Databases and Information Systems IX
G. Arnicans et al. (Eds.)

doi:10.3233/978-1-61499-714-6-297

Automatic Distribution of Local Testers for Testing Distributed Systems

Jüri VAIN [1], Evelin HALLING, Gert KANTER, Aivo ANIER and Deepak PAL
Department of Computer Science, Tallinn University of Technology, Estonia
http://cs.ttu.ee/

Abstract. *Low-latency systems* where reaction time is primary success factor and design consideration, are serious challenge to existing integration and system level testing techniques. Modern cyber physical systems have grown to the scale of global geographic distribution and latency requirements are measured in nanoseconds. While existing tools support prescribed input profiles they seldom provide enough reactivity to run the tests with simultaneous and interdependent input profiles at remote front ends. Additional complexities emerge due to severe timing constraints the tests have to meet when test navigation decision time ranges near the message propagation time. Sufficient timing conditions for remote online testing have been proposed in remote Δ-testing method recently. We extend the Δ-testing by deploying testers on fully distributed test architecture. This approach reduces the test reaction time by almost a factor of two. We validate the method on a distributed oil pumping SCADA system case study.

Keywords. model-based testing, distributed systems, low-latency systems

1. Introduction

Modern large scale cyber-physical systems have grown to the size of global geographic distribution and their latency requirements are measured in microseconds or even nanoseconds. Such applications where latency is one of the primary design considerations are called *low-latency systems* and where it is of critical importance – to *time critical systems*. A typical example of distributed time critical system is smart energy grid (SEG) where delayed control signals can cause overloads and blackouts of whole regions. Thus, the proper timing is the main measure of success in SEG and often the hardest design concern.

Since large SEG-s systems are mostly distributed systems (by distributed systems we mean the systems where computations are performed on multiple networked computers that communicate and coordinate their actions by passing messages), their latency dynamics is influenced by many technical and non-technical factors. Just to name a few, energy consumption profile look up time (few milliseconds) may depend on the load profile, messaging middleware and the networking stacks of operating systems. Similarly, due to cache miss, the caching time can grow from microseconds to about hundred

[1]Corresponding Author: Jüri Vain; Department of Computer Science, Tallinn University of Technology, Akadeemia tee 15A, 19086 Tallinn, Estonia; E-mail: juri.vain@ttu.ee

milliseconds [1]. Reaching sufficient feature coverage by integration testing of such systems in the presence of numerous latency factors and their interdependences, is out of the reach of manual testing. Obvious implication is that scalable integration and system level testing presumes complex tools and techniques to assure the quality of the test results [2]. To achieve the confidence and trustability, the test suites need to be either correct by construction or verified against the test goals after they are generated. The need for automated test generation and their correctness assurance have given raise to model based testing (MBT) and the development of several commercial and academic MBT tools. In this paper, we interpret MBT in the standard way, i.e. as conformance testing that compares the expected behaviors described by the system requirements model with the observed behaviors of an actual implementation (implementation under test). For detailed overview of MBT and related tools we refer to [3] and [4].

2. Related Work

Testing distributed systems has been one of the MBT challenges since the beginning of the 90s. An attempt to standardize the test interfaces for distributed testing was made in ISO OSI Conformance Testing Methodology [5]. A general distributed test architecture, containing distributed interfaces, has been presented in Open Distributed Processing (ODP) Basic Reference Model (BRM), which is a generalized version of ISO distributed test architecture. First MBT approaches represented the test configurations as systems that can be modeled by finite state machines (FSM) with several distributed interfaces, called ports. An example of abstract distributed test architecture is proposed in [6]. This architecture suggests the Implementation Under Test (IUT) contains several ports that can be located physically far from each other. The testers are located in these nodes that have direct access to ports. There are also two strongly limiting assumptions: (i) the testers cannot communicate and synchronize with one another unless they communicate through the IUT, and (ii) no global clock is available. Under these assumptions a test generation method was developed in [6] for generating synchronizable test sequences of multi-port finite state machines. However, it was shown in [7] that no method that is based on the concept of synchronizable test sequences can ensure full fault coverage for all the testers. The reason is that for certain testers, given a FSM transition, there may not exist any synchronizable test sequence that can force the machine to traverse this transition. This is generally known as *controllability* and *observability* problem of distributed testers. These problems occur if a tester cannot determine either when to apply a particular input to IUT, or whether a particular output from IUT is generated in response to a specific input [8]. For instance, the controllability problem occurs when the tester at a port p_i is expected to send an input to IUT after IUT has responded to an input from the tester at some other port p_j, without sending an output to p_i. The tester at p_i is unable to decide whether IUT has received that input and so cannot know when to send its input. Similarly, the observability problem occurs when the tester at some port p_i is expected to receive an output from IUT in response to a given input at some port other than p_i and is unable to determine when to start and stop waiting. Such observability problems can introduce fault masking.

In [8], it is proposed to construct test sequences that cause no controllability and observability problems during their application. Unfortunately, offline generation of

test sequences is not always applicable. For instance, when the model of IUT is non-deterministic it needs instead of fixed test sequences online testers capable of handling non-deterministic behavior of IUT. But even this is not always possible. An alternative is to construct testers that includes external coordination messages. However, that creates communication overhead and possibly the delay introduced by the sending of each message. Finding an acceptable amount of coordination messages depends on timing constraints and finally amounts to finding a tradeoff between the controllability, observability and the cost of sending external coordination messages.

The need for retaining the timing and latency properties of testers became crucial natively when time critical cyber physical and low-latency systems were tested. Pioneering theoretical results have been published on test timing correctness in [9] where a remote abstract tester was proposed for testing distributed systems in a centralized manner. It was proven that if IUT ports are remotely observable and controllable then 2Δ-condition is sufficient for satisfying timing correctness of the test. Here, Δ denotes an upper bound of message propagation delay between tester and IUT ports. However, this condition makes remote testing problematic when 2Δ is close to timing constraints of IUT, e.g. the length of time interval when the test input has to reach port has definite effect on IUT. If the actual time interval between receiving an IUT output and sending subsequent test stimulus is longer than 2Δ the input may not reach the input port in time and the test goal cannot be reached.

In this paper we focus on distributed online testing of low latency and time-critical systems with distributed testers that can exchange synchronization messages that meet Δ-delay condition. In contrast to the centralized testing approach, our approach reduces the tester reaction time from 2Δ to Δ. The validation of proposed approach is demonstrated on a distributed oil pumping SCADA system case study.

3. Preliminaries

3.1. Model-Based Testing

In model-based testing, the formal requirements model of implementation under test describes how the system under test is required to behave. The model, built in a suitable machine interpretable formalism, can be used to automatically generate the test cases, either offline or online, and can also be used as the oracle that checks if the IUT behavior conforms to this model. Offline test generation means that tests are generated before test execution and executed when needed. In the case of online test generation the model is executed in lock step with the IUT. The communication between the model and the IUT involves controllable inputs of the IUT and observable outputs of the IUT.

There are multiple different formalisms used for building conformance testing models. Our choice is Uppaal timed automata (TA) [10] because the formalism is designed to express the timed behavior of state transition systems and there exists a family of tools that support model construction, verification and online model-based testing [11].

3.2. Uppaal Timed Automata

Uppaal Timed Automata [10] (UTA) used for the specification of the requirements are defined as a closed network of extended timed automata that are called *processes*. The

processes are combined into a single system by the parallel composition known from the process algebra CCS. An example of a system of two automata comprised of 3 locations and 2 transitions each is given in Figure 1.

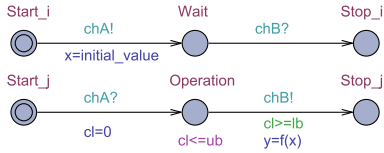

Figure 1. A parallel composition of Uppaal timed automata

The nodes of the automata are called *locations* and the directed edges *transitions*. The *state* of an automaton consists of its current location and assignments to all variables, including clocks. The initial locations of the automata are graphically denoted by an additional circle inside the location.

Synchronous communication between the processes is by hand-shake synchronization links that are called *channels*. A channel relates a pair of edges labeled with symbols for input actions denoted by e.g. `chA?` and `chB?` in Figure 1, and output actions denoted by `chA!` and `chB!`, where `chA` and `chB` are the names of the channels.

In Figure 1, there is an example of a model that represents a synchronous remote procedure call. The calling process Process_i and the callee process Process_j both include three locations and two synchronized transitions. Process_i, initially at location Start_i, initiates the call by executing the send action `chA!` that is synchronized with the receive action `chA?` in Process_j, that is initially at location Start_j. The location Operation denotes the situation where Process_j computes the output `y`. Once done, the control is returned to Process_i by the action `chB!`

The duration of the execution of the result is specified by the interval $[lb, ub]$ where the upper bound *ub* is given by the *invariant* `cl<=ub`, and the lower bound *lb* by the *guard condition* `cl>=lb` of the transition Operation $\rightarrow$ Stop_j. The *assignment* `cl=0` on the transition Start_j $\rightarrow$ Operation ensures that the clock *cl* is reset when the control reaches the location Operation. The global variables `x` and `y` model the input and output arguments of the remote procedure call, and the computation itself is modelled by the function `f(x)` defined in the declarations section of the Uppaal model.

The inputs and outputs of the test system are modeled using channels labeled in a special way described later. Asynchronous communication between processes is modeled using global variables accessible to all processes.

Formally the Uppaal timed automata are defined as follows. Let Σ denote a finite alphabet of actions $a, b, \ldots$ and C a finite set of real-valued variables p, q, r, denoting clocks. A guard is a conjunctive formula of atomic constraints of the form $p \sim n$ for $p \in C, \sim \in \{\geq, \leq, =, >, <\}$ and $n \in \mathbb{N}^+$. We use $G(C)$ to denote the set of clock guards. A timed automaton A is a tuple $\langle N, l_0, E, I \rangle$ where N is a finite set of locations (graphically denoted by nodes), $l_0 \in N$ is the initial location, $E \in N \times G(C) \times \Sigma \times 2^C \times N$ is the set of edges (an edge is denoted by an arc) and $I : N \rightarrow G(C)$ assigns invariants to locations (here we restrict to constraints in the form: $p \leq n$ or $p < n, n \in \mathbb{N}^+$. Without the loss of generality we assume that guard conditions are in conjunctive form with conjuncts including besides clock constraints also constraints on integer variables. Similarly to

clock conditions, the propositions on integer variables k are of the form $k \sim n$ for $n \in \mathbb{N}$, and $\sim \in \{\leq, \geq, =, >, <\}$. For the formal definition of Uppaal TA full semantics we refer the reader to [12] and [10].

4. Remote Testing

The test purpose most often used in MBT is conformance testing. In conformance testing the IUT is considered as a black-box, i.e., only the inputs and outputs of the system are externally controllable and observable respectively. The aim of black-box conformance testing according to [13] is to check if the behavior observable on system interface conforms to a given requirements specification. During testing, a tester executes selected test cases on an IUT and emits a test verdict (pass, fail, inconclusive). The verdict shows correctness in the sense of input-output conformance relation (IOCO) between IUT and the specification. The behavior of a IOCO-correct implementation should respect after some observations following restrictions:

(i) the outputs produced by IUT should be the same as allowed in the specification;

(ii) if a quiescent state (a situation where the system can not evolve without an input from the environment [14]) is reached in IUT, this should also be the case in the specification;

(iii) any time an input is possible in the specification, this should also be the case in the implementation.

The set of tests that forms a test suite is structured into test cases, each addressing some specific test purpose. In MBT, the test cases are generated from formal models that specify the expected behavior of the IUT and from the coverage criteria that constrain the behavior defined in IUT model with only those addressed by the test purpose. In our approach Uppaal Timed Automata (UTA) [10] are used as a formalism for modeling IUT behavior. This choice is motivated by the need to test the IUT with timing constraints so that the impact of propagation delays between the IUT and the tester can be taken into account when the test cases are generated and executed against remote real-time systems.

Another important aspect that needs to be addressed in remote testing is functional non-determinism of the IUT behavior with respect to test inputs. For nondeterministic systems only online testing (generating test stimuli on-the-fly) is applicable in contrast to that of deterministic systems where test sequences can be generated offline. Second source of non-determinism in remote testing of real-time systems is communication latency between the tester and the IUT that may lead to interleaving of inputs and outputs. This affects the generation of inputs for the IUT and the observation of outputs that may trigger a wrong test verdict. This problem has been described in [15], where the Δ-testability criterion (Δ describes the communication latency) has been proposed. The Δ-testability criterion ensures that wrong input/output interleaving never occurs.

4.1. Centralized Remote Testing

Let us first consider a centralized tester design case. In the case of centralized tester, all test inputs are generated by a single monolithic tester. This means that the centralized tester will generate an input for the IUT, waits for the result and continues with the next set of inputs and outputs until the test scenario has been finished. Thus, the tester has to

wait for the duration it takes the signal to be transmitted from the tester to the IUT's ports and the responses back from ports to the tester. In the case of IUT being distributed in a way that signal propagation time is non-negligible, this can lead into a situation where the tester is unable to generate the necessary input for the IUT in time due to message propagation latency. These timing issues can render testing an IUT impossible if the IUT is a distributed real-time system.

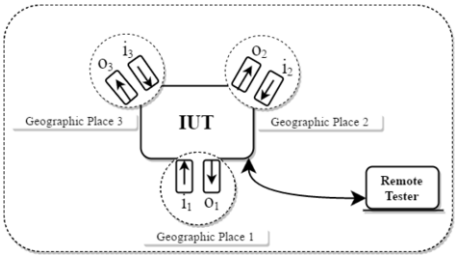

Figure 2. Remote tester communication architecture

To be more concrete, let us consider the remote testing architecture depicted in Figure 2 and the corresponding model depicted in Figure 3 and 4. In this case the IUT has 3 ports (p_1, p_2, p_3) in geographically different places to interact within the system, inputs i_1, i_2 and i_3 at ports p_1, p_2 and p_3 respectively and outputs o_1 at port p_1, o_2 at port p_2, o_3 at port p_3.

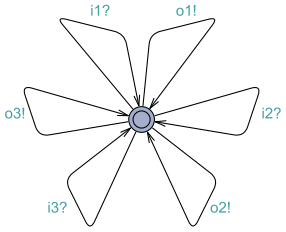

Figure 3. IUT model

We model a multi-ports timed automata by splitting the edges with multiple communication actions to a sequence of edges each labeled with exactly one action and connected via committed locations, so that all ports of such group are updated at the same time. In Figure 4 the labels on the edges represent the transitions and the transition tuple ($\texttt{L0}, \texttt{L1}, i_1!/(o_1?, o_2?)$) is represented by sequence of edges each labeled with exactly one action and connected via committed locations. For example the sequence of edges from location `L0` to `L1` with labels $i_1!$, $o_1?$ and $o_2?$ represents the multiple communication actions where the input $i_1!$ at port p_1 in location `L0` being able to trigger a transition that leads to the output $o_1?$ and $o_2?$ at ports p_1, p_2 respectively and the location becoming `L1`.

Using such splitting of edges with committed locations, we model a three port automata shown in Figure 4 where the tester sends an input i_1 to the port p_1 at `Geographic Place 1` and receives a response or outputs o_1 and o_2 from IUT at `Geographic Place 1` and `Geographic Place 2` respectively. After receiving the result, the tester is in location `L1`, it gets both i_3 on port p_3 and i_2 on port p_2. Then, either it follows the intended

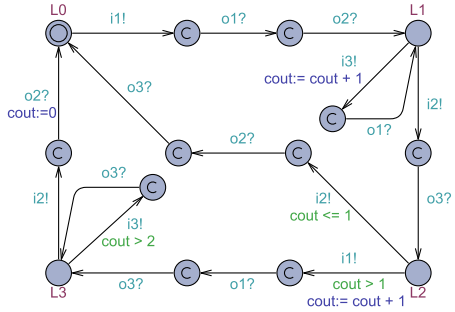

Figure 4. Remote Tester model

path sending i_3 before i_2, or it sends i_2 before i_3. If tester decides to send i_3 before i_2 it receives an output o_1 at port p_1 and returns to location `L1`. The transition is a self loop if its start and end locations are the same. If tester decides to send i_2, the IUT responds with an output o_3 at port p_3. Now, the tester is in location `L2`, it gets both i_1 on port p_1 and i_2 on port p_2. Based on guard condition and previously triggered inputs and received outputs the next input is sent to IUT and tester continues with the next set of inputs and outputs until the test scenario has been finished.

The described IUT is a real-time distributed system, which means that it has strict timing constraints for messaging between ports. More specifically, after sending the first input i_1 to port p_1 at `Geographic Place 1` and after receiving the response o_1 and o_2 at `Geographic Place 1` and `Geographic Place 2` respectively, the tester needs to decide and send the next input i_2 to port p_2 at `Geographic Place 2` or input i_3 to port p_3 at `Geographic Place 3` in Δ time. But, due to the fact that the tester is not at the same geographical place as the distributed IUT, it is unable to send the next input in time as the time it takes to receive the response and send the next input amounts to 2Δ, which is double the time allotted for the next input signal to arrive.

Consequently, the centralized remote testing approach is not suitable for testing a real-time distributed system if the system has strict timing constraints with non negligible signal propagation times between system ports. To overcome this problem, the centralized tester is decomposed and distributed as described in the next section.

5. Distributed Testing

The shortcoming of the centralized remote testing approach is mitigated with extending the Δ-testing idea by decomposing the monolithic remote tester into multiple local testers. These local testers are directly attached to the ports of the IUT. Thus, instead of bidirectional communication between a remote tester and the IUT, only unidirectional synchronization between the local testers is required. The local testers are generated in two steps: at first, a centralized remote tester is generated by applying the reactive planning online-tester synthesis method of [16], and second, a set of synchronizing local testers is derived by decomposing the monolithic tester into a set of location specific tester instances. Each tester instance needs to know now only the occurrence of i/o events

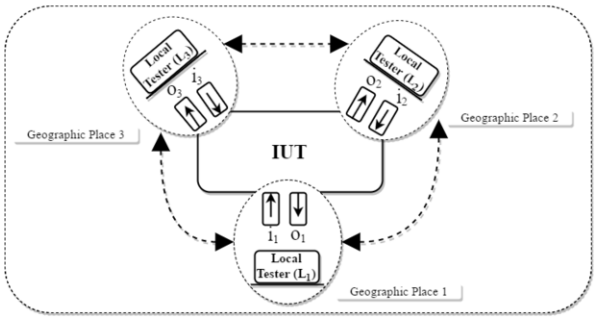

Figure 5. Distributed Local testers communication architecture

at other ports which determine its behavior. Possible reactions of the local tester to these events are already specified in its model and do not need further feedback to the event sender. The decomposing preserves the correctness of testers so that if the monolithic remote tester meets 2Δ requirement then the distributed testers meet (one) Δ-controllability requirement.

We apply the algorithm described in 5.1 to transform the centralized testing architecture depicted in Figure 2 into a set of communicating distributed local testers, the architecture of which is shown in Figure 5. After applying the algorithm, the message propagation time between the local tester and the IUT port has been eliminated because the tester is attached directly to the port. This means that the overall testing response time is also reduced, because previously the messages had to be transmitted over a channel with latency bidirectionally. The resulting architecture mitigates the timing issue by replacing the bidirectional communication with a unidirectional broadcast of the IUT output signals between the distributed local testers. The generated local tester models are shown in Figure 6, Figure 7, Figure 8 and Figure 9.

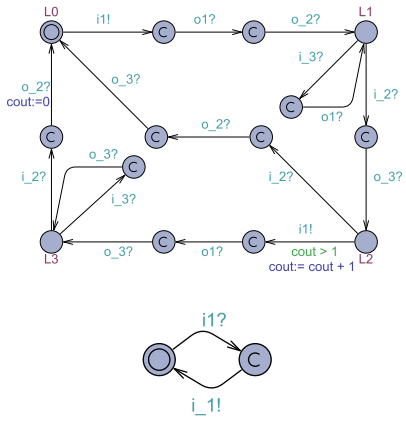

Figure 6. Local tester at Geographic Place 1

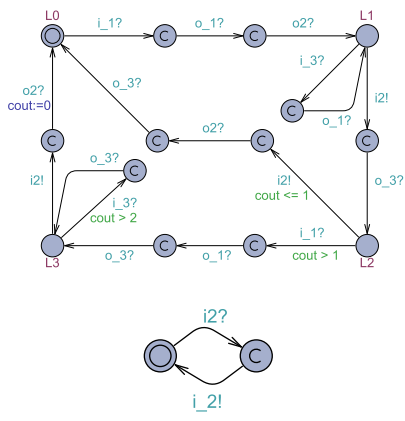

Figure 7. Local tester at Geographic Place 2

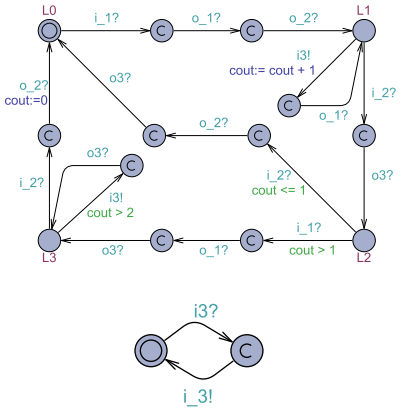

Figure 8. Local tester at Geographic Place 3

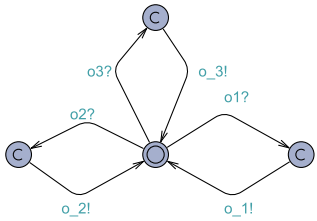

Figure 9. Output Event Synchronizer

5.1. Tester Distribution Algorithm

Let M^{MT} denote a monolithic remote tester model generated by applying the reactive planning online-tester synthesis method [16]. $Loc(IUT)$ denotes a set of geographically different port locations of IUT. The number of locations can be from 1 to n, where $n \in \mathbb{N}$ i.e. $Loc(IUT) = \{l_n | n \in \mathbb{N}\}$. Let P_{l_n} denotes a set of ports accessible in the location l_n.

1. For each l, $l \in Loc(IUT)$ we copy M^{MT} to M^l to be transformed to a location specific local tester instance.
2. For each M^l we go through all the edges in M^l. If the edge has a synchronizing channel and the channel does not belong to the the set of ports P_{l_n}, we do the following:
   - if the channel's action is *send*, we replace it with the co-action *receive*.
   - if the channel's action is *receive*, we do nothing.
3. For each M^l we add one more automaton that duplicates the input signals from M^l to IUT, attached to the set of ports P_{l_n} and broadcasts the duplicates to other local testers to synchronize the test runs at their local ports. Similarly the IUT local output event observations are broadcast to other testers for synchronization purposes like automaton in Figure 9.

6. Correctness of Tester Distribution Algorithm

To verify the correctness of distributed tester generation algorithm we check the bi-simulation equivalence relation between the model of monolithic centralized tester and that of distributed tester. For that the models are composed by parallel compositions so that one has a role of words generator on i/o alphabet and other the role of words acceptor machine. If the i/o language acceptance is established in one direction then the roles of models are reversed. Since the i/o alphabets of remote tester and distributed tester differ due to synchronizing messages of distributed tester the behaviors are compared based on the i/o alphabet observable on IUT ports only. Second adjustment of models to be made

for bi-simulation analysis is the reduction of message propagation delays to uniform basis either on Δ or 2Δ in both models. Assume (due to closed world assumption used in MBT):

- the centralized remote tester model: $M^{remote} = TA^{IUT} \parallel TA^{r-TST}$
- the distributed tester model: $M^{distrib} = TA^{IUT} \parallel |||_i \, TA_i^{d-TST}$ $i = [1, n]$, n - number of ports locations.
- to unify the timed words $TW(M^{remote})$ and $TW(M^{distrib})$ the communication delay between IUT and Tester is assumed.

Definition (correctness of tester distribution mapping): The mapping $M^{remote} \xrightarrow{Algorithm} M^{distrib}$ is correct if TA^{r-TST} and $|||_i \, TA_i^{d-TST}$ are observation bisimilar, i.e. if TA^{r-TST} and $|||_i \, TA_i^{d-TST}$ are respectively generating and accepting automata on common i/o alphabet $\Sigma^i \cup \Sigma^o$ then all timed words $TW(TA^{r-TST})$ are recognizable by $|||_i \, TA_i^{d-TST}$ and all timed words $TW(|||_i \, TA_i^{d-TST})$ are recognizable by TA^{r-TST}.

Here, alphabet $\Sigma^i \cup \Sigma^o$ includes i/o symbols used at IUT-TESTER interfaces of M^{remote} and $M^{distrib}$.

Correctness verification of the distribution mapping:

Step 1: (Constructing generating-accepting automata synchronous composition):

- label each output action of TA^{r-TST} with output symbol a! and its co-action in $|||_i \, TA_i^{d-TST}$ with input symbol a?;
- define parallel composition $TA^{r-TST} \parallel |||_i \, TA_i^{d-TST}$ with synchronous i/o actions.

Step 2: (Bisimilarity proof by model checking): TA^{r-TST} and $|||_i \, TA_i^{d-TST}$ are observation bisimilar if following holds: $M^{remote} \models \texttt{not deadlock} \wedge M^{distrib} \models \texttt{not deadlock} \Rightarrow TA^{r-TST} \parallel |||_j \, TA_j^{d-TST} \models \texttt{not deadlock}$ $j = [1, n]$, n - number of local testers , i.e. the composition of bisimilar testers must be non-blocking if the testers composed with IUT model separately are non-blocking.

7. Case Study

7.1. Use Case

The benefit of using the proposed method is demonstrated in the use case of an EMS (Energy Management System) which is integrated into the SCADA (Supervisory Control And Data Acquisition) system of an industrial consumer. An EMS is essentially a load balancing system. The target of the balancing system is the load on power supplies called feeders to an industrial consumer. These industrial power consumers have multiple feeders to power the devices required for their operations (e.g., pumps and pipeline heating systems). The motivation for balancing the power consumption between the feeders stems from the fact that the power companies can enforce fines on the industrial consumers if the power consumption exceeds certain thresholds due to safety considerations and possible damage to the equipment. Therefore, the consumer is motivated to share the power consuming devices among the feeders minimize or eliminate such energy consumption spikes completely.

Let us consider a use case in which an oil terminal has two feeders and multiple power consuming devices (consumers). The number of consumers can range from some

to many. In our use case we have 32 consumers, but in other cases it can be more. These consumers are both pumps and pipeline heating systems. The pumps have a high surge power consumption when starting up which must be taken into consideration when designing an EMS. The EMS monitors the current consumption by polling the consumers via a communication system (e.g., PROFIBUS, CAN bus or Industrial Ethernet). The PROFIBUS communication system is standardized in EN 50170 international standard.

Because the oil terminal stores oil it is considered an explosion hazard area and therefore, a special communication system that is certified for explosive areas - PROFIBUS PA (Process Automation) is used. PROFIBUS PA meets the 'Intrinsically Safe' (IS) and bus-powered requirements defined by IEC 61158-2. The maximum transfer rate of PROFIBUS PA is 31.25 kbit/s which can limit the system response speed if there are many devices connected to the PROFIBUS bus and each device has significant input and output data load.

The EMS is able to switch devices from being supplied from either feeder. Ideally, the power consumption is shared equally among both feeders at all times. This means that the EMS monitors the devices and switches devices over to other feeders if the power consumption is unbalanced among the feeders. In normal operation, the feeder loads are kept sufficiently low to accommodate new devices starting up in a way that the surge consumption will not exceed the threshold power of the feeders.

The EMS polls every power consumer periodically and updates the total consumption. Based on this total consumption, the EMS will command the power distribution devices to switch around from first feeder to the second in case the load on the first feeder is higher than on the second and vice versa.

In our use case we simulate the power consumption of the devices as the input to the IUT. The tester monitors the output (the EMS feeder load values). The test purpose is to verify that neither of the power loads exceed the specified threshold. Exceeding this limit might cause equipment damage and the power company can impose fines upon violating this limit.

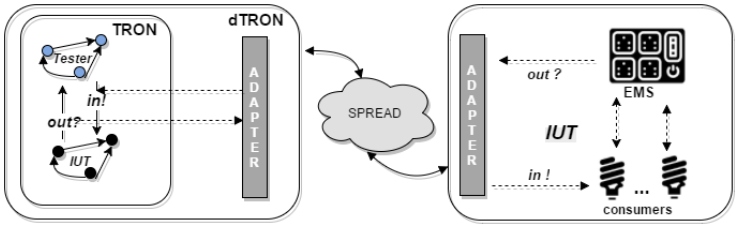

Figure 10. Case Study Test Architecture

The test architecture is depicted in Figure 10. In the right side of the figure, we can see the EMS and consumers as the implementation under test. The test model and test runner is on the left side. The test is executed via DTRON, which transmits the inputs and outputs via Spread. In the IUT and tester models we are going to introduce, the signals prefixed with $i\_$ or $o\_$ are synchronizing signals sent through Spread message serialization service . The signals without the aforementioned prefixes are internal signals which are not published to the Spread network. The input to the IUT is provided by the remote tester model is depicted in Figure 13 which simulates the device power consumption levels and creates challenging scenarios for the EMS. The EMS queries the consumers which are

modeled in Figure 12 and balances the load between the feeders based on the total power consumption monitoring data . The EMS model is shown in Figure 11 which displays the querying loop. The querying is performed in a loop due to the semi-duplex nature of communication in PROFIBUS networks. The EMS also takes the maximum power limit into account as the total power consumption must not exceed this level. This can be seen in the remote tester model shown in Figure 13. Remote tester nondeterministically selects a consumer and sends the level of energy consumption for that particular device to the input port of the IUT. Then the remote tester waits s time units before requesting the current feeder energy levels. On the model, it is indicated as $i\_get\_line\_balance$!. After receiving the current values the tester will check whether they are within allowed range. If the values exceed the limit the test verdict is fail. Otherwise the tester will continue with the next iteration.

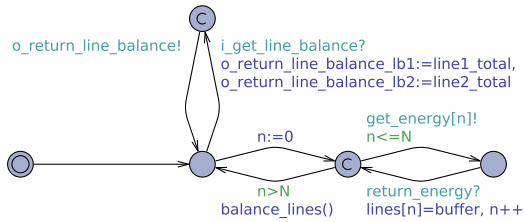

Figure 11. Energy Management System model

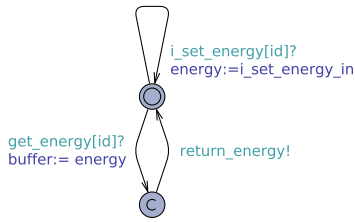

Figure 12. Consumer model

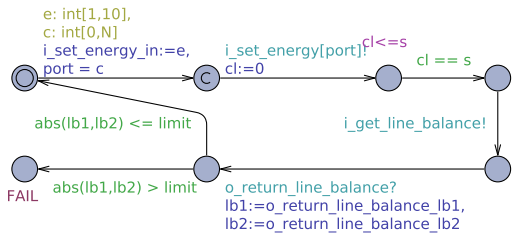

Figure 13. Remote Tester model

The communication delay between receiving the signal from EMS with the current feeder energy levels and sending input to the IUT is 2Δ. According to the specification the system must stabilize the load between feeders in stabilization time limit s after receiving the input. If Δ is very close to system stabilization time limit s indicated in the remote tester model in Figure 13 the remote tester fails to send the signal in time to the IUT.

For this reason, we introduce the distributed tester Figure 14 where each local component of the tester is closely coupled to the IUT input ports. As shown in chapter 5 this approach reduces the delay by up to Δ. This guarantees that after receiving the output from EMS we can send new input to IUT within less than s time units.

7.2. Test Execution Environment DTRON

Uppaal TRON [11] is a testing tool, based on Uppaal engine, suited for black-box conformance testing of timed systems [11]. DTRON [13] extends this enabling distributed

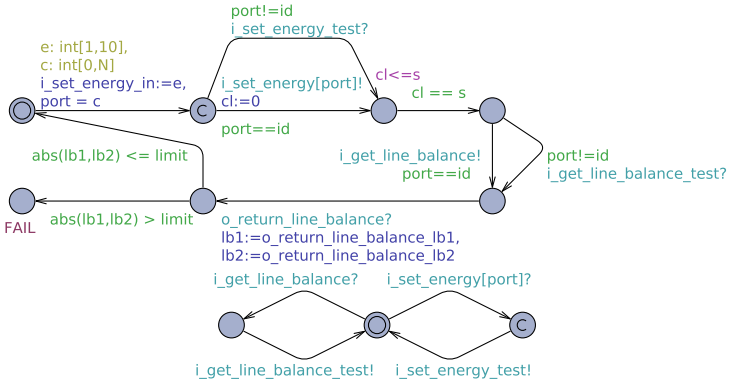

Figure 14. Parametrized local tester template for distributed testing

execution. It incorporates Network Time Protocol (NTP) based real-time clock corrections to give a global timestamp (t_1) to events at IUT adapter(s). These events are then globally serialized and published for other subscribers with a Spread toolkit [18]. Subscribers can be other IUT adapters, as well as DTRON instances. NTP based global time aware subscribers also timestamp the event received message (t_2) to compute and possibly compensate for the overhead time it takes for messaging overhead $\Delta = t_2 - t_1$.

Δ is essential in real-timed executions to compensate for messaging delays that may lead to false-negative non-conformance results for the test-runs. Messaging overhead caused by elongated event timings may also result in messages published in on order, but revived by subscribers in another. Δ can then also be used to re-order the messages in a given buffered time-window t_Δ. Due to the online monitoring capability DTRON supports the functionality of evaluating upper and lower bounds of message propagation delays by allowing the inspection of message timings. While having such a realistic network latency monitoring capability in DTRON our test correctness verification workflow takes into account theses delays. For verification of the deployed test configuration we make corresponding time parameter adjustments in the IUT model.

8. Conclusion

We extend the Δ-testing method proposed originally for single remote tester by introducing multiple local testers on fully distributed test architecture where testers are attached directly to the ports of IUT. Thus, instead of bidirectional communication between a remote tester and IUT only unidirectional synchronization between the local testers is needed in given solution. A constructive algorithm is proposed to generate local testers in two steps: at first, a monolithic remote tester is generated by applying the reactive planning online-tester synthesis method of [16], and second, a set of synchronizing local testers is derived by partitioning the monolithic tester into a set of location specific tester instances. The partitioning preserves the correctness of testers so that if the monolithic remote tester meets 2Δ requirement then the distributed testers meet (one) Δ-controllability requirement. Second contribution of the paper is that distributed testers are generated as Uppal Timed Automata. According to our best knowledge the real time distributed testers have not been constructed automatically in this formalism yet. As for

method implementation, the local testers are executed and communicating via distributed test execution environment DTRON [13]. We demonstrate that the distributed deployment architecture supported by DTRON and its message serialization service allows reducing the total test reaction time by almost a factor of two. The validation of proposed approach is demonstrated on an Energy Management System case study.

References

[1] A. Brook, Evolution and Practice: Low-latency Distributed Applications in Finance. Queue - Distributed Computing, ACM, New York (2015), vol. 13, no. 4, pp. 40-53.

[2] G. Hackenberg, M. Irlbeck, V. Koutsoumpas, and D. Bytschkow, Applying formal software engineering techniques to smart grids. In Software Engineering for the Smart Grid (SE4SG), 2012 International Workshop, IEEE (2012), pp. 50-56.

[3] M. Utting, A. Pretschner, and B. Legeard, A taxonomy of Model-based Testing. Software Testing, Verification & Reliability, John Wiley and Sons Ltd., Chichester, UK (2012), vol. 22, iss. 5, pp. 297-312.

[4] J. Zander, I. Schieferdecker, P. J. Mosterman (eds), Model-Based Testing for Embedded Systems. CRC Press (2011).

[5] ISO. Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework - Parts 1-5. International Standard IS-9646. ISO, Geneve (1991).

[6] G. Luo, R. Dssouli, G. v. Bochmann, P. Venkataram, A. Ghedamsi, Test generation with respect to distributed interfaces. Computer Standards & Interfaces, Elsevier (1994), vol. 16, iss. 2, pp.119-132.

[7] B. Sarikaya, G. v. Bochmann, Synchronization and specification issues in protocol testing. In: IEEE Trans. Commun., IEEE Press, New York (1984), pp. 389-395.

[8] R. M. Hierons, M. G. Merayo, M. Núñez, Implementation relations and test generation for systems with distributed interfaces. Springer-Verlag (2012), Distributed Computing, vol. 25, no. 1, pp. 35-62.

[9] A. David, K. G. Larsen, M. Mikuionis, O. L. Nguena Timo, A. Rollet, Remote Testing of Timed Specifications. In: Proceedings of the 25th IFIP International Conference on Testing Software and Systems (ICTSS 2013), Springer, Heidelberg (2013), pp. 65-81.

[10] J. Bengtsson, W. Yi, Timed Automata: Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets: Advances in Petri Nets. LNCS, Springer, Heidelberg (2004), vol. 3098, pp. 87–124.

[11] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, A. Skou, Testing Real-Time Systems Using UPPAAL. In: Hierons, R. M., Bowen, J. P., Harman, M. (eds.) Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers. LNCS, Springer, Heidelberg (2008), vol. 4949, pp. 77-117.

[12] G. Behrmann, A. David, K. G. Larsen, A Tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, Springer, Heidelberg (2004), vol. 3185, pp. 200-236.

[13] DTRON - Extension of TRON for distributed testing, http://www.cs.ttu.ee/dtron.

[14] J. Tretmans: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools, Springer-Verlag (1996), vol. 17, no. 3, pp. 103-120.

[15] R. Segala, Quiescence, fairness, testing, and the notion of implementation. In: Best, E. (eds.) 4th Intrenational Conference on Concurrency Theory (CONCUR'93). LNCS, Springer, Heidelberg (1993), vol. 715, pp. 324-338.

[16] J. Vain, M. Kääramees, M. Markvardt: Online testing of nondeterministic systems with reactive planning tester. In: Petre, L., Sere, K., Troubitsyna, E. (eds.) Dependability and Computer Engineering: Concepts for Software-Intensive Systems, IGI Global, Hershey (2012), pp. 113-150.

[17] A. Anier, J. Vain, Model based Continual Planning and Control for Assistive Robots. In: Proceedings of the International Conference on Health Informatics, SciTePress, Setúbal (2012), pp. 382-385.

[18] The Spread Toolkit, http://spread.org/.