Robustness of Composed Timed Systems

Hacène Fouchal[1], Antoine Rollet[2], and Abbas Tarhini[1⋆]

[1] GRIMAAG,
Université des Antilles et de Guyane,
F-97157 Pointe-à-Pitre, Guadeloupe, France
Hacene.Fouchal@univ-ag.fr
[2] CReSTIC/LICA,
Université de Reims Champagne-Ardenne,
BP 1039 F-51687 Reims Cedex, France
{Antoine.Rollet, Abbas.Tarhini}@univ-reims.fr

Abstract. In this study we present a technique for testing robustness of Real-time systems described as Component-Based System having timing constraints. Each component is modeled as a Timed Input-Output Automaton (TIOA). For robustness issue, we handle two specifications : a nominal one (the more detailed specification) and a degraded one (considering only vitale functionnalities). We derive test sequences from the nominal specification of each component. We proceed to a mutation technique on these sequences in order to simulate hostile environments. Then we present a detailled algorithm for the application of test sequences on the Implementation of the system. This is done by means of an adequate test architecture consisting of the Implementation Under Test (IUT) of components, and a distributed tester that consists of a set of coordinating testers. Each tester is dedicated to test a single component.

Key-words: Real-Time System, Timed Automata, Component based System, Testing, Robustness.

1 Introduction

The complexity of systems becomes higher and higher. One reason of this complexity increase is the fact that systems consist of many independent distributed components running concurrently on heterogeneous networks. On other hand, traditional testing methods are not able to take care of all features of such systems like time constraints, distribution, integration of a component in a complex system. Such integration may lead to architectural mismatches when assembling components with incorrect behavior [1], leaving the system in a hostile environment. The criticality of such systems requires the creation of software components that can function correctly even when faced with improper usage or stressful environmental conditions. The degree of tolerance to such situations is referred to as a component's robustness.

⋆ also at Lebanese American University, Beirut, Lebanon

M. Bieliková et al. (Eds.): SOFSEM 2005, LNCS 3381, pp. 155–164, 2005.

In this paper, we will show how to model a Real-Time Component-Based Systems (RTCBS). Then how to test robustness on such systems. The mutation method inserts hazards to test sequences generated from the component's nominal (original) specification. The mutated test sequences are used to test the system's robustness according to a corresponding degraded specification which describes the minimal authorized behavior in case of unexpected situations.

The execution of timed-mutated test sequences on the IUT is divided into two phases. In the first phase, the tester executes a set of test sequences to test the robustness of each component in isolation. In the second phase, the tester also executes and monitor timed test sequences, but to test the robustness of interaction between components integrated in a RTCBS and thus testing the system's robustness.

In section 2 we give a review on related work done on testing distributed and component based systems. Section 3 provides a background of different modeling techniques used to represent CBS. In Section 4 we focus on our method for testing Real-Time Component Based Systems. In Section 5, we detail the testing architecture as well as the test execution process. Finally, we conclude in Section 6.

2 Related Work

As far as we know, robustness testing of RTCBS is not widely developed, but there some interesting works done on distributed testing of real-time systems [2]; however, only few of them, [3], [4] dealt with testing component based systems, and others [5], [6], [7] studied the construction of component based systems.

Distributed testing was studied with different test architectures. [2] proposed a distributed tester, with the distribution of global test sequence (GTS) into local test sequences executed over local testers, where the synchronization and the fault detectability problems are discussed.

Robustness testing has been studied in many areas. It is based on insertion of hazards into test sequences. In the following, we give an overview of selected works on robustness testing and fault injection methods.

A fault injection approach called Software Implemented Fault Injection (SWIFI) has been developped in many studies. Among, theses studies the FIAT system ([8]) modifies the binary image of a process in memory (fault injection at compile time). FTAPE ([9]) combines system-wide fault injection with a controllable workload in order to create high stress conditions and fault propagation for the machine.

[10] proposed an object oriented approach to test software robustness-based on parameter data types rather than component functionality. It is a black box approach, ie the tester uses only the description of the component interface in terms of parameters and data types. They consider that most robustness failures of software are caused by code that simply neglects to test for invalid inputs.

Castanet and all present in [11], a study on formal robustness testing. It mainly consider all possible faults that a system can execute. The designer has to insert them into the specification and finally derives test sequences from

the modified specification. In the following section we discus how a real-time component-based system is best modelled.

3 Formal Description of Systems

Software architecture (SA) dynamics that represent the individual components and their interactions are used in building component-based systems. [12] distinguished between the following perspectives of CBS:

- the individual component *implementation*: the executable realization of a component,
- the component *interface (interaction)*: summarizes the properties of the component that are externally visible to the other parts of the system,
- real-time component: component technology for real-time systems should support specification and prediction of timing.

In fact, the time factor problem in reactive real-time component based systems (RTCBS) is not enough investigated [12]. The time issue should be included in the model of such systems, where an action from the environment is modeled as an input event, while the reaction from the system is an output event. A Timed Input Output Automata (TIOA) is able to specify a sequence of inputs and their corresponding outputs; moreover, it shows timing constraints on events occurring in the system. Consequently, a TIOA best represents the (SA) dynamics of a RTCBS. Time is considererd as continous. But this model does not consider data exchanges.

Even this model is well known for state explosion in industrial systems, we have chosen it to describe our systems. It has been defined by Alur-Dill [13] as a finite set of states and a finite set of clocks which are real-valued variables. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started or reset. Each transition of the system might reset some of the clocks, and has an associated enabling condition (EC) which is a constraint on the values of the clocks. A transition is enabled only if its corresponding state is reached and the current clock values satisfy its enabling condition.

4 Test Sequence Mutation

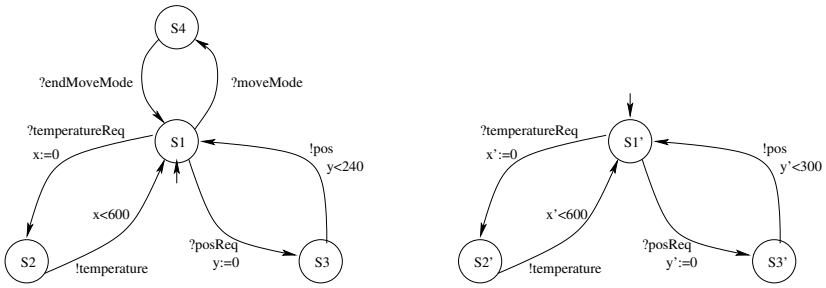
The purpose of our testing method is to check how the system reacts to hazards, and consequently to some stressful situations. A hazard could be defined as any unexpected event from the environment. In our study we handle external hazards. External hazards are modeled as actions received by the system or erroneous timing constraints. The automatic generation and integration of hazards in the test sequences can be respectively found in [14] and [15].

Each component of the RTCBS is described by a nominal and a degraded specification. Each of these specifications is represented by input-complete, reduced, strongly connected timed input-output automata TIOA.

For each component, we derive a test sequence for each controllable state. The generation method uses the test purpose technique [16]. The generated test sequences start with an input event and end with output event. For each controllable state we derive a sequence able to characterize this state among all others.

A state s of a TIOA is said controllable, if all reached states by s are connected to it by transitions labelled by only input actions.

Fig.1 shows an example of a nominal specification with initial state s_1 and a degraded specification with initial state s'_1. A transition is represented by an arrow between two states and labeled by (action; EC; Cs). The set of actions A={?temperatureReq,!temperature,?posReq,!pos,?moveMode,?endMoveMode} and the set of states S=$\{s_1, s_2, s_3, s_4\}$ and S'=$\{s'_1, s'_2, s'_3\}$



(a) Nominal spec. (b) Degraded spec.

Fig. 1. A nominal and a degraded specification

4.1 Hazard Injection

Hazards are inserted in the generated test sequences to simulate a hostile environment.

In fact, we will modify only some input actions in test sequences. Since we check the ability of the system to react correctly in presence of relevant actions, we do not allow to modify any input action of the degraded specification.

Suppose that $M = (S, A, C, T, s_0)$ is the nominal specification, and $M' = (S', A', C', T', s'_0)$ is the degraded one.

The designer decides which scenario he wants to integrate in a sequence based on some previous experiences. He may have already a collection of known scenari of non expected events from the environment. The insertion may be done automatically based on a probabilistic choice or manually. Let $T = \{Tr_1.Tr_2...Tr_n\}$ this sequence, where each Tr_i from M. Let $T' = \{Tr'_1.Tr'_2...Tr'_{n'}\}$ the mutated sequence. In the following we detail the most well known situations:

1. Replacing an input action
   We simulate the fact that another component sends an unexpected action to the tested component. The designer chooses a transition $Tr_i = (a_i; EC_i; C_{si})$

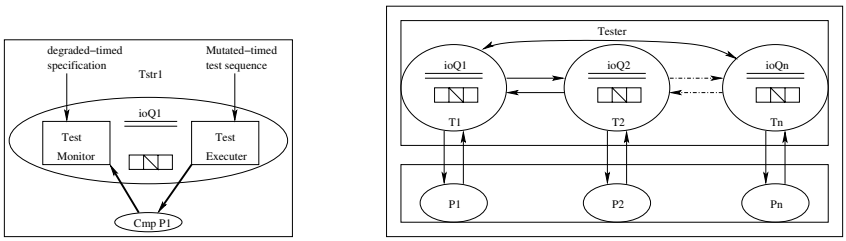
of T such that $a_i \in A_I - A'_I$. Then we change the action a_i with an action $a' \in A_I - A'_I$. Thus, the result T' is $T' = \{Tr_1.Tr_2... Tr_{i-1}.Tr'_i.Tr_{i+1}...Tr_n\}$ with $Tr'_i = (a'; EC_i; C_{si})$.

2. Changing the instant of an input action occurrence
   Here, we simulate the fact that another component sends the expected action, but not at the right moment. This could happen for example in case of heavy CPU processing. The designer chooses a transition $Tr_i = (a_i; EC_i; C_{si})$ of T such that $a_i \in A_I - A'_I$ and such that EC_i is bounded. Then we change the enabling condition EC_i with EC' such that EC' is the complementary of EC_i. In fact, we just delay the occurrence of the transition with an amount of time δ, such that the occurrence arrives later than expected. Thus, the result T'= $\{Tr_1.Tr_2... Tr_{i-1}.Tr'_i.Tr_{i+1}...Tr_n\}$ where $Tr'_i = (a_i; EC'; C_{si})$.
3. Exchanging two input actions
   We simulate the situation where other components have scheduling troubles. The designer chooses two transitions $Tr_i = (a_i; EC_i; C_{si})$ and $Tr_j = (a_j; EC_j; C_{sj})$ of T such that $a_i \in A_I - A'_I$ and $a_j \in A_I - A'_I$. Then we exchange the actions a_i and a_j. Thus, the result T' is $T' = \{Tr_1.Tr_2...Tr'_i...Tr'_j ...Tr_n\}$ with $Tr'_i = (a_j; EC_i; C_{si})$ and $Tr'_j = (a_i; EC_j; C_{sj})$.
4. Adding an unexpected transition
   Here, we simulate the fact that a component of the whole system sends an unexpected additional action. This may be caused for example by some troubles with any sensor. The designer chooses a transition $Tr_i = (a_i; EC_i; C_{si})$ of T such that $a_i \in A_I$. He also chooses an action $b \in A_I - A'_I$. We will add a transition in the sequence with the input action b without changing the timing conditions of the whole sequence. In fact, we insert the new action respecting the timing conditions allowed for Tr_i. Finally the resulting sequence becomes:
   $T' = \{Tr_1.Tr_2...Tr_{i-1}.Tr'.Tr'_i.Tr_{i+1}...Tr_n\}$ with Tr' =$(b; EC'; -)$ such that $EC' \subset EC_i$ and Tr'_i =$(a_i; EC_i; C_{si})$
5. Removing a transition
   In this part, we simulate the fact that an information is lost in the system. This could happen for many reasons, such that a problem in communication channels. The designer chooses a transition $Tr_i = (a_i; EC_i; C_{si})$ of T such that $a_i \in A_I - A'_I$ and we just remove it from the sequence. Thus the the resulting sequence becomes: T' =$\{Tr_1.Tr_2...Tr_{i-1}.Tr_{i+1}...Tr_n\}$.

5 Test Execution

Fig. 2 illustrates the test architecture we use. It consists of a set of distributed local testers. For each component, P_m, of the system, a dedicated Tester, T_m,is assigned. Each tester is also detailled with its modules : TEU (Test Executer Unit) and TMU (Test Monitor Unit).

This part is inspired by the work done in [17] dedicated to an adapted test architecture for distributed timed systems.



(a) unit tester (b) main architecture

Fig. 2. Test Architecture

5.1 Tester Coordination

Components communicate with each other through their corresponding testers. An output from component P_i to component P_j is achieved by sending the output to tester T_i which in turns sends to tester T_j, next, T_j forwards the output to P_j. In this model we ignore the time taken by the communication between components through testers (we only cnsider the amount of time taken by direct component communication).

Testers communicate with each other through the input/output queues (ioQ). The output transition from tester T_i, sent as an input, to tester T_j will wait on queue ioQ_j until this transition is enabled in T_j. On the other hand, the execution of a communication test sequence CTs_j by tester T_j will pause if this test sequence requires P_j to wait on an input from component P_i. The execution in tester T_j will resume after receiving the needed input from tester T_i; and thus, testers give a higher priority to handle inputs received from the components over inputs received from other testers that are stored in local testers' queues.

In order to cover time space, for any input action,"?A", the corresponding tester sends "A" as soon as the enabling condition EC of A satisfies the instant "t" reached by the local tester's clock, and another experiment is performed at the latest instant satisfying EC.

5.2 Test Execution Algorithm

The testing execution process is done in two phases. The first phase tests the robustness of each component separately, and the second phase tests the robustness of communication among components.

In both phases, each tester T_m executes, using TEU, the corresponding mutated test sequences and records, using TMU, the corresponding feedbacks from each component into execution traces without instantaneous evaluation. We check the robustness of each recorded execution trace from each component with its corresponding degraded specification. The theoretical framework of this issue is described in [18]

In the first phase, we ignore all communication requests from other components; and thus, all input actions from testers are sent at the instant where those inputs are needed in the nominal specification S^{P_m} without considering the communication from any other tester T_i , and therefore, here we are checking the robustness of the component in isolation, without taking any communication input from other components.

An execution of a mutated test sequence $Ts_m$by T_m gives a verdict success, iff, in its corresponding recorded execution trace, the reception of outputs from P_m are accepted in the degraded specification $S_d^{P_m}$.

In the second phase, all testers T_m execute, using TEU, the same communication test sequence ITs generated from the communication nominal specification. Tester T_i executes only its corresponding events (transitions) in ITs ignoring all transitions for other testers. An execution of tester T_m to a communication test sequence ITs_m gives a verdict success, iff , in its corresponding recorded execution trace, the sending of inputs and reception of outputs $O_m^{T_i}$ from P_m or T_i are accepted in the communication degraded specification S_d^{Cm}.

Algorithm 1 : Test execution
This code is executing on component / tester indexed "m".
Input: S^{Pm} Test sequences: Ts_m, ITs_m ; ITs_m initially is empty
Output: verdict pass, fail: set of failed components.

Phase 1:

1 For all components P_m
2 For all test sequences Ts_m
3 Select a test sequence Ts_j^m
4 Respecting timing constraints, apply each event of Ts_j^m via tester T_m to P_m.
5 Record the inputs and the corresponding feedbacks into traces for each component.
6 Check the $accept_{\mathcal{R}}$ of each recorded trace with its degraded specification
7 Assign the appropriate verdict (Robust — NonRobust)

Phase 2:

9 For all components:
This code is executing on component / tester indexed "m".
10 For all test sequences ITs_m
11 Select a test sequence ITs_j^m from ITs_m
12 For all transitions Tr_k of ITs_j^m
13 If (current event of ITs_j^m is a local transition:Tr_k^m)
14 Respecting timing constraints, T_m applies current event of ITs_j^m on component P_m.
15 Record the inputs and the corresponding feedbacks into traces for each P_m,

taking into consideration the update of enabling conditions
with time delays.
16 T_m forwards the *appropriate* output of $O_m^{T_i}$ to io-queue ioQ_i
of tester T_i.
17 If (tester T_i is waiting on this input)
18 Tester T_m Signals Tester T_i to resume execution:**signal**(T_i).
19 else if current event is not local transition: Tr_k^i
20 if transition $Tr_k^i is$ found in (already sent to) current local queue ioQ_m
21 Respecting timing constraints, of Tr_k^i, T_m applies Tr_k^i to P_m.
22 Record the corresponding traces for each component.
taking into consideration the update of enabling
conditions with time delays.
23 else if the other tester T_i is not waiting on an input from this tester
24 pause execution of T_m until it gets input from T_i: **wait** (T_m).
25 else if (the other tester T_i is waiting on an input from this tester T_m)
26 Supply T_m with the appropriate input from S^{P_m}
to be applied on P_m.
27 Record the corresponding traces for each component taking into
consideration the update of enabling conditions
with time delays. Identify a deadlocks state on test sequence ITs_j^m.
28 Signal all testers waiting on T_m.
29 Check the $accept_{\mathcal{R}}$ of each recorded trace with the degraded .
specification of the corresponding component.
30 Assign the appropriate verdict (Robust — NonRobust)

Comments on the algorithm First phase is locally testing robustness of each component and also identifies interoperable sequences. In the second phase, each tester T_m selects one of its sequences in ITs_m, and applies it, while the other testers parallely do the same with other sequences. At the end of the algorithm, all the possible combinations of sequences has been tested. Supposing that each tester has an average of k sequences, and N components, this gives about k^N test cases.

For each output transition of sequence ITs_m, tester T_m has to send this action to the corresponding tester, and eventually to signal it resuming of execution. If it is an input, the tester identifies if it comes from the environment or from a tester. In the last case, T_m applies the corresponding action only an other componant has already sent this action to P_m.

If a tester T_i is waiting on an input from T_j, it pauses the execution until the corresponding action arrives. However, if tester T_i is waiting on an input from T_j and reciproqually, then a deadlock case is identified.

6 Conclusion and Future Work

Robustness testing for real-Time component-based systems is discussed in this paper. To the best of our knowledge, quite a few works has been done in this

field. In this paper, we present a methodology for testing robustness of real-time component-based systems using fault injection and adequate distributed test architecture. Each component is described by its nominal specification and its degraded one. Three main contributions are noticeable in this method:

The first is the automatic generation of test sequence set for each component from its nominal specification. Then relevant faults are inserted in these sequences in order to simulate hostile environment.

The second contribution is that by the end of the first phase of the test execution, we are able to tell about all robust-implemented stand-alone components. In this phase, we experiment mutated test sequences on each component of the IUT and we record the results traces. These later are checked on the degraded specification. Each component is said to be robust if the verdict of experimentation of recorded traces on the degraded specification gives the verdict success.

A third contribution is the test-execution algorithm that executes and synchronizes test sequence execution on local testers. The synchronization is done via two atomic statements, Signal() and Wait(), and a set of input-output queues. A queue is attached to each tester. The robustness of the whole IUT is deduced if the communication events between components are accepted by the degraded specifications of all components.

As a future work, we intend to investigate more realistic hazard insertion by using metrics produced by real case studies. We need also to experiment this methodology in a real case study such as industrial control software or complex embedded systems.

We intend to use other real-time modeling languages as UML-RT or statecharts to describe RTCBS. We will reduce the state explosion problem inherent to the timed automata model.

References

1. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch: Why reuse is so hard. IEEE software (1995)
2. Khoumsi, A.: Timing issues in testing distributed systems. In: 4th IASTED International Conference on Software Engineering and Applications, SEA2000, November 2000. (2000)
3. Bertolino, A., Corradini, F., Inveradi, P., Muccini, H.: Deriving test plans from architectural descriptions. In: ACM Proceedings, International Conference on Software Engineering ICSE2000, June 2000. (2000)
4. Zimmerer, P.: Test architectures for testing distributed systems. In: 12th International software quality week (QW'99), May 1999. (1999)
5. Schmidt, H.: Trustworthy components-compositionality and prediction. The Journal of Systems and Software **65** (2003) 215–225
6. Zalewski, J.: Developing component-based software for real-time systems. In: 27th Euromicro Conference 2001: A Net Odyssey (euromicro'01), September 2001. (2001)

7. Tesanovic, A., Nystrom, D., Hansson, J., Norstrom, C.: Towards aspectual component-based development of real-time systems. In: Proceeding of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2003), February 2003. (2003)
8. Barton, J.H., Czeck, E.W., Segall, Z.Z., Siewiorek, D.P.: Fault injection experiments using fiat. IEEE Trans. Comput. **39** (1990) 575–582
9. Tsai, T.K., Iyer, R.K., Jewitt, D.: An approach towards benchmarking of fault-tolerant commercial systems. In: Symposium on Fault-Tolerant Computing. (1996) 314–323
10. Kropp, N.P., Jr., P.J.K., Siewiorek, D.P.: Automated robustness testing of off-the-shelf software components. In: Symposium on Fault-Tolerant Computing. (1998) 230–239
11. Castanet, R.: Les enjeux du test de robustesse. In: Journées du Réseau Thématique Prioritaire SECC. (2002)
12. Brinksma, E., Coulson, G., Crnkovic, I.: (Project ist-2001-34820 - artist- advanced real-time systems. roadmap: Component-based design and integration platforms) http://www.systemes-critiques.org/ARTIST/.
13. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
14. Fouchal, H., Rollet, A.: Embedded system testing. In: Proceedings of 7th International Conference on Principles of Distributed Systems (OPODIS 2003), December 10-13 2003, La Martinique, France. Lecture Notes in Computer Science, Springer-Verlag (2003)
15. Rollet, A.: Testing robustness of real-time embedded systems. In: Proceedings of Workshop On Testing Real-Time and Embedded Systems (WTRTES), Satellite Workshop of FM 2003 Symposium, Pisa, Italy - September 13, 2003. (2003)
16. Fouchal, H., Petitjean, E., Salva, S.: An User-Oriented Testing of Real Time Systems. In: Proceedings of the International Workshop on Real-Time Embeded Systems RTES'01 (London), IEEE Computer Society. (2001)
17. Khoumsi, A.: Testing distributed real-time systems in the presence of inaccurate clock synchronization. Journal of Information Soft. Technology (IST) **45** (2003)
18. Tarhini, A., Rollet, A., Fouchal, H.: A pragmatic approach for robustness testing on real time component based systems. In: The 3rd ACS/IEEE International Conference on Computer Systems and Application (AICSSA05), January 2-5 2005, Cairo, Egypt. (2004) accepted.