Multi-Fragment Markov Model Guided Online Test Generation for MPSoC

Jüri Vain[1], Leonidas Tsiopoulos[1], Vyacheslav Kharchenko[2], Apneet Kaur[1], Maksim Jenihhin[1], and Jaan Raik[1]

[1] Tallinn University of Technology, Tallinn, Estonia,
{firstname.lastname}@ttu.ee
[2] National Aerospace University KhAI, Kharkiv, Ukraine,
{firstname.lastname}@csn.khai.edu

Abstract. Online monitoring and model-based validation are commonly accepted quality assurance measures for mission critical systems. We propose an integrated approach where the Multi-Fragment Markov Models (MFMM) are used for specifying the system reliability and security related behavior on high-level of abstraction, and the more concrete state and timing constraints related with MFMM are specified explicitly using Uppaal Probabilistic Timed Automata (UPTA). To interrelate these two model classes we demonstrate how the MFMM is mapped to UPTA. The second contribution is the test case selection mechanism for online testing where the test cases are prioritized by probabilities of execution modes. The hypotheses on which mode the system switches next are provided by MFMM and the hypotheses are tested using UPTA models that specify the mode behavior forming a test suite for online conformance testing of modes. The approach is illustrated with the Bonfire Multi-Processor System-on-Chip.

Keywords: Formal methods, Uppaal Timed Automata; verification; Markov models, model-based testing, Multi-Processor System-on-Chip

1 Introduction

The spacecraft systems are inherently mission critical. Their verification can be undertaken during several different stages [2, 9, 13]. Such stages are (i) development stage, (ii) qualification stage, (iii) acceptance stage, (iv) flight preparation stage, (v) flight execution stage and (vi) after landing stage. For spacecraft systems the complete verification functions cannot be exercised in full on the ground due to the inability to simulate complete conditions of the flight. Faults can occur during the flight due to, e.g. cosmic particles and aging. Furthermore, blocking of certain functions due to hardware and software faults might be necessary in order to guarantee the safe operation of the system.

Following from the results presented in [13], in this paper we focus on the *online corrective verification* implemented by combining predictive Markov modeling with online Model-Based Testing (MBT).

Online monitoring and model-based validation techniques are commonly accepted quality assurance measures for mission critical systems. Online testing is considered to be the most appropriate technique for MBT of embedded systems where the implementation under test (IUT) is modeled using nondeterministic models [18]. The nondeterminism stems from the physical nature of the IUT, particularly, its internally interacting parallel processes, loose timing conditions, and hardware related jitter. Often, the term *on-the-fly* is used instead of online testing to describe the test generation and execution at runtime.

If the critical system behavior is complex it is hard or practically impossible to predict all potential causes why the system can deviate from its specified behavior, e.g. would it be due to the bugs, hardware faults or cyber-attacks, unexpected load profiles or other reasons that have impact on systems QoS. Therefore, in model-based approaches the causal relations are often approximated with probabilistic ones. On the other hand, for error detection and correction the online verification of design correctness should explore the more concrete state space for explicit conditions under which the faults expose.

We propose an integrated approach with Multi-Fragment Markov Models (MFMM) [16] and Uppaal Probabilistic Timed Automata (UPTA) [8]. MFMM are used for specifying the system reliability and security related probabilistic behavior on high-level of abstraction which is relevant for Markov analysis and assessment of the availability of system functions depending on scenarios for online execution and verification. Besides, MFMM allow to describe probabilistic behavior considering time-varying system parameters, in particular, failure rate caused by software design faults. The more concrete state and timing constraints related with MFMM are specified explicitly with UPTA. To interrelate these two model classes we demonstrate how the MFMM is mapped to UPTA which is more relevant for explicit state model checking and model-based test generation.

The second contribution is the test case selection mechanism for online testing which prioritizes the test cases by probabilities of modes to be confirmed by online testing. The hypotheses on which mode the system switches next are provided by MFMM and the hypotheses are tested by running conformance tests against the UPTA models that specify the mode to be identified.

The approach is illustrated with the Bonfire Multi-Processor System-on-Chip (MPSoC) example maintained as an open-source project, available at [4], where the probabilistic model checking of high-level mode switching scenarios is applied on MFMM. A set of UPTA models that refines the behavior of MFMM modes is applied thereafter as a test suite for model-based conformance testing to confirm the hypothesis that the system is in a given mode.

2 Related Work

Markov chains is a well established model-based analytical tool for understanding stochastic systems behavior [11]. Stochastic approaches based on Markov models have been applied on multi-processor systems [5].

Related to the integration of Markov models and system MBT approaches, several works exist for improving the testing efficiency. For example one of the first works presented a method for statistical testing based on a Markov chain model of software usage [19]. A more recent related work by Kashyap et al. [10] proposes using Markov chains to create system models from available system usage data in order to create test plans that employ a "likelihood-based", test prioritization scheme considerably improving the coverage of generated test suites.

Corresponding testing methodology for the automotive domain is presented by Siegl et al. [14]. A Markov Chain Usage Model is studied to describe all possible usage scenarios of the IUT and provided a basis to derive systematically test cases. These test cases can be further processed inside the Extended Automation Method employed by AUDI AG. The approach makes it also possible to get both indicators for the test-planning in advance and to obtain dependability measures after the test cases have been processed.

All the aforementioned works are concerned with offline MBT while our approach is targeting online MBT. This provides the mechanism of efficient verification by conformance testing the hypothesis about system current mode predicted by the Markov model. To the best of authors' knowledge this is the first work to combine Markov models and UPTA for model checking and online MBT.

3 Online Model-Based Testing

The state-based explosion problem experienced by many model-based offline test generation methods is avoided by the online techniques because only a limited part of the state-space needs to be kept track of when a test is running. However, exhaustive planning would be difficult on-the-fly because of the limitations of computational resources at the time of test execution. Thus, developing a planning strategy for online testing should address the trade-off between reaction time and online planning depth to reach the practically feasible test cases.

The simplest approach to on-the-fly selection of test stimuli in online MBT is to apply so called random-walk strategy where no computation sequence of IUT has an advantage over the others. The test is performed usually to discover violations of input/output conformance relation IOCO [15] or timed input/output conformance relation TIOCO [7] between the IUT and its model. Random exploration of the state space may lead to test cases that are unreasonably long and nevertheless may leave the test purpose unachieved. On the other hand, such long test cases, when a test runs hours or even days, might help detect intricate bugs that do not fit under well-defined test coverage criteria.

In order to overcome the deficiencies of long lasting testing usually additional heuristics, e.g. "reactive planning tester" [17], are applied for guiding the exploration of the IUT state space. The extreme of guiding the selection of test stimuli is exhaustive planning by solving at each test execution step a full constraint system set by the test purpose and test planning strategy possibly leading to the explosion of the state space. Therefore, model checking based approaches are used mostly in offline test generation.

Another option for reducing the online test generation and planning overhead is to partition the test models into smaller ones where the test cases do not need the full blown model coverage. Following this idea we define in this paper the test models by system availability modes so that each of the test models refines only one state (availability mode) of the abstract Markov model. By reducing the size of test models in this way we can apply simple random-walk strategy instead of those that require computationally expensive state space exploration.

4 Multi-Fragment Markov Models

Kharchenko et al. [13] developed a set of potential scenarios of reaction for verification of detected faults during execution of spacecraft software and defined a basic set of availability models. In order to assess the availability depending on such scenarios a set of corresponding single- [9] and multi-fragment [12] Markov models were developed. Some possible scenarios are as follows: SC1 - all possible types of verification are performed. Online verification and correction of faults are impossible during the flight; SC2 - all possible types of verification are performed. Online verification is impossible during the flight. Some correction of faults is possible and made only in case of detecting of fault; SC3 - all possible types of verification are performed. Correction of all faults is performed; SC4 - all possible types of verification are performed (in ground conditions and during the flight). Correction of detected faults is performed. Blocking of parts of functions and system degradation is possible.

For the work in this paper we are concerned with scenarios SC3 and SC4. The availability model corresponding to these scenarios is presented in Fig. 1. The model includes a set of states and transitions caused by failures and repairs of hardware and software and possibilities of failure detection and system repair. There is a tree-like graph of transitions; the type of transition depends on the type of event occurred: detecting of fault or start of verification process. The graph models for scenarios SC3 and SC4 are equal; scenario SC4 models additionally the degradation of system functions. Thus, the probability of transition into the state of verification (and detecting of fault) may be higher for scenario SC4. In Fig. 1 failure rate of both software and hardware is represented by λ.

The availability model for the Bonfire MPSoC inspired by this Markov-chain based availability model will be presented and discussed in Section 6.

5 Uppaal Probabilistic Timed Automata

Uppaal Timed Automata (UTA) [6] address behavioral and timing aspects of systems providing efficient data structures and algorithms for their representation and analysis through model checking. An Uppaal Timed Automaton (UTA) is given as the tuple $(L,\ E,\ V,\ CL,\ Init,\ Inv,\ T_L)$, where L is a finite set of locations, E is the set of edges defined by $E \subseteq L \times G(CL,V) \times Sync \times Act \times L$, where $G(CL,V)$ is the set of constraints in guards. $Sync$ is a set of synchronisation actions over channels. Act is a set of sequences of assignment actions

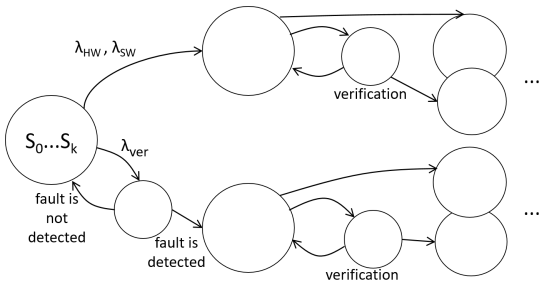

Fig. 1. Markov chain-based availability model with on-line verification.

with integer and boolean expressions as well as with clock resets. V denotes the set of integer and boolean variables. CL denotes the set of real-valued clocks ($CL \cap V = \varnothing$). $Init \subseteq Act$ is a set of assignments that assigns the initial values to variables and clocks. $Inv : L \rightarrow I(CL, V)$ is a function that assigns an invariant to each location, $I(CL, V)$ is the set of invariants over clocks CL and variables V. $T_L : L \rightarrow \{ordinary, urgent, committed\}$ is the function that assigns the type to each location of the automaton.

Uppaal Probabilistic Timed Automata (UPTA) [8] is a stochastic and statistical modeling extension of UTA. UPTA preserves the standard UTA input language such as integer variables, data structures and user-defined C-like functions. Additionally, UPTA automata support branching edges where weights can be added to give a distribution on discrete transitions. It is important to note that rates and weights may be general expressions that depend on the states and not just simple constants. For the work in this paper we use the branching edges with probability weights.

6 Availability MFMM and Mapping to UPTA

The Bonfire MPSoC employs a Network-on-Chip with a 2D mesh topology. Each router uses wormhole switching equipped with fault tolerant mechanisms. A processor is connected to each router via a network interface. The router comprises of an input buffer, a routing computation unit, a switch allocator granting requests to the same output based on Round-Robin policy and a crossbar switch to forward the packets. An abstract view of Bonfire is presented in Fig. 2. a. For this paper we are considering an instantiation of the platform with four processors and four routers. Since the routing mesh of Bonfire is the IUT the availability of which is studied in this paper, the test interface (denoted by dashed line in Fig 2. a) is defined by the interfaces between the network interfaces and routers, i.e., the processors constitute an environment for the IUT.

The availability model for Bonfire is shown in Fig. 2. b. Note that for simplicity we do not present a Multi-Fragment Markov model. Naturaly the multiple fragments expand to the right of the first fragment after software corrections. It

is assumed that the right-most fragment would not include any software faults. The single fragment model of Fig. 2. b is adequate to be used for the presentation of our integrated approach in this paper. The area within the dashed lines in Fig. 2. b corresponds to the part of the Markov model which will be mapped to UPTA for modeling of the mode switching and for stochastic model checking.

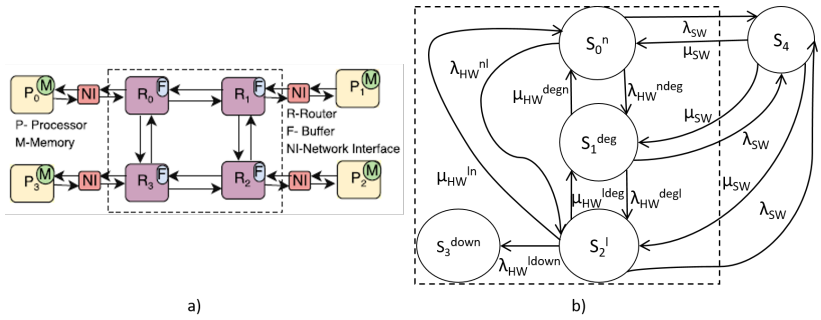

Fig. 2. a) Bonfire MPSoC, b) Availability Markov model for Bonfire MPSoC.

Initially the system implements all functions according to the specification and stays at the S_0, *normal* operational mode. Due to possible hardware (physical) faults the system can transit to modes S_1 and S_2 which model the *degraded* and *low* operational modes we have defined for this case study. State S_3 models the *System Down* mode where the system is not operable anymore. Due to possible recovery functions the system operation can be restored from other states. Moreover, in each operational mode, system failure might occur due to software (design) fault causing the system to transit to state S_4. This transition is executed when the process of online MBT (which is applicable in each operational mode) has detected a software fault. The online MBT process corresponds to the verification process shown in the lower part of Fig. 1. For clarity it is not shown explicitly in Fig. 2. b. We assume that the system software can be corrected by some means, e.g. software update (not shown in Fig. 2. b), and the system can then transit back to any of the operational modes where the fault occured.

The UPTA model corresponding to the Markov Bonfire model is presented in Fig. 3. a. The transitions of the Markov model in Fig. 2. b are mapped to probabilistic transitions (denoted by dotted line and labeled with probabilities) of UPTA shown in Fig. 3. a. The probabilities of UPTA are computed based on failure rates of Fig. 2. b. They specify the probabilities of events that cause transition from one mode to another. The probabilities of events that trigger transitions are normalized with respect to the model time unit denoted by *tu*. Combining the probabilities with explicit clock constraints in the model of Fig. 3.a is needed for SMC of timing properties.

The UPTA automata that interface the abstract availability model in Fig. 3.a with detailed mode behavior models are shown in Fig. 3.b. Initially all automata

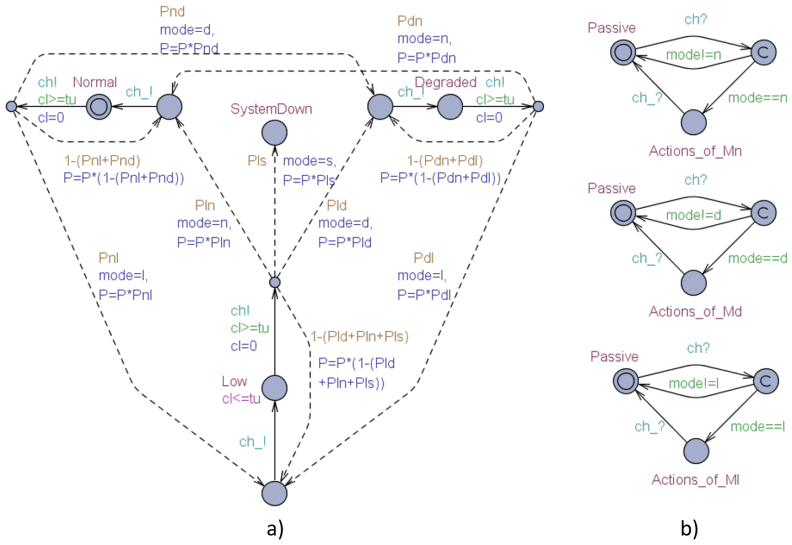

Fig. 3. a) Mode UPTA for Bonfire MPSoC based on the Markov availability model, b) Mode adapter automata for the different operational behaviors.

are in location *Passive* waiting for synchronisation from the Mode switching automaton. Upon synchronisation they will either return back to the *Passive* location or the relevant internal behavior of a mode will be triggered. Locations *Actions_of_Mn*, *Actions_of_Md* and *Actions_of_Ml* represent respectively each mode which are elaborated in the next subsections.

7 UPTA Models of the Bonfire MPSoC

We have created the models of the main components of the Bonfire MPSoC platform, namely, the processor and the router considering an instantiation of the platform with four processors and four routers. These models can be used for online MBT for each different operational mode of the system. The processor component model comprises the behavioral view that is assumed to be unchanging throughout the modes and the router component models comprise the behaviors in normal, degraded and low mode that represent respectively functioning under ideal conditions, the reliability and security views. Here by views we mean *design views* which should be consistent with each other to ensure the soundness of design.

The reliability view concerns the probability of data packet transmission failure on any link between processors and routers as well as between the routers. The security view concerns the possibility of cyber attack to some processor after which the attack should be detected by the traffic pattern generated by this processor and the attacked processor should be blocked. Note that the timing

view is implicitly conjoined with the aforementioned views due to the modeling formalism. Thus, in this paper the behavioral view of Bonfire relates to the normal operational mode of the system, the reliability view relates to the degraded mode with all processors still operating and the security view relates to the low mode. If more than one processors are blocked then the system is shut down.

7.1 Behavioral view

The UPTA templates representing the behavioral view of the processors and routers are shown in Fig. 4 a and Fig. 4 b, respectively. The environment to the routing system consists of four processors which are instantiated with four processes, each sending and receiving frames to and from the routing system. The routing system is modeled as UPTA processes parameterized with source and destination router id-s respectively. The router template specifies (i) the case where there is no frame in its input queue (the edge from $Wait$ location down) and (ii) the case where there is at least one frame in the input queue to be forwarded to the next router (the edge from $Wait$ location to right and its subsequent edges and locations reachable before return to location $Wait$).

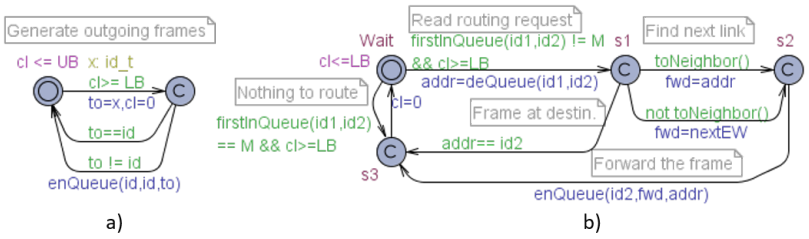

Fig. 4. a) UPTA for normal behavior view of a) Processor and b) Router link.

The composed model of the behavioral view represents design assumptions A1 to A10 and guarantees G1 to G3 of routing as follows. A1 - Processors are indexed clock-wise from 0 to 3 in the mesh and denoted respectively with P_i, where $i : [0, 3]$. The same holds for local routers R_i; A2 - Processors communicate by sending and receiving frames; A3 - Any processor P_i can send at most one frame per tu to its local router R_i; A4 - Any pair of neighbor routers has direct communication link in two directions, a $link(i, j)$ denotes a link from R_i to R_j and $link(j, i)$ a link from R_j to R_i, where each link $link(i, j)$ has a buffer $buf(i, j)$ to store the frame postponed due to congestion; A5 - An oldest element in the buffer $buf(i, j)$ is forwarded along the link $link(i, j)$; A6 - Buffer emptiness is checked and content sent once at any tu; A7 - A frame is written by router R_i to buffer $buf(i, j)$ of its outgoing link $link(i, j)$ during the same tu when it is read from some of its incoming link $link(k, i)$; A8 - Congestion rises when the frames of different incoming links $link(i, j)$ and $link(k, j)$ of router R_j arriving

in the same tu need to be forwarded along the same outgoing link $link(j, l)$; A9 - The congestion can rise at most between two incoming links in one tu; A10 - The frames are identified by their source and destination processor id-s.

The explicit design *guarantees* for the processors and routers are as follows. G1 - Any frame sent from a processor P_i does not have congestion on any link with some other frame sent from the same processor P_i; G2 - The frames once routed along $link(i, j)$ are never routed backwards along $link(j, i)$; G3 - In the routing system satisfying assumptions A1 to A10 any frame can be routed to its destination with at most three hops, i.e. one along local link and two along links between routers while the congestion can occur in inter router link only. We additionally assume that the frames are never sent to the processor itself and reading a frame by processor from the routing system never blocks.

Each router in normal mode implements the design assumptions A4−A7 and provides the guarantees G2, G3 and ensures that no package loss occurs.

7.2 Reliability view

The UPTA template for the reliability view of the system is presented in the upper part of Fig. 5. While the processor template remains the same, the routing template for the reliability view extends the behavioral view assumptions with the assumptions about the correct delivery of the frame over the link between routers is p and frame transmission failure with probability $1-p$. The refinement of the edge with label "Forward the frame" with two alternative outcomes - one for successful and the other for failure of transmission between a pair of routers introduces probabilities p and $1-p$ respectively of these options. The guarantees G2 and G3 still hold. The probability of packet delivery with two hops and with the same upper time bound as in behavioral view is p^2.

7.3 Security view

The UPTA model template for the security view of the system is presented in the lower part of Fig. 5. The assumptions of the security view maintain the assumptions of the behavioral view regarding writing and reading frames to and from processors. The assumptions are extended with the possibility of cyber attack resulting in the corruption of processor code in one of the processors $P_0 - P_3$. We have to introduce a new guarantee G4 which ensures that the detection of the attack occurs at latest during r tu after corrupted code is activated in a processor P_x. The detection is implemented by monitoring the traffic between a processor P_x and its local router R_x, i.e the frames sent over $link(x, x)$ (see edge $< s1\_1, s1\_2 >$ in the template).

Another guarantee related to attacks is G5 which ensures that the processor P_x under attack (with corrupted code) should not interfere with the functioning of the routing system including R_x. This is implemented as follows: when the number of consecutive sends from processor P_x to some processor P_j exceeds a threshold value r then Boolean variable *attack* is assigned the value *true* causing the link $link(x, x)$ from P_x to R_x to be blocked for further communication with

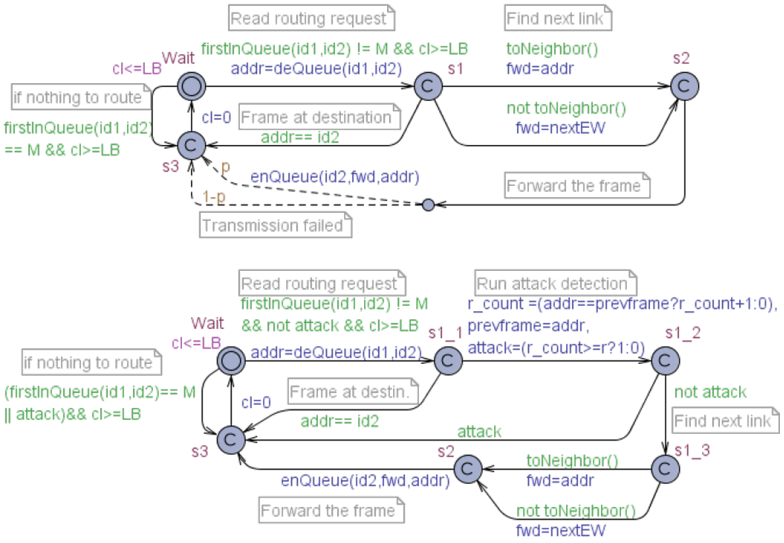

Fig. 5. UPTA for Router link reliability view with probability for frame loss (upper) and security view with modeling of attack and blocking of processors (lower).

P_x (edge $< Wait, s1\_1 >$ is not enabled). Note that router R_x can still forward packages from other routers.

7.4 Correctness properties

Proving the correctness of the model of the IUT means that the guarantees of the routing system design need to be verified provided the assumptions A1 to A10 are properly implemented and represented in the design models as of Fig 4 to Fig 5. In the following we introduce some examples of integral QoS related assume-guarantee properties that rely on above mentioned design assumptions and guarantees.

For the behavioral view, property **i)** states that any frame sent from processor P_i to P_j for $i, j = 0, ..., M-1$ where $i \neq j$, will reach its destination at most with t_k time units. Property **ii)** states that under given environment assumptions (including the contention mode) the length of message queue $buf(i, j)$ at any $link(i, j)$ does not exceed the buffer size BS.

For the reliability view, property **i)** states that any frame sent from processor P_i to P_j for $i, j = 0, ..., M-1$ where $i \neq j$, will reach its destination during k time units with probability p_k. Property **ii)** states that under given environment assumptions (message loss rate) the length of message queue $buf(i, j)$ at any link (i, j) does not exceed the buffer size BS with probability p_l.

For the security view, property **i)** states that the maximum time of cyber attack detection and mitigation (processor blocking) has upper bound. Property

ii) states that in case of attack the maximum package delivery time of remaining routing system does not drop below t_k time units.

7.5 Verification of Router's Multi-View Operational Modes

Let us elaborate on the verification of the interrelated properties regarding the sending of data frames to their destination address of the views that could interfere. Similar reasoning holds for the verification of the interrelated properties regarding the buffer size for the behavioral and reliability views.

For verifying property **i)** of the behavioral view we construct a property recognizing automaton by applying a "Stopwatch" UPTA template (see Fig. 6) and compose it with the behavioral model by synchronizing Stopwatch start action with frame send action from processor P_i to router R_i and stop action with receive action when the frame is sent to link $R_j - P_j$. There are several verification assumptions as follows. Since the routing architecture is North-South and East-West symmetric, the maximum transport delay can be measured from any processor P_i to processor P_j where $i \neq j$ and P_i and P_j are in the mesh opposite diagonal corners. Verification of P_i to P_j transport delay can be done w.r.t. any generated single frame separately without considering the full flow from same P_i because of G1. Frame generation time instant T_gen at P_i can vary from 0 to load stabilization time in the routing system. Time out T_out, i.e the upper time bound we check can be 6 since due to the topology maximum 2 hops should be performed to reach the destination and each hop may have one congestion at most due to G2 (to one link i, j, where $i \neq j$ only P_i and R_i can compete). To identify the receive event for model checking end-to-end transfer time we have to encode also the sender address P_i in the traceable frame. So we use encoding of frames as a pair of constants where the first corresponds to source address and the second to destination address. AB means P_0 sends to P_1, DB means P_3 sends to P_1, etc.

Stopwatch automaton is used for verifying the guarantee that the end-to-end delay upper bound is T_out. This is proven by model checking $M_B \models \mathsf{A}\Diamond Stopwatch.Pass$.

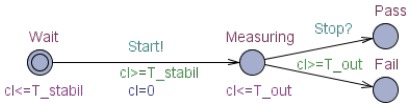

Fig. 6. "Stopwatch" Property recognizing automaton for verifying property (i) of behavioral view.

In addition to the verification scheme for property **i)** of the behavioral view, property **i)** of the reliability view for successful end-to-end delivery needs to be calculated by adding an update: $p\_route = p\_route \times p_{ij}$, to each router process transition labeled with "Forward the frame" where p_{ij} denotes the probability

of successful transmission over $link(i, j)$ provided the link is on the route. Under the assumptions of the given view the validity of the guarantee of the property is verified by checking $M_{Rel} \models \mathsf{A}\Diamond Stopwatch.Pass \;\&\&\; p\_route >= p\_k$.

Similarly to verifying property **i)** of the behavioral view, the model checking task for verifying interrelated property **ii)** of the security view is $M_{Sec} \models \mathsf{A}\Diamond Stopwatch.Pass$.

8 Online MBT of Bonfire MPSoC Execution Modes

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. For UPTA there exists the online MBT tool Uppaal Tron [3]. The UPTA models developed in Section 7 for conformance testing of Bonfire MPSoC modes consist of IUT which is the routing system and the environment of IUT which consists of processors. Interactions between the environment and IUT constitute the symbolic representation of the test run against which the conformance of real I/O sequences are checked. Since we have a different IUT model for each mode the test cases concern only those models that correspond to the mode. As mentioned before the models of environment, i.e. processors are the same in all test cases.

The overall test setup used in the context of MBT with Uppaal Tron as the test engine and dTron [1] as the adapter generation framework is given in Fig. 7. The test model represents symbolically the interaction between IUT and its environment components. The test adapter is responsible for translating symbolic I/O of the test model to frames passed to the routing system and mapping the frames received at the destination processor back to model symbolic output. The dTron layer allows the adapters to be distributed over the routing system ports while ensuring that measuring the model time is in synchrony with physical clocks at I/O ports.

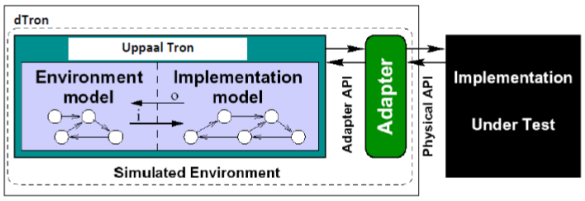

Fig. 7. The test setup involving the Uppaal Tron test engine and the distributed adapter library dTron.

The test configuration used in the current work consists of test execution environment dTron and one test adapter per processor-router link that transforms abstract input/output symbols of the model to input/output data of the

IUT. The setup is outlined in Fig. 7. Uppaal Tron is used as a primary test execution engine which simulates symbolic interactions between the IUT and its environment. The interactions are monitored during model execution. When the environment (one of the processors in the model) initiates an input action i Tron triggers input data generation in the adapter and the actual test data is written to the IUT (implemented routers) interface. In response to that, the receive event at destination processor port produces output data that is transformed back to symbolic output o. Thereafter, the equivalence between the real output returned and the output o specified in the model is checked. The run continues if there is no conformance violation, i.e. there exists an enabled transition in the model with parameters equivalent to those passed by the IUT. In addition to input/output conformance, the real-time input/output conformance is checked by Uppaal Tron.

9 Conclusions

In this paper we have addressed the online monitoring and model-based validation techniques for mission critical systems. We propose an integrated approach where the MFMM are used for specifying the system reliability and security related behavior on high-level of abstraction relevant for Markov analysis, and the more concrete state and timing constraints related with MFMM are specified explicitly using UPTA for stochastic modeling and model-based test generation. To interrelate these two model classes we have shown the mapping from MFMM to UPTA. The second contribution is the test case selection mechanism for online testing where the test cases are prioritized by the probabilities of modes. The hypotheses on which mode the system switches next are provided by MFMM and the hypothesis are tested by conformance tests using UPTA models that specify the mode behavior. The approach is illustrated with the Bonfire MPSoC where the probabilistic model checking of high-level mode switching scenarios is applied on MFMM. A set of UPTA models that refines the behavior of MFMM modes is applied thereafter as a test suite for model-based conformance testing to confirm the hypothesis of being in the given mode.

As future work, more practical experiments are planned for quantitative analysis to estimate how much the probability guided online testing shortens the time of identifying system availability mode in time-critical applications.

Acknowledgements This research was supported by the Estonian Ministry of Education and Research institutional research grant no. IUT33-13 and in part by projects H2020 RIA IMMORTAL, H2020 TWINN TUTORIAL and by European Union through the European Structural and Regional Development Funds.

References

1. dTron, https://cs.ttu.ee/dtron/

2. ECSS-E-40-05. Software Verification, Validation and Testing (2005)
3. Uppaal Tron, http://people.cs.aau.dk/~marius/tron/
4. Bonfire project wiki (2017), https://github.com/Project-Bonfire/Bonfire/wiki
5. Aupperle, B., Meyer, J.: State space generation for degradable multiprocessor systems. In: Fault-Tolerant Computing, 1991. FTCS-21. Digest of Papers., Twenty-First International Symposium. IEEE (1991)
6. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–237. Springer Verlag (2004)
7. Brinksma, E., Tretmans, J.: Testing transition systems: An annotated bibliography. In: Cassez, F., Jard, C., Rozoy, B., Ryan, M.D. (eds.) Modeling and Verification of Parallel Processes: 4th Summer School, MOVEP 2000, France, 2000 Revised Tutorial Lectures. pp. 187–195. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
8. Bulychev, P.E., et al.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: Wiklicky, H., Massink, M. (eds.) Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems (2012)
9. Guimaraes, A.P., Oliveira, H.M.N., Barros, R., Maciel, P.R.: Availability analysis of redundant computer networks: A strategy based on reliability importance. In: IEEE 3rd International Conference on Communication Software and Networks (ICCSN), 2011. IEEE (2011)
10. Kashyap, A., Holzer, T., Sarkani, S., Eveleigh, T.: Model based testing for software systems: An application of markov modulated markov process. International Journal of Computer Applications 46(14), 13–20 (May 2012)
11. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. Springer-Verlag New York (1976)
12. Kharchenko, V., Odarushchenko, O., Odarushchenko, V., Popov, P.: Selecting mathematical software for dependability assessment of computer systems described by stiff markov chains. In: Proc. 9th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer (ICTERI 2013). vol. 1000, pp. 146–162. CEUR-WS (2013)
13. Kharchenko, V., Ponochovnyi, Y., Boyarchuk, A.: Availability Assessment of Information and Control Systems with Online Software Update and Verification. Springer International Publishing (2014)
14. Siegl, S., Dulz, W., German, R., Kiffe, G.: Model-driven testing based on markov chain usage models in the automotive domain. In: Waeselynck, H. (ed.) 12th European Workshop on Dependable Computing, EWDC 2009, Toulouse, France (2009)
15. Tretmans, J.: Testing concurrent systems: A formal approach. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR'99 Concurrency Theory: 10th International Conference Eindhoven, The Netherlands, 1999 Proceedings. pp. 46–65. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
16. Trivedi, K.S., Kim, D.S., Roy, A., Medhi, D.: Dependability and security models. In: Proceedings of 7th International Workshop on Design of Reliable Communication Networks, 2009. DRCN 2009. IEEE (2009)
17. Vain, J., Kaaramees, M., Markvardt, M.: Online testing of nondeterministic systems with the reactive planning tester. In: Petre, L., Sere, K., Troubitsyna, E. (eds.) Dependability and Computer Engineering: Concepts for Software-Intensive Systems. pp. 113–150. Hershey, PA: IGI Global (2012)
18. Veanes, M., et al.: Formal methods and testing. chap. Model-based Testing of Object-oriented Reactive Systems with Spec Explorer, pp. 39–76. Springer-Verlag, Berlin, Heidelberg (2008)
19. Whittaker, J.A., Thomason, M.G.: A markov chain model for statistical software testing. IEEE Trans. Softw. Eng. 20(10), 812–824 (Oct 1994)