China*

Keywords: Discrete Event Control System, Timed Automata, Modeling and Verification, Repair, Mutation, Reconfiguration.

Abstract: Timed Automata (TA) is a formalism for formal modeling and verification of systems with temporal requirements. Reconfigurable hierarchical timed automata (RHTA) extend TA to cover reconfigurability and hierarchy of large reconfigurable discrete event control systems (RDECS). After formal modeling of an RDECS with RHTA, formal verification against functional properties is done using model-checker. In the case of non-satisfaction of a property, the model-checker generates a counterexample. Mostly, non-satisfaction of a functional property is owing to incorrect clock constraints (guards and invariants). In this paper, we propose an approach based on mutation testing for repairing the faulty RHTA model so that the concerned functional property be satisfied. First, the hierarchy structure of each configuration is tested and repaired. Then, the generated counterexample is used to repair the wrong guards specified in TA models which are constructing the RHTA model. Experimentation shows that the proposed approach is able to repair a considerable part of the RHTA model designed initially.

1 INTRODUCTION

Hierarchical RDECSs have hierarchical changeable structures and behaviors over time. Reconfiguration is required either to respond to requirements by users or to handle unexpected hardware malfunctions. Hierarchical modeling is useful since it is applicable to real-life embedded systems, it specifies large systems in different levels and it allows the reuse of components. To introduce reconfigurability and hierarchy to TA models, the extension reconfigurable hierarchical timed automata (RHTA) is proposed in (Bettira et al., 2019). RHTA is a structure of hierarchical behavior graphs where each graph is constituted of basic components (TA models) and it represents a configuration. A behavior graph is reconfigured to another one by applying reconfiguration functions.

Verification of required properties is done using a model-checker which receives two inputs, the system model and the set of properties to be checked specified in a temporal logic such as computational tree logic (CTL) (Boucheneb et al., 2009). Formal verification is the most reliable validation activity for RDECSs. However, it does not include a repairing process that fixes the initial design in the case of non-validation.

There are several works on testing timed automata (and its extensions) (Springintveld et al., 2001), (Hessel et al., 2008), and (Luthmann et al., 2019) however, the reparation of these models is not widely handled. In (Nielsen and Skou, 2003), the test generation from timed automata is automated. In (Aichernig et al., 2013), model-based mutation testing is introduced. Authors in (Aichernig et al., 2014) combine model-based mutation, classical mutation testing, and model-based debugging to propose a methodology for mutation-based debugging of real-time systems. In (Aichernig et al., 2015), several optimizations of a symbolic conformance checker are proposed using constraint solving techniques to solve the state space explosion problem. The work (Arcaini et al., 2019) on software product lines proposes an evolutionary approach to obtain a new feature model that captures the given requested changes from existing products. Repairing Timed automata is explicitly handled in (André et al., 2019). The authors propose a repairing process using abstraction and testing. The process includes six steps for repairing TA clock guards. One

Bettira, R., Kahloul, L. and Khalgui, M.
A Novel Approach for Repairing Reconfigurable Hierarchical Timed Automata.
DOI: 10.5220/0009408503980406
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 398-406
ISBN: 978-989-758-421-3

of these steps is the generation of constraints which is an expensive phase.

To the best of our knowledge, no previous work has dealt with RHTA repairing. In this paper, we propose a novel approach for repairing the initial RHTA designed for a hierarchical RDECS. First, the hierarchy structure of each configuration is tested and repaired. Then, basic components of each configuration which are verified against a set of functional properties are tested and repaired (i.e., correction of time constraints). Finally, the reconfiguration automaton of the whole RHTA is repaired. The approach is based on mutation testing and the generated counterexample from the RHTA verification stage. The approach is able to partially repair incorrect models in a reasonable time considering the large size of RHTA models.

The remainder of this paper is organized as follows. Section 2 presents the background concerning the extension RHTA. Section 3 provides an approach for the reparation of RHTA models. Section 4 applies and evaluates the proposed contribution. Finally, Section 5 concludes this paper.

2 BACKGROUND

Before we introduce our approach, some basic definitions are briefly recalled in this section.

2.1 Timed Automata

A timed automaton as reported in (Bengtsson and Yi, 2003), (Bouyer et al., 2008) is formally defined as A= (L, l_0, C, Σ, E, Inv) where (i) L is a set of locations, (ii) $l_0 \in L$ is the initial location, (iii) C is a set of clocks, (iv) Σ is a set of input, output, and internal (denoted τ) actions, (v) $E \subseteq L \times \Sigma \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard, and a set of clocks to be reset, and (vi) $Inv : L \rightarrow B(C)$ assigns invariants to locations.

The semantics of TA (Bengtsson and Yi, 2003), (Bouyer et al., 2008) is given through a transition system (denoted by *TS*, it is also named timed transition system or labeled transition system) where a state is a pair $\langle l, u \rangle$, l is the current location and u is a function providing the current values of clocks ($u : C \rightarrow R_{\geq 0}$). There are two types of transitions between states. The automaton may either delay for some time (i.e., a delay transition), or follows an enabled edge (i.e., an action transition).

2.2 Reconfigurable Hierarchical Timed Automata

2.2.1 Formalization

RHTA (Bettira et al., 2019) are an extension of timed automata for hierarchical RDECSs modeling and verification. An RHTA is formally defined as RA= (BC, BG, R), where (i) BC is a finite set of basic components, (ii) $BG = \{g_0, g_1, ..., g_{n-1}\}$ is a set of n hierarchical behavior graphs representing the different behaviors (configurations) performed by the modeled system, and (iii) R is a finite set of m reconfiguration functions $R = \{r_1, r_2, ..., r_m\}$. A reconfiguration function r is a structural modification that transforms one configuration to another configuration after the fulfillment of certain preconditions.

A basic component in set BC is represented as a timed automaton. A hierarchical behavior graph is a tuple g= (V, V_0, T, Lab, $Prob$, F) where,

- V: is a finite set of vertices (nodes) such that, $v \in V$ is represented by one or several basic components (i..e, one TA model or a TA network).
- $V_0 \subset V$: is a set of initial vertices.
- $T : V \rightarrow V$ is the decomposition transition relation between two vertices from two successive levels (i.e., activation of components).
- Lab: $2^V \rightarrow \{AND, OR\}$ is a labelling function mapping a subset of vertices outgoing from the same vertex to one of the labels "AND" or "OR".
- $Prob$: $T \rightarrow]0, 1]$ is a probabilistic transition function that maps each transition to a real number in interval $]0, 1]$, where 1 is the default probabilistic value for a transition.
- $F \subset V$: is a finite set of end vertices in the last level of the hierarchy.

A behavior graph represents one configuration of the hierarchical system (as seen in Figure 1):
Formally, a reconfiguration function r is a couple ($Cond$, m) where,

- $Cond \in \{true, false\}$ is the precondition of r.
- $m : (g, v) \rightarrow (g', v')$ is the structure modification instruction that transforms one graph (i.e., configuration) g to another g', where $g, g' \in BG$.

The set of fundamental structure modification instructions of RHTA is detailed in Table 1 such that $v_1, v_2 \in V$ are two vertices, $A \in BC$ is a basic component, $x \in]0, 1]$ is a real number, and $X \subseteq V$ is a subset of vertices. The concatenation between basic modification instructions is denoted by $+$ to build complex modification instructions.

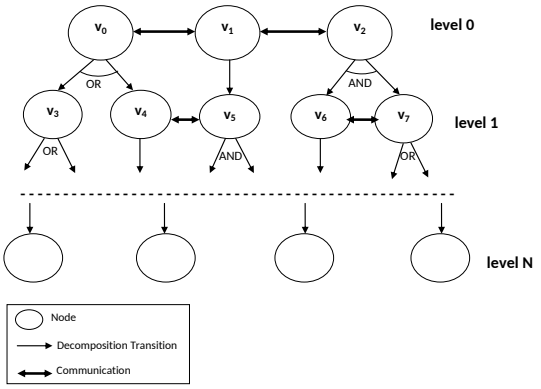

Figure 1: A configuration of RHTA.

Table 1: Fundamental structure modification instructions of RHTA.

Instruction	Symbol
Add A to vertex v_1	new(A, v_1)
Remove A from vertex v_1	rmv(A, v_1)
Add vertex v_1	new(v_1)
Remove vertex v_1	rmv(v_1)
Insert a transition from v_1 to v_2 with the probability value x	new(v_1, x, v_2)
Remove transition from v_1 to v_2	rmv(v_1, v_2)
Modify the probability value of the transition from v_1 to v_2 by x	mdf(v_1, x, v_2)
map the label *AND* to X	$AND(X)$
map the label *OR* to X	$OR(X)$

2.2.2 Verification of RHTA

Semantics of hierarchical behavior graphs *BG* of RHTA (Bettira et al., 2019), (Roufaida et al., 2019) is given through hierarchical transition systems (HTS). An HTS is defined by HTS= (*S*, *Tr*) where *S* is a set of states representing the different *TS*s of automata in each $v \in V$ and $Tr \subseteq (S \times S)$ is the decomposition transition function between those states such that $Tr \subseteq T$. The verification idea is about considering similarities between different configurations, redundant parts are eliminated in each generated configuration. We show in Figure 2 different steps followed for the verification of RA= (*BC*, *BG*, *R*) as reported in (Bettira et al., 2019). Basic components are verified first, then initial configuration, after that verification of other generated configurations, and finally the entire system. TA model-checker is applied in the different stages of the process to verify the satisfaction of a set of properties (i) in each component of each configuration and (ii) between configurations (reconfiguration functions). In the case of incorrectness, a counterexample is generated and the process exits. In the next section, we show how counterexamples can be used to repair the initial RHTA model.

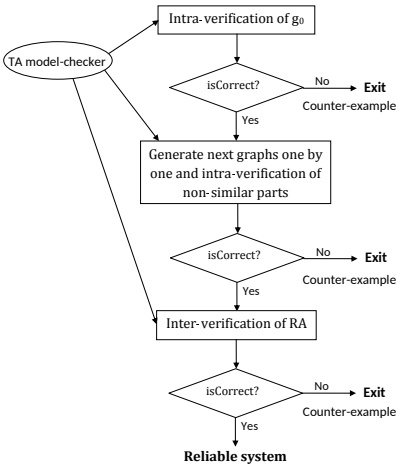

Figure 2: Verification process of RHTA.

3 AN APPROACH FOR REPAIRING RHTA MODELS

In this section, we propose a novel repairing approach of RHTA as depicted in Figure 3. The approach takes the system under validation (SUV) and its model designed with RHTA as inputs to obtain a repaired RHTA model. SUV is considered as a black box that can be queried for acceptance of RHTA structure (i.e., how the system components are organized) and also for acceptance of reconfiguration functions. Besides, the approach exploits the counterexample generated by TA model-checker with the mutation testing for the reparation of the basic components themselves (i.e., TA models). Indeed, we can not handle all kinds of faults for one TA model in an economic correction time. We focus on repairing guards, invariants, and resets which have been incorrectly defined because they are mostly responsible for non-satisfaction of a functional property.

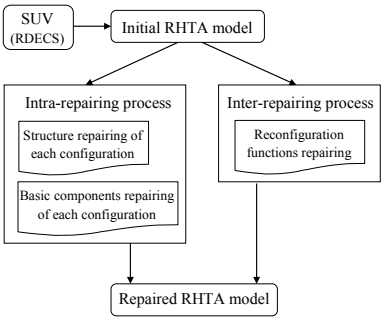

Figure 3: RHTA general repair process.

3.1 RHTA Structure Repairing

Definition 1. In one behavior graph (configuration), each node (component) with its sub-nodes of the next level is defined as a block. Let $v_s \in V$ denotes the source node and $V_{targ} \subset V$ denote the set of its sub-nodes for one block.

Definition 2. A block mutant represents one block with a simple modification. Under the assumption that each component is on its correct level and only activation connections and labels nature can be wrong, we propose three mutation operations effectuated over a bloc to implement three kinds of structural faults that can be done by the modeler.

- **Change Source:** v_s is replaced by another node on the same level. This modification is for detecting whether v_s is the correct activator component of its sub-components.
- **Change Target:** each target node in V_{targ} is replaced by another node on the same level. This modification is for detecting whether V_{targ} is the correct set of sub-components that v_s should activate.
- **Change Label:** change the label $Lab(V_{targ})$. This modification is for detecting whether the modeler sets the correct activation mode (i.e., "AND" or "OR") for the set of sub-components V_{targ}.

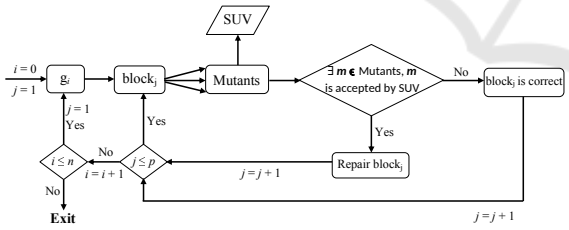

Figure 4: Structure repair process for an RHTA.

Figure 4 represents the structure repairing process of an RHTA. First, each configuration (behavior graph) in $BG = \{g_0, ..., g_n\}$ is divided into p blocks. Then, a set of mutants is generated for each block. After that, SUV is queried for the acceptance of the mutants. If all generated mutants for one block are not accepted, the original block is correct. Else, the block is repaired based on accepted mutants.

3.2 RHTA Basic Components Repairing

Verification of an RHTA against a set of functional properties consists of applying TA model-checker to each *TS(v)* of the initial configuration g_0, and to non-similar parts (i.e., new TSs) of the other configurations generated from g_0. Non-satisfaction of functional properties such as deadlock-freeness, safety, ...etc. is due to wrong guards, invariants, and resets of clocks specified by the designer during the modeling stage. In this case, the counterexample is generated as one path of couples under the form (l_x, u_x) from *TS(v)*. l_x is a set of locations of TA network which is represented in the node v and u_x is the clocks valuation. In our approach, we use this counterexample to detect and repair wrong guards, resets, and invariants.

Definition 3. A TA mutant represents an intentionally modified version of the original designed TA which implements a common modeling error. We propose five mutation operations that affect guards, resets, and invariants.

- **Change Guard:** each equality/inequality sign appearing in the guard is replaced by another one. This implements incorrect enabling conditions in transitions.
- **Negate Guard:** each guard is replaced by its negation. This covers faults because of a modeler forgot to negate an enabling condition.
- **Change Invariant:** each equality/inequality sign appearing in the invariant is replaced by another one. This implements incorrect stay conditions in locations.
- **Negate Invariant:** each invariant is replaced by its negation. This covers faults because of a modeler forgot to negate a stay condition.
- **Change Reset Clock:** each clock appearing in a reset is replaced by another one. This implements the incorrect choice of clocks to be reset.

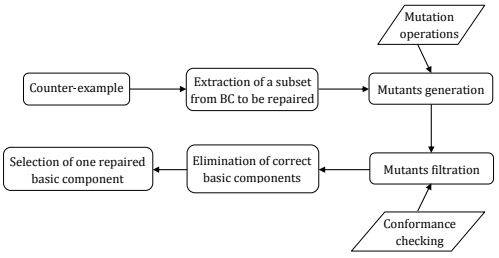

Figure 5: Basic components repairing process.

From generated counterexample, figure5 depicts the different steps followed for detecting and repairing wrong guards, resets, and invariants in TA models (basic components).

- Step 1: let $TA_{test} \subseteq BC$ denote the set of TA which can implement wrong guards, resets, and invariants. We identify TA_{test} through, $\forall A = (L, l_0, C,$

$\Sigma, E, Inv) \in BC, \forall(l_x, u_x)$, if $(C_x \cap C \neq \emptyset)$ then add A to TA_{test}. C_x is the set of clocks appeared in u_x.

- Step 2: a set of mutants is generated for each $A \in TA_{test}$ using relevant mutation operations.
- Step 3 and 4: to detect whether the faults in the mutated models have been really implemented by the modeler, a conformance check between the original and the mutated TA models is required. In this step, timed input/output conformance relation tioco (Schmaltz and Tretmans, 2008), (Krichen and Tripakis, 2009) is used to analyze the set of mutants. If all mutants of one $A \in BC$ are not conformed, A is eliminated of TA_{test} (i.e., A has been correctly designed).
- Step 5: finally, one mutant from the remaining mutants is selected for each $A \in TA_{test}$ to be its repaired model A_{rep}. The selected mutant A_{rep} is the closest to A because we suppose that the designer always makes simple modeling errors.

3.3 Inter-repairing of RHTA

Once each configuration is repaired structurally and behaviorally, the whole reconfiguration scenario should be also repaired if inter-verification of RA is incorrect (see Figure 2). This phase is necessary since the system during reconfiguration may behave incorrectly despite the reparation of each configuration separately.

In the inter-verification of RHTA (Bettira et al., 2019), *RA* is considered as an automaton where states are the different configurations and transitions are the reconfiguration functions. In the case of incorrectness, the counterexample generated by the model-checker is a chain of configurations that causes the problem. First, this chain is exploited to identify the set of reconfiguration functions that can be wrong. After that, SUV is queried again for the acceptance of these reconfiguration functions. Finally, the modeler rechecks and repairs not accepted reconfiguration functions.

4 EXPERIMENTATION

In this section, we apply the different parts of the proposed RHTA repair approach on the "train gate system" as a real system to show and evaluate the reparation results in term of number of repaired faults and its correction time.

4.1 Illustrative Example (Train Gate System)

We consider *EXP*= (BC_{exp}, BG_{exp}, R_{exp}) as a simple system designed with RHTA such that:

- BC_{exp}= $\{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$ is the set of TA models representing the basic components of this system.
- BG_{exp}= $\{g_0, g_1, g_2\}$ is the set of behavior graphs representing the possible configurations as presented in Figure 6.
- R_{exp}= $\{r_1, r_2, r_3, r_4\}$ are four reconfiguration functions that reconfigure the system from g_0 to g_1, from g_1 to g_2, from g_2 to g_1, and from g_2 to g_0, respectively. These reconfiguration functions are described in Table 2.

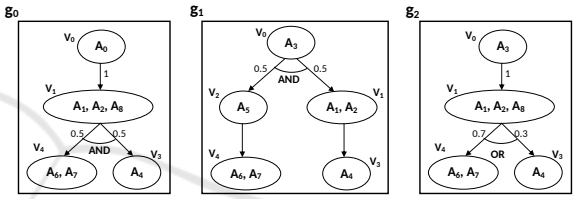

Figure 6: System configurations.

Table 2: Reconfiguration functions of *EXP*.

reconfig. functions	*Cond*	*m*
r_1	Malfunction 1	new(v_2)+ new(v_0, 0.5, v_2)+ mdf(v_0, 0.5, v_1)+ new(v_2, v_4)+ rmv(v_1, v_4)+ mdf(v_1, 1, v_3)+ rmv(A_8, v_1)+ AND(v_1, v_2)+ rmv(A_0, v_0)+ new(A_3, v_0)
r_2	User requirement 1: deactivate the component v_2	rmv(v_2)+ rmv(v_0, v_2)+ rmv(v_2, v_4)+ mdf(v_0, 1, v_1)+ new(v_1, 0.7, v_4)+ mdf(v_1, 0.3, v_3)+ new(A_8, v_1)+ OR(v_3, v_4)
r_3	User requirement 2: reactivate the component v_2	new(v_2)+ new(v_0, 0.5, v_2)+ mdf(v_0, 0.5, v_1)+ new(v_2, v_4)+ rmv(v_1, v_4)+ mdf(v_1, 1, v_3)+ rmv(A_8, v_1)+ AND(v_1, v_2)
r_4	Malfunction 2	rmv(A_3, v_0)+ new(A_0, v_0) + AND(v_3, v_4)+ mdf(v_1, 0.5, v_3)+ mdf(v_1, 0.5, v_4)

4.1.1 Structure Repair of EXP

We choose the configuration g_1 of the modeled system *EXP* to apply the structure repair process, the structure of other configurations is repaired in the same way. First, g_1 is divided into three blocks. Second, a set of mutants is generated according to feasible mutation operations for each block. Third, SUV

is queried for acceptance of each mutant to decide finally about correctness or reparation of the concerned block. We illustrate in Figure 7 each block with its generated mutants. One block is repaired through applying changes of accepted mutants. Repaired blocks are recomposed to form the repaired structure of the configuration g_1.

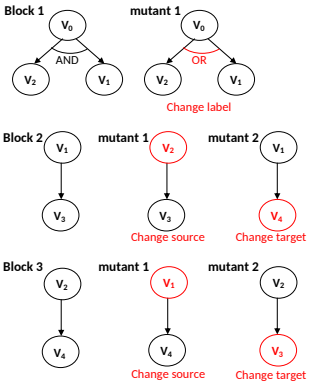

Figure 7: Mutants of the blocks.

4.1.2 Basic Components Repair of EXP

As the purpose is the reparation of basic components, we only focus on one node (vertex) of one configuration to show the application of this repair process. Let us consider v_1 of the initial configuration g_0 as the node which represents the train control (i.e., controls access to a bridge for several trains). We show in Figure 8, Figure 9, and Figure 10 TA models implementing v_1. The basic components constituting TA network represented in v_1 are $T1$, $T2$, $T3$, B, and Q such that $T1$, $T2$, and $T3$ are three TA models instantiated from the train model, B is one TA model instantiated from the bridge model, and Q is one TA model instantiated from the queue model.
We suppose that $\phi = AFG\ not\ deadlock$ is the required deadlock-free property to be checked (i.e., ϕ is specified in CTL). The model-checker used in this experiment is UPPAAL (Bengtsson and Yi, 2003). UPPAAL takes the TA network $TS(v_1)$ and the property ϕ as inputs for verification. We show in Figure 11 and Figure 12 verification results and generated counterexample, respectively.

The counterexample shows that the system can not proceed from any train (i.e., all trains are blocked). The system is blocked in the state (l_x, u_x) = ({start, occupied, safe, safe, danger}, {*time*= 186.624113, *time* = 980.165337, *time* = 395.092529}). The clock "*time*" is only appeared in the train model, thereby it is the model that can be incorrectly designed. We have $TA_{test} = (T1, T2, T3))$, it is sufficient to apply the repair process on one train model because they have the

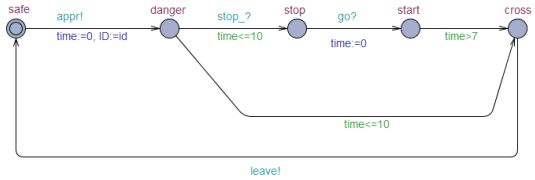

Figure 8: The train.

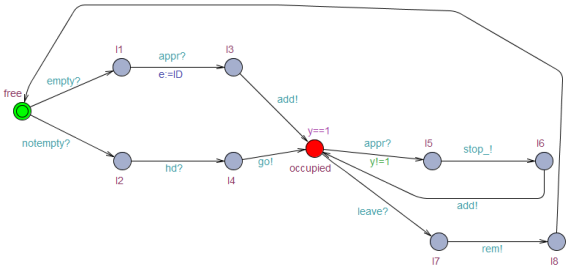

Figure 9: The bridge.

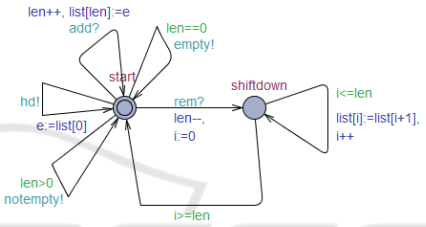

Figure 10: The queue.

same design. Mutants that can be generated for the train model are obtained by:

- negating the guards which gives a mutant m_1 with $time > 10$ and a mutant m_2 with $time \leq 7$,
- changing guards which gives four mutants m_3, m_4, m_5, and m_6 with $time \geq 10$, $time < 10$, $time \geq 7$, and $time < 7$, respectively.
- There are no invariants in this model, relevant mutation operations can not be applied.
- There are no clocks other than *time*, the change reset clock operation can not be applied.

Conformance checking is then applied to the six generated mutants. Results show that m_1 and m_3 are conformed mutants. Finally, the process selects m_1 to be the repaired train model because we suppose that the designer only forgot to negate the guard on the clock *time*. Repaired train model is shown in Figure 13.
To show the performance of our repair approach, UPPAAL model-checker is re-applied to $TS(v_1)$. Figure 14 shows that the property ϕ is satisfied after repairing each of $T1$, $T2$, and $T3$.

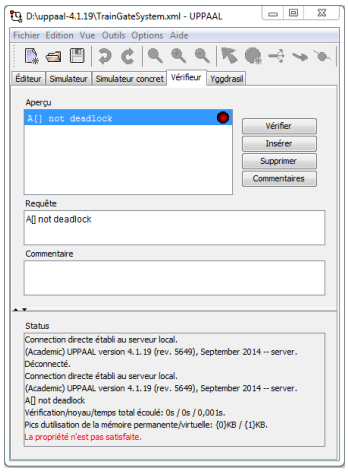

Figure 11: Verification result.

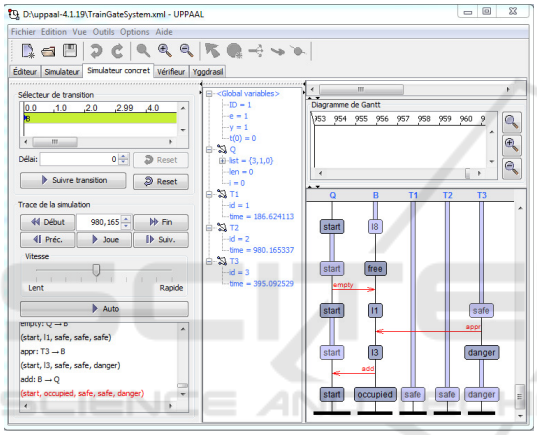

Figure 12: Generated counterexample.

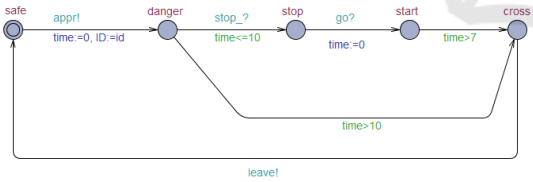

Figure 13: Repaired train model.

4.1.3 Inter-repair of EXP

Considering the system *EXP*, the reconfiguration automaton of *EXP* is represented in Figure 15. We suppose that (g_0, g_1, g_2, g_0) is the counterexample generated by the model-checker after *EXP* inter-verification, thereby the reconfiguration functions that can be wrong are r_1, r_2, and r_4. The modeler re-specifies not accepted reconfiguration functions by SUV based on structure faults that have been detected and repaired in the structure repairing phase.

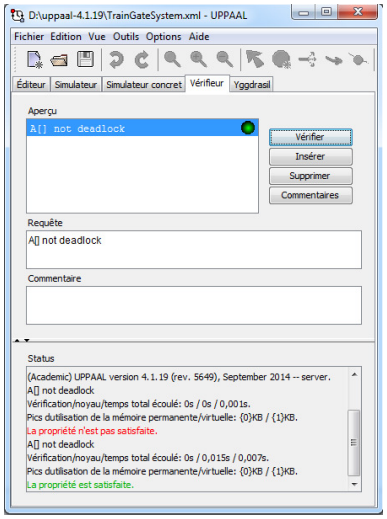

Figure 14: Verification result after reparation.

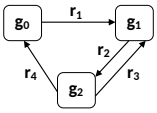

Figure 15: Reconfiguration automaton of *EXP*.

4.2 Evaluation of Performance

4.2.1 Evaluation in Term of Number of Repaired Faults

The number of repaired faults is dependant on the number of incorrect clocks constraints and the communication between these clocks in a TA network. The generated counterexample is able to detect and cover wrong guards, resets, and invariants in TA models as much as they are connected. The curve in Figure 16 shows that the approach is able to repair a considerable number of detected incorrect clock constraints. On the other hand, the number of repaired structural faults is dependant on the size of RHTA. Indeed, the number of structural faults is developed slowly comparing to RHTA size because the designer makes fewer faults during structure modeling. Figure 17 shows that the number of repaired structural faults is correspondent to this development.

4.2.2 Evaluation in Term of Reparation Time

The proposed approach has a reasonable complexity compared to the large size of RHTA models. We show in Table 3 the required time for each repairing phase where: 1) $2O(3p)$ is the complexity of applying the three mutation operations over p blocks and applying acceptance testing on all generated mutants. 2)

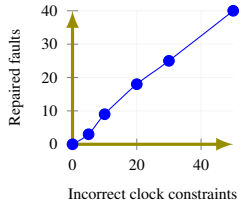

Figure 16: Evaluation of the number of repaired clock constraints.

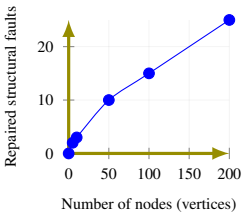

Figure 17: Evaluation of the number of repaired structural faults.

$O(|BC|)$ is the complexity of extracting incorrect basic components, $2O(5|TA_{test}|)$ is the complexity of applying and testing all mutants generated after applying the five mutation operations on each TA of TA_{test}, and $O(accpt\_mt)$ is the complexity of selecting a repaired TA from accepted mutants ($accpt\_mt$ is the number of accepted mutants). 3) $2O(r)$ is the complexity of testing and repairing r reconfiguration functions which are appeared in the generated counterexample.

Table 3: Required correction time for each phase.

Phase	Structure repair	clock constraints repair	inter-repair
Complexity	$2O(3p)$	$O(\|BC\|)$ + $2O(5\|TA_{test}\|)$ + $O(accpt\_mt)$	$2O(r)$

5 CONCLUSION

This paper proposes an approach of three phases for repairing RHTA models, (i) structure repair, (ii) basic components repair, and (iii) inter-repair. The approach is based on mutation testing and the generated counterexample from the RHTA verification stage. One counterexample is not able to detect all faults in one configuration however, it covers many parts that can be wrong. The paper also provides an application example on a simple reconfigurable hierarchical system designed with RHTA for illustrating both of structure repair process and inter-repairing of an RHTA. For applying and evaluating the basic components repair process, the train gate system is used as a real timed system. The process is able to detect and repair incorrect guards in basic components (TA models) which makes the model satisfying the required property.

In future work, we plan to add improvements to the proposed approach in order to control the cost of time and memory space required for repairing large RHTA such as RHTA modeling smart grids as hierarchical reconfigurable systems.

REFERENCES

Aichernig, B. K., Hörmaier, K., and Lorber, F. (2014). Debugging with timed automata mutations. In *International Conference on Computer Safety, Reliability, and Security*, pages 49–64. Springer.

Aichernig, B. K., Jöbstl, E., and Tiran, S. (2015). Model-based mutation testing via symbolic refinement checking. *Science of Computer Programming*, 97:383–404.

Aichernig, B. K., Lorber, F., and Ničković, D. (2013). Time for mutants—model-based mutation testing with timed automata. In *International Conference on Tests and Proofs*, pages 20–38. Springer.

André, É., Arcaini, P., Gargantini, A., and Radavelli, M. (2019). Repairing timed automata clock guards through abstraction and testing. In *International Conference on Tests and Proofs*, pages 129–146. Springer.

Arcaini, P., Gargantini, A., and Radavelli, M. (2019). Achieving change requirements of feature models by an evolutionary approach. *Journal of Systems and Software*, 150:64–76.

Bengtsson, J. and Yi, W. (2003). Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets*, pages 87–124. Springer.

Bettira, R., Kahloul, L., Khalgui, M., and Li, Z. (2019). Reconfigurable hierarchical timed automata: Modeling and stochastic verification. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2364–2371. IEEE.

Boucheneb, H., Gardey, G., and Roux, O. H. (2009). Tctl model checking of time petri nets. *Journal of Logic and Computation*, 19(6):1509–1540.

Bouyer, P. et al. (2008). Model checking timed automata.

Hessel, A., Larsen, K. G., Mikucionis, M., Nielsen, B., Pettersson, P., and Skou, A. (2008). Testing real-time systems using uppaal. In *Formal methods and testing*, pages 77–117. Springer.

Krichen, M. and Tripakis, S. (2009). Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304.

Luthmann, L., Gerecht, T., Stephan, A., Bürdek, J., and Lochau, M. (2019). Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints. *Journal of Systems and Software*, 149:535–553.

Nielsen, B. and Skou, A. (2003). Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5(1):59–77.

Roufaida, B., Laid, K., and Mohamed, K. (2019). Parallel verification methodology of reconfigurable hierarchical timed automata. In *The 2019 European Simulation and modeling Conference*. eti.

Schmaltz, J. and Tretmans, J. (2008). On conformance testing for timed systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 250–264. Springer.

Springintveld, J., Vaandrager, F., and D'Argenio, P. R. (2001). Testing timed automata. *Theoretical computer science*, 254(1-2):225–257.