Model Based Testing of Distributed Time Critical Systems

J"uri Vain1, Gert Kanter2, Seshadhri Srinivasan3

1Department of Software Science Tallinn University of Technology Tallinn, Estonia; juri.vain@ttu.ee
2Department of Software Science, Tallinn University of Technology, Tallinn, Estonia; gert.kanter@ttu.ee
3Berkeley Education Alliance for Research, Singapore; Seshadhri.srinivasan-bears@berkeley.sg

Abstract — Model-based testing incorporates steps such as test model construction, test purpose specification, test generation, deployment and execution. While the verification has not been traditionally an obligatory part of this process we extend the test development model by introducing model-based techniques, a tool and verification conditions of provably correct test development for time critical distributed systems. We demonstrate how Uppaal Timed Automata models and related tool family supports the development and verification of symbolic tests. Since distributed testing needs additional test deployment effort, we present the test controllability criteria for remote and distributed testing, provide an algorithm of distributing remote tests to improve the test performance and propose a technique of proving the correctness of distributed tests in terms of bisimulation equivalence between the remote and distributed tests.

I. INTRODUCTION

In the process of developing complex networked systems such as Cyber-Physical Systems (CPS) the problems of inherent concurrency over wide spectrum of services and heterogeneous architectures need to be addressed. The heterogeneous components introduce functional, timing, safety, performance, and security features on multiple scales. In safety/mission/business-critical applications the networking of feature-rich components needs to be paired with predictability of system’s emerging behaviour to guarantee required QoS. This is almost impossible to achieve without design validation methods that are scalable and relevant to holistic design views. While the features of functionality have gained major attention in traditional software development approaches, achieving the predictable timing of critical services in the presence of heterogeneous and evolving distributed architectures remains still a challenge. Therefore, validation methods like bench testing and encasing alone, although helpful and widely used, have become inadequate for full–fledged networked systems. As stated in [1] CPS software quality and software process productivity issues can be mitigated with model-based (MB) techniques and tools that operate on relevant level of abstraction. Model-based testing (MBT) as one group of these techniques provides the black-box testing solution for reducing software testing effort [2]. MBT suggests the use of abstract models for specifying the expected behaviour of the system under test (SUT) and automatically generating tests from models. According to Utting et al. [3] MBT incorporates steps such as SUT modelling, test purpose specification, test generation, test deployment and execution. Though MBT workflow relies inherently on the techniques of model engineering, the verification of the test development process and its products is not generally an obligatory part of MBT. On the other hand, the provably correct development (PCD) disciplines studied in [4], capitalize on the development process paired with verification and design correctness assurance steps. Applying PCD processes to testing is motivated by the need need to improve the trustability of testing results by showing their formal correctness through entire test development and execution process. In this paper we focus on the model-based online testing of distributed systems with timing constraints capitalizing on the correctness criteria and proving them through MBT workflow.

MBT is generally understood as conformance testing where the SUT is assumed to be a black-box where only its inputs and outputs are externally controllable and observable respectively. The internal behavior of the system is abstracted away. The aim of black-box conformance testing according to [3] is to check if the behaviour observable on the system interfaces conforms to the system requirements specification. During MBT a tester executes selected test cases (extracted from the system requirements model) by running SUT in the test harness and emits a test verdict (pass, fail, inconclusive). The verdict shows test result in the sense of conformance relation between SUT and the requirements model. A “classical” conformance relations is Input-Output Conformance (IOCO) introduced by Tretmans [5]. The behaviour of a IOCO-correct implementation should respect after some observations following restrictions:

(i) the outputs produced by SUT should be the same as allowed in the requirements model;

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

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

The set of tests that forms a test suite is divided into test cases, each addressing some specific test purpose. In MBT, the test cases are generated from formal models that specify the expected behaviour of the SUT and from the coverage criteria
that constrain the behaviour with only those addressed by the test purpose. In this paper Uppaal Timed Automata (Uppaal TA) [6] are used to model SUT behaviour. This choice is motivated by the need to test the SUT with timing constraints so that the impact of communication delays between the SUT and the tester can be taken into account when the test cases are generated and executed against a distributed time critical SUT.

Another crucial aspect that needs to be addressed when testing networked systems is non-determinism of SUT with respect to test inputs. It includes both functional and timing non-determinism. In nondeterministic systems online testing (generating test stimuli on-the-fly) is applicable in contrast to that of deterministic systems where test sequences can be generated offline, i.e. before running the tests. For instance, the source of non-determinism is communication latency between the tester and the SUT that may lead to mixed interleaving of test inputs and outputs to/from remote ports. The phenomena that are related to communication delays and non-determinism faced in distributed testing have been outlined and $\Delta$-testability criterion for such systems introduced in [7]. $\Delta$ describes the communication latency that ensures that SUT input and output messages do not get mixed when the tester communicates with spatially distributed components of SUT.

In the rest of the paper we demonstrate how Uppaal TA models and Uppaal tool family provide support for provably correct test development, i.e. constructing SUT models, specifying coverage criteria using these models, extracting symbolic tests from them, and adding test deployment information for running test on networked non-deterministic systems with timing constraints. Finally, we present the test controllability criteria for remote and distributed testing, provide an abstract tester distribution algorithm and a proof technique to demonstrate the bisimulation equivalence (relative to test i/o actions) between the centralized tests and distributed tests derived from the centralised ones.

II. CONSTRUCTING TEST MODELS

A. Modelling Timing Aspects of SUT

Model-based testing of timing aspects presumes formal representation of quantitative time related notions attributed to events and actions of SUT. Uppaal TA provide representation of timing constraints using clock variables, and synchronizations expressed either by clock conditions or special synchronization labels channels. In [8] we have suggested a modelling pattern called Action (depicted in Figure 1) for constructing the test models using few basic model elements and composition rules.

An Action models a fragment of system behavior on a given level of abstraction as an atomic state transition. The Action in the SUT or Tester model component is triggered by its input event $in?$ and Action termination (expectedly within some bounded time interval called response time) generates an output event $out!$ of that component. The SUT input events (stimuli in the testing context) are generated by Tester, and the output events (SUT responses), if not silent events, are assumed to be observable to the Tester. Synchronous interactions between SUT and Tester components are represented in Uppaal TA model by channels that link input/output events occurring simultaneously in the parallel components of SUT and Tester. The test data communicated over test interfaces are modelled by using global data structures such as variables and arrays accessible by both SUT and Tester.

Main timing attribute of the Action is its duration. Due to timing imperfections and abstraction it is often feasible to represent the duration in the SUT model as a closed interval $[l\text{\_bound}, u\text{\_bound}]$, where $l\text{\_bound}$ and $u\text{\_bound}$ denote lower and upper bound respectively. The duration interval $[l\text{\_bound}, u\text{\_bound}]$ is expressed in Uppaal TA as a pair of predicates on clocks as shown in Figure 1. The clock reset $clock = 0$ on the edge $Pre\_location \rightarrow Action$ makes the time constraint specification local to the Action and independent from accumulated durations of Actions executed earlier. The clock invariant $clock <= u\text{\_bound}$ of location $Action$ forces the Action to terminate latest at time instant $u\text{\_bound}$ after the clock reset. The guard $clock >= l\text{\_bound}$ on the edge $Action \rightarrow Post\_location$ defines the earliest time instant (w.r.t. clock reset) when the outgoing transition of Action can be executed.

From tester’s point of view SUT has two types of control states: passive and active. Note that in Uppaal TA, the states defined as valuations of data variables are called locations and there are many clock states bounded by duration interval that corresponds to a location. So, in passive states SUT is waiting for the test stimuli and in its active state SUT can choose a move to the next state. Due to the TA semantics the transition to new location can be delayed as long as the active state invariant is true. The location can be left earliest when the guard of outgoing transition $Action \rightarrow Post\_location$ evaluates to true. For instance, in Figure 1, the locations $Pre\_location$ and $Post\_location$ are passive while $Action$ is an active location.

We construct the test model consisting of one or many SUT and Tester automata by composing instances of the Action pattern using sequential, alternative and parallel composition.

Definition 1 Composition of Action patterns

Let $F_i$ and $F_j$ be Uppaal TA fragments composed of Action patterns with pre-locations $l^i_{pre}$, $l^j_{pre}$ and post-locations $l^i_{post}$, $l^j_{post}$ respectively. The composition of $F_i$ and $F_j$ is the union of elements of both fragments satisfying following conditions:

- sequential composition $F_i ; F_j$ is Uppaal TA fragment where $l^i_{post} = l^j_{pre}$;
- alternative composition $F_i + F_j$ is Uppaal TA fragment with $i_{pre}^{F_i} \rightarrow i_{pre}^{F_j}$ and $i_{post}^{F_i} \rightarrow i_{post}^{F_j}$;
- synchronous parallel composition $F_i \parallel F_j$ is Uppaal TA fragment where $F_i$ and $F_j$ are in different automata so that their transitions $i_{pre}^{F_i} \rightarrow \text{Action}_i$ and $i_{pre}^{F_j} \rightarrow \text{Action}_j$ are synchronized, and also $\text{Action}_i \rightarrow i_{post}^{F_i}$ and $\text{Action}_j \rightarrow i_{post}^{F_j}$ are synchronized.

We define well-formedness (wf) of test models according to the following inductive definition.

**Definition 2 Well-formedness**

- atomic Action pattern is well-formed;
- sequential composition of well-formed models is well-formed;
- alternative composition of well-formed models is well-formed if the output labels are distinguishable.
- parallel composition of well-formed models is well-formed if provided their execution intervals are consistent, i.e. $[\text{Bound}_i, \text{Bound}_i] \cap [\text{Bound}_j, \text{Bound}_j] \neq \emptyset$.

As shown in [9], large class of Uppaal TA modes of practical value can be transformed to bisimilar (w.r.t. i/o actions at test interfaces) well-formed representation.

In the rest of the paper we assume generally that $M^{SUT}$ is well-formed and denote it by $wf(M^{SUT})$. An example of the well-formed model is depicted in Figure 2 left.

**B. Correctness of Test Models**

Model based test generation algorithms presume that SUT models have properties which guarantee the feasibility of generated tests. For instance, among these properties are connectedness, input completeness, output observability and strong responsiveness. In this section we demonstrate how these properties formulated originally for IOTS (Input-Output Transition System) models [10] can be verified also on well-formed Uppaal TA models.

a) **Connected Control Structure and Output Observability:** Uppaal TA model is connected if there exists an executable path from any location to any other location. Since SUT represents an open system that is interacting with its environment we need for verification of the connectedness by model checking a nonrestrictive environment model which provides test stimuli and receives test responses in any possible order the SUT model can interact with its environment. Such an environment model is canonical tester proposed in [10]. A canonical tester can be created for well-formed SUT model in Figure 2 left using the pattern shown in Figure 2 right.

The canonical tester composed in parallel with SUT model implements random walk test strategy that is useful in endurance testing. On the other hand, it may be very inefficient when the testing purpose is to cover few selected elements of SUT model. Having synchronous parallel composition of SUT and the canonical tester (shown in Figure 2) the connectedness of SUT can be model checked with query (1) expressed in Computation Tree Logic CTL. Query (1) expresses the absence of deadlocks in interaction between SUT and the canonical tester.

\[
A[] \text{not deadlock} \quad (1)
\]

The output observability means that all state transitions of the SUT model are observable and identifiable by its outputs. By Definition 2 the output observability of SUT models is a co-product of well-formedness because each input event and Action location has an outgoing edge that generates a locally (w.r.t. the Action input event) unique output event.

b) **Input Enabledness:** Input enabledness of SUT models means that no blocking of SUT due to irrelevant test inputs can occur. Straightforward way of transforming a SUT model (if the number of inputs is bounded) to input enabled automaton by introducing self-looping transitions with input labels that are not represented on other transitions that depart from the same location. That makes SUT modelling tedious and causes an exponential increase of its size. Alternatively, when relying on the notion of observational equivalence one can approximate the input enabledness in Uppaal TA by exploiting the semantics of synchronizing channels and encoding input symbols as boolean variables $I_1, \ldots, I_n \in \Sigma^m$. Then the prelocations $i_{pre}^{I_j}$ of actions $A_i$ (see Figure 2) need to be modified by applying following transformation:

- assume there are $k$ outgoing edges from pre-location $i_{pre}^{I_j}$ of action $A_i$, each of these edges $e_j$ is labeled with one input symbol $I_j$, $j = 1, k$ from the input alphabet $\Sigma^m(M^{SUT})$;
- add a self-loop edge $i_{pre}^{I_j} \rightarrow i_{pre}^{I_j}$ that models the acceptance of all inputs in $\Sigma^m(M^{SUT})$ except $I_j$, $j = 1, k$ and specify the guard of this auxiliary edge with boolean expression: $not(\bigvee_{j=1}^{k} I_j)$.

Provided the branching factor $B$ of the edges that are outgoing from $i_{pre}^{I_j}$ is, as a rule, substantially smaller than the size of input alphabet $\Sigma^m(M^{SUT})$, we can save compared to straightforward algorithm $\Sigma^m(M^{SUT}) - B(i_{pre}^{I_j})$ edges at each pre-location of the Action pattern. Note that due to the $wf-$construction rules the number of pre-locations never exceeds the number of actions in the model. That is due to alternative composition that merges pre-locations of the composition. A fragment of alternative composition accepting all inputs in $\Sigma^m(M^{SUT})$ is depicted in Figure 3 (time constraints are ignored here for clarity). Symbols $I_1$ and $I_2$ in the figure denote predicates $Input == i_1$ and $Input == i_2$ respectively.

c) **Strong Responsiveness:** Strong responsiveness (SR) means that SUT model always reaches a quiescent state (location) after executing finite number of transitions. In other words, there is not reachable livelocks (a loop that includes only $\varepsilon$-transitions) in the SUT model. Since transforming $M^{SUT}$ to $wf(M^{SUT})$ does not eliminate $\varepsilon$-transitions there is no guarantee that $wf(M^{SUT})$ is strongly responsive by default. Strong responsiveness is verified by Algorithm 1.

**Algorithm 1 Checking strong responsiveness**

1) By input enabled Action pattern in Figure 3 the input events of $M^{SUT}$ are encoded by using channel in? and
III. CORRECTNESS OF TESTS

A. Structural coverage of Tests

The purpose of MBT is to detect the violation of conformance relations between the model and SUT. Therefore, tests are expected to cover possibly many elements and behaviors of the SUT model within the time and computation resources allocated for testing. The structural coverage criteria are defined in terms of structural elements (states, transitions, conditions) of the SUT model. Rushby et al have suggested in [13] to express the structural coverage of state machine models by means of trap-variables. To specify the coverage items in the model the expressions assigning boolean value true to trap-variables are added to the edges of the SUT model. When using traps as coverage items we declare the test being coverage correct if the test chooses SUT inputs so that all traps are true when test terminates.

Definition 3 (Trap) coverage correctness of the test

We say that a test is coverage correct (with respect to the specified set of traps) if the test run covers all the transitions that are labelled with traps in the SUT model.

Definition 4 Optimality of the test

We say that the test is length (time) optimal if there is no shorter (resp. faster) test run among those being coverage correct.

In the following we provide an ad hoc procedure of verifying the coverage correctness in terms of model checking queries and model building constraints.

Direct way of verifying the coverage correctness of the test in Uppaal tool is to model check:

\[ M^{SUT} \models M^{Test} \models A \forall (i : int[1..n]) t[i] \]

where \( t[i] \) denotes the \( i \)-th element of the array \( t \) of traps. The test model of query (2) is the synchronous parallel composition of SUT and test automata. An example of this composition is depicted in Figure 4.
B. Boundedness of tests

Testing time depends on the length and duration of executing the test scenarios satisfying given coverage criteria.

Definition 5 Length-wise boundedness of the test

A test is length-wise k-bounded (consisting of k stimuli/responses at most) if there exists an upper bound k to all test runs of the test model $M^{SUT} \parallel M^{Test}$.

When proving the length-wise boundedness of tests using Uppaal model checker the properties of well-formed SUT models simplify the task considerably. Having a $w f(M^{SUT})$ we set the lower and upper bounds $lb_i$ and $ub_i$ of its actions to 1, i.e., $lb_i = ub_i = 1$ for all $i \in [1, \ldots, |Action|]$. Then the length of the test sequence and its duration are numerically equal. When checking an integer bound k of test sequences the following model checking query proves the coverage of $n$ traps with at most k stimuli and responses

$$w f(M^{SUT} \parallel M^{Test}) = A \oslash (\forall (i : int[1, n]) t[i]) \land TM \leq k,$$

where $TM$ is a clock variable introduced to represent the progress of global time in the model. Note that in the test model $M^T$ we assume the duration of input selection actions is negligible compared to that of durations of $M^{SUT}$ actions. It means Action locations in $M^{Test}$ are of type "committed" (instantaneous).

Generalizing this approach for SUT models with arbitrary time constraints we can assume that all edges of $w f(M^{SUT})$ are attributed with time constraints satisfying conditions of well-formedness described in Section 2.1.

Definition 6 Time-wise boundedness of the test

A test is time-wise bounded if there exists an upper bound $TH$ to the duration of all test runs of the test model $M^{SUT} \parallel M^{Test}$.

Verification of time-wise boundedness requires model checking of (4):

$$w f(M^{SUT} \parallel M^{Test}) = A \oslash (\forall (i : int[1, n]) t[i]) \land TM \leq TH$$

Since not all of $M^{SUT}$ edges need to be labeled with traps (and not covered by test) we apply compaction procedure to $M^{SUT}$ to abstract away from the excess of information (for IOCO testing) in $M^{SUT}$. With the compaction procedure we merge the sequences of trapless edges with the trap-labelled edge the trapless ones are precedings. As the result, the sequence of trapless actions is merged with the trap labeled action making an aggregate. The first constituent edge of the aggregate contributes its input event and the last edge its output event. The other i/o events of the aggregate will be hidden from the symbolic model and will be implemented in the test adapter described in Section IV. The lower and upper bounds for the composite action are computed: the lower bound is the sum of lower bounds of the shortest path in the aggregate and the upper bound is the sum of upper bounds of the longest path of the aggregate plus the longest upper bound (the later is needed to compute the test termination condition). After compaction of deterministic SUT model it can be proved that the duration $TH$ of coverage correct tests have length satisfying following bound condition:

$$\sum_{i} lb_i \leq TH \leq \sum_{i} ub_i + max(ub_i), \ i = 1, n$$

where $n$ is the number of traps in $M^{SUT}$. In case of non-deterministic SUT models, for showing the length-wise boundedness of generated tests we need an assumption of fairness of $M^{SUT}$. A model $M$ is $k$-fair if the difference in the number of executions of alternative transitions of non-deterministic choices (sharing same departure location) never exceeds the bound $k$. During the test run the test execution environment DTRON [9] is capable of monitoring the $k$-fairness and reporting about its violations. The safe upper bound estimate of the test length in case of non-deterministic models can be calculated for the worst case by multiplying the deterministic upper bound by factor $k$. The lower bound still remains $\sum lb_i$. 
IV. Correctness of Test Deployment

A. Test adapters

The execution of symbolic tests presumes test adapters that map symbolic i/o alphabet used in the test model $M^{SUT}||M^{Test}$ to SUT executable inputs. Similarly, real outputs from SUT need to be transformed back to symbolic outputs to be compared for conformance checking. Both mappings performed by test adapters may introduce additional delays that are not reflected neither in SUT nor abstract test models. Also, distributed test configurations may introduce delays due to signal propagation time that can alter ordering of test stimuli and responses specified in the model. Network monitors, e.g., the one included in DTRON tool [9] measure the latency of form $\Delta = \delta_i, \delta_u$ at each test input and output adapter. To verify the feasibility of the executable test suite, the latency estimates need to be incorporated also in the test model and their impact re-verified against the correctness conditions defined in the earlier development steps.

The key property to be verified when deploying MBT test for remote testing of distributed SUT is $\Delta - testability$. The concept proposed in [7] introduces parameter $\Delta$ which defines minimum delay between consecutive test stimuli necessary to maintain the ordering of input-output events at remote SUT ports. When verifying the correctness of test deployment with test adapters as well as for remote testing one needs to proceed as follows:

Step 1: estimate the latency at each input and output adapter. For any input symbol $a \in \Sigma^I(M^{SUT})$ and any output symbol $b \in \Sigma^O(M^{SUT})$ get the interval estimates of its total latency (including delay caused by adapters and propagation delays): $\Delta_a = [\delta_a, \delta_a^u]$ and $\Delta_b = [\delta_b, \delta_b^u]$ respectively.

Step 2: modify the timed guards $Grd$ and invariants $Inv$ of each action of $w_f(M^{SUT})$ as follows:
- $Inv \equiv \delta_l \leq lb \rightarrow Inv' \equiv \delta_l \leq lb + \delta_a + \delta_b^u$
- $Grd \equiv \delta_l \leq lb \rightarrow Grd' \equiv \delta_l \leq lb + \delta_a^t + \delta_b^u$

Step 3: rerun the verification tasks of earlier verification steps with $\Delta$-extended model $w_f(M^{SUT+\Delta})$.

The procedure of constructing the test adapters in testing framework DTRON are described in detail in [9].

B. Distributed test deployment

In remote testing sending test inputs and waiting for outputs is assumed to take not less than $2\Delta$. This is time needed for signal travelling from tester to SUT ports and back to tester. Thus, when testing non-deterministic systems, for test to be on-line controllable we face $2\Delta$ delay barrier. The consequence is that for realtime systems that are expected to react to new inputs sooner than $2\Delta$ after providing some output, the remote testing is not applicable. A remote test is $2\Delta$-controllable if in the presence of network (incl. test harness) delays less than $\Delta$ wrong interleavings of inputs and outputs are excluded.

The performance and reaction time restrictions caused by $2\Delta$ delay in remote testing can be mitigated by decomposing the central remote test to multiple local tests which are attached to the SUT test ports directly. Instead of bidirectional communication between a remote test and the SUT, only unidirectional synchronization between the local tests is required. In the approach of [11] the local tests are generated in two steps: at first, a centralized remote test is generated by applying the reactive planning online-tester synthesis method of [12], and second, a set of synchronizing local tests is derived by decomposing the test into a set of location specific test instances. The decomposition preserves the test correctness so that if the original remote test is $2\Delta$-controllable then the distributed test is $\Delta$-controllable and bisimilar to original with respect to the test interface i/o actions.

Algorithm 2 Test distribution

Let $M^{CT}$ be a model of a centralized remote tester, $Loc(SUT) = \{l_i \mid i \in [1, n]\}$ a set of $n$ spatially separated port locations of SUT, and $P^F$ a set of SUT test ports accessible in the location $l_i$.

Step 1: Copy an instance $M^{CT}_i$ of the centralized remote tester $M^{CT}$ for each $l_i \in Loc(SUT)$.

Step 2: For each of the $M^{CT}_i$, $i \in [1, n]$:

Case 1: If the edge of $M^{CT}_i$ is labeled with an i/o action $a_k^t$ interacting with any local port $p$ of location $l_i \in Loc(SUT)$, then refine the edge with $a_k^t$ so that the i/o event is broadcasted also to the instances $a_k^t$ of same action $a_k$ in other locations $l_j \in Loc(SUT)$, $i \neq j$.

Case 2: If the edge of $M^{CT}_i$ is labeled with an output action $a_k^o$ interacting with the port $p$ of location $l_j \in Loc(SUT)$, where $i \neq j$, then substitute the output action $a_k^o$ with an input action $a_k^t$ and synchronize it with the instance $a_k^t$ in $M^{CT}_j$ via broadcast channel (created in Step 2 Case 1).

C. Correctness of Test Distribution

To verify the correctness of distributed test deployment we check (relative with respect to test i/o actions) bisimulation equivalence between the model $M^C = M^{CT}||M^{SUT}$ of the centralized remote test and the model $M^D = M^{DT}||M^{SUT}$ where $M^{DT}$ is derived from $M^{CT}$ by distributing it to a local tests $M^{DT}_i = ||M^{CT}_i, i \in [1, n]$.

For the models $M^C$ and $M^D$ are composed by synchronous parallel compositions so that one has a role of words generator on the test i/o alphabet and other the role of words acceptor. If the i/o language acceptance is established in one direction then the roles of models are inverted. Second adjustment of models to be made for bisimulation analysis is the reduction of message propagation delays in the models to uniform basis. Without loss of generality we can assume it is 0 in both models.

The mapping $M^{CT} \xrightarrow{\text{Algorithm 2}} M^{DT}$ is correct if $M^{CT}$ and $M^{DT}$ are observation bisimilar, i.e. if $M^{CT}$ and $M^{DT}$ are resp. generated and accepting Uppaal TA on common test i/o alphabet $\Sigma^I \cup \Sigma^O$ then all timed words $TW(M^{CT})$ generated by $M^{CT}$ are recognizable by $M^{DT}$ and all timed words $TW(M^{DT})$ generated by $M^{DT}$ are recognizable by $TW(M^{CT})$.

D. Verification:

Step 1: Constructing synchronous parallel composition of the generating and accepting automata:
Split all the edges $e$ of $M^C$ that carry the label of a test $i/o$ action into two edges $e'$ and $e''$ connected via committed location, where $e'$ copies the labeling of $e$ and $e''$ is labeled with a unique auxiliary output action (channel) name $Ch$. Do the same splitting with the corresponding edges $e$ of $M^D$ with the difference that instead of auxiliary actions $Ch$ the edges $e''$ are labeled with their co-actions $Ch!$. This ensures that when ever a test $i/o$ action is enabled in $M^C$ the same action must be enabled also in $M^D$ provided models are composed in parallel with auxiliary synchronization conditions, denoted $M^C |_{aux} M^D$. If this is not the case then the deadlock in $M^C |_{aux} M^D$ is reachable. Note that due to the semantics of Uppaal TA the synchronization via channels is symmetric and it suffices checking the deadlocks without inverting the direction of auxiliary synchronization channel between $M^C$ and $M^D$.

**Step 2: Bisimilarity proof by model checking**

The composition of bisimilar testers must be non-blocking if the testers composed with SUT model separately are non-blocking, i.e. if the following holds:

$M^C \not\vdash not\ deadlock \land M^D \not\vdash not\ deadlock \implies M^C |_{aux} M^D \not\vdash not\ deadlock.$

**CONCLUSION AND DISCUSSION**

This work has been motivated by the need to increase the trust on testing results paired with the predictability of systems emerging behaviour to guarantee multi-critical QoS in complex systems. We have focused on the formal correctness aspects of model-based online testing of distributed systems with timing constraints. The approach demonstrated is capitalizing on the correctness criteria and model checking based verification technique that covers main steps of MBT workflow. Due to the space limit the scalability and complexity issues have been addressed only partially and the authors see further opportunities in integration of these techniques to improve the scalability of test verification. As an extension of our current work along this line we see possible improvements in pairing our provably correct test development approach with contract based design methodology, aspect-oriented modelling, compositional verification, and model abstraction techniques.

**REFERENCES**


