Mobile Networks and Applications (2023) 28:732–743
https://doi.org/10.1007/s11036-019-01369-6

Improving Formal Verification and Testing Techniques for Internet of Things and Smart Cities

Moez Krichen[1,2]

Published online: 6 September 2019

Abstract
We are interested in formal verification and model-based testing for Internet of Things and Smart Cities. In general these two techniques suffer from state explosion problem. To remedy this situation we propose a set of techniques which aim to reduce the cost, duration and complexity of the considered problems. On the first hand the techniques realted to formal verification are as follows. First, Abstraction consists in modelling a part of the system accurately and the other parts at high level. Second, Modularization and Compositionality consist in splitting the whole system into smaller subsystems. Third, Symmetry Detection exploits symmetries that take place during the system execution. Fourth, Data Independence consists in detecting that the behaviour of the considered system does not depend on some data inputs. Fifth, Eliminating Functional Dependencies consists in removing dependency among state variables. Sixth, Exploiting Reversible Rules consists in collapsing subgraphs of the graph of states into abstract states. On the second hand the techniques related to model-based testing are as follows. First, Refinement Techniques extract test scenarios directly from the untimed specification. Second, the Reduction of the Size of Digital-Clock Tests Technique provides a heuristic to reduce the size of the generated tests. Third, the Timed Automata Testers Generation Technique allows to produce testers in the form of deterministic timed automata. Fourth, the Test Cases Updating Technique after System Evolution makes it possible to reduce the number of tests to be generated after each adaptation. Fifth, the Resource Aware Test Component Placement Technique allows to produce a placement plan of the different testers. Sixth, Coverage Technique generates a reasonable-size set of tests. A case study is proposed in order to illustrate the use of these techniques.

Keywords Formal · Verification · Model-based · Testing · Internet of things · Smart cities · State explosion · Optimization

1 Introduction

The Internet of Things (IoT) [2, 3, 40] has now become a key technology that can span multiple technology areas, from data discovery and processing to networking and data analysis. It is used in many applications ranging from home security and factory automation to health care delivery [4, 37, 41] and self-driving [1, 8, 9, 49]. The Internet of Things offers many benefits and applications to smart cities, including: improvement of traditional public services such as transport, traffic and parking; control and maintenance of public spaces; monitoring the validity of buildings and workplaces; reducing time lost in administrative transactions; monitoring and control of energy consumption; smart lighting for the city. In addition, the amount of data collected by connected objects allow citizens to better understand the state and evolution of the city. As the number of connected objects increases, so does the number of potential problems. For this we need to adopt adequate and effective test techniques. Among the types of possible tests we mention:

- Hardware Tests: This type of tests consists in verifying that the device is working properly and that it it is compatible with other software and devices.
- Functional tests: In the face of the complexity of connected objects, it is vital to ensure that the different devices work as expected on any type of environment.
- Security tests: Within the Internet of Things ecosystem, huge volumes of data are accessible by users. Verifying

✉ Moez Krichen
moez.krichen@redcad.org

[1] Faculty of CSIT, Al-Baha University, Al-Baha, Saudi Arabia

[2] ReDCAD Laboratory, University of Sfax, Sfax, Tunisia

data privacy and user authentication is a key element of IoT solution testing.

In this work[1] we are interested in the so-called *formal verification and model-based testing* (MBT). The latters can be considered as a specific branch of *formal methods*. Formal methods are alternative, mathematically based techniques for system specification and development, as well as automatic property checking. A range of formal languages and techniques exist to handle different types of properties at different levels of system development. More precisely the MBT approach consists in automatically extracting test cases from the formal model of the considered system under test (SUT). In general Formal Verification and MBT suffer from the well known *state explosion problem* which corresponds to the fact that formal verification and test generation may require a huge amount of time and a very large space to compute and store generated test cases. The situation becomes even more critical when dealing with large and sophisticated distributed systems like IoT Systems and smart cities. To remedy this situation we propose in this article to use a set of techniques taken from the literature and some of previous works and which aim to reduce the cost, duration and complexity of the generated tests. On the first hand, the techniques related to formal verification are as follows:

- Abstraction: consists in abstracting details of the system to be verified which are not relevant to the properties of interest.
- Modularization and Compositionality: consist in decomposing the verification of large systems into relatively smaller subproblems of reasonable complexity.
- Symmetry Detection: exploits symmetries that take place during the system execution, in order to minimize the corresponding state space.
- Data Independence Detection: corresponds to the situation when the designer of the system to be verified detects that the behaviour of the considered system does not depend on some data inputs.
- Eliminating Functional Dependencies: consists in removing functional dependency among state variables.
- Exploiting Reversible Rules: consists in collapsing subgraphs of the graph of states into abstract states.

On the second hand, the techniques related to MBT are as follows:

- Refinement Techniques: instead of transforming an untimed specification into a timed specification and then extracting timed test cases from it, this technique allows to extract the test scenarios directly from the untimed specification and then transform them into timed tests; thus reducing the cost of test generation in a very remarkable way.
- Reducing the Size of Digital-Clock Tests: when the size of the generated tests is large this technique provides a heuristic to reduce the size of these tests by compacting the corresponding graphs.
- Generating Timed Automata Testers: consists in generating testers in the form of deterministic timed automata with resources fixed in advance (e.g., number of clocks).
- Updating Test Cases after System Evolution: for the case of dynamic and evolutive systems this technique makes it possible to reduce the number of tests to be generated after each adaptation of the system under test by detecting all the old tests which are still valid and can be reexecuted without any modification.
- Resource Aware Test Component Placement: allows to produce a placement plan of the different testers on the nodes of the system under test that takes into account available resources such as RAM, CPU and battery.
- Coverage Techniques: consist in generating a reasonable-size set of tests according to specific selection criteria to cover particular elements of the model of the system under test (e.g., nodes, states, transitions, etc.).

The rest of this article is organized as follows. Section 2 provides a brief overview about the Internet of Things, smart cities, formal methods and model-based testing. Section 3 defines the model of timed automata adopted in this work, the untimed and timed conformance relations and the different types of tests adopted. Section 4 lists the main techniques used for improving formal verification procedures. Section 5 gives more details about the different techniques used to improve the test generation procedure. In Section 6, we propose a simple case study in order to illustrate the use of the different proposed techniques. Finally Section 7 gives a conclusion for the paper and presents some perspectives for future work.

2 Preliminaries

2.1 Internet of things and smart cities

Internet of Things The International Telecommunication Union defines the Internet of Things (IoT) to be a "global infrastructure for the information society, which provides advanced services by interconnecting objects (physical or virtual) with the technologies of the Internet. existing and evolving interoperable information and communication".

Smart cities The term"smart city" was born in the 1990s. This expression is primarily the result of a strategy of

[1]This article is an extension of our previous work [27] presented in SCITA 2017 [38]

reconquest of the market set up by the firm IBM. Wishing to raise profits in a period of recession, the firm has indeed identified cities as a huge potential market, associating them with information and communication technologies [51]. Anchored in a context of urban marketing and publicity, the term "smart city" is also an expression, which among many others, is used to define the city of the future. The scientific literature agrees on this aspect: the definitions vary according to the context and there is no consensual definition. The authors of [5] identified 23 different definitions. This variety can be explained in particular by the fact that the smart city, because of the diversity of the fields it touches, is an object of multidisciplinary research. However, there is a common assumption for all these different meanings: the smart city is a data-driven city. Indeed the intelligent transformation of cities caused by the emergence of new technologies has progressively integrated several aspects of urban life (e.g., economy, education, infrastructure, transport, environment, safety and quality of life).

2.2 Formal methods

When developing a computer system, the exhaustive detection of design errors remains difficult in the context of simple functional tests or manual verification procedures. Thus, in the early 1980s, researchers began to make computer systems verification methods more rigorous, particularly by automating them [16, 47]. Indeed, with the appearance of mathematical models for the specification of dynamic systems [31], the first formal systems verification approaches have emerged. We can distinguish two categories of computerized methods of formal verification: the automated theorem proving and model checking.

The automated theorem proving stems from Frege's prediction theory, which consists in demonstrating theorems, expressed in the first-order logic, thanks to a certain number of axioms and rules of inference. From the point of view of system verification, it is then, from a set of formulas and a property, expressed in the form of a theorem, to prove that the system respects the property, by calculating a proof tree for the theorem. However, although demonstrated as complete by Gödel in 1929, the calculation of the first-order predicates was also proved to be undecidable by Church in 1936 and by Turing in 1937. The theorem proof is, in fact, not fully automatable for the logic of the first order (propositional calculus being decidable) and may require human interaction. There are several proof assistants, such as HOL [18], Isabelle [45] and Coq [12], that allow computer-assisted verification for systems expressed in the logic of predicates, but other methods are more suited to the profile. dynamic systems. In particular, the methods Z, B and Event-B for example, make it possible to model the transitions of a system more easily thanks to an abstract machine model.

Model checking theory, independently proposed by Clarke and Emerson in 1981 [16] as well as by Queille and Sifakis in 1982 [47], allows, given the formal specifications of a system and a property, to automate the verification process under certain conditions. The principle of verification by model checking is to algorithmically traverse the complete state space of a system to validate a certain property. When the state space of a system is isomorphic to a finite automaton, it is easy to determine a path algorithm and thus to verify certain properties. The verification procedure is then fully automatable. However, the number of states of a system can often be very large, or even infinite in some cases, which is the main disadvantage of model checking. Indeed, the algorithmic complexity of the verification procedure depends on the size of the system as well as the shape of the property to be verified. Moreover, in some cases of infinite systems, the problem becomes undecidable.

2.3 Model based testing

Model-Based Testing (MBT) or generation of model-based tests is an approach where the system is represented as an abstract model that represents the behaviors of the system (all or in part). This approach consists of reasoning on this speciation model to automatically calculate abstract sequences (because of the same level of detail as the model) of stimuli that constitute the tests. These sequences of stimuli are then concretized and then performed on the (real) system under test. The verdict is given by comparing the results obtained on the system under test with the results produced by the model. The model thus constitutes the test oracle because it makes it possible to predict the expected results on the target system [43, 55] . It represents a dynamic view and clarifies the initial state of the system by formalizing system behavior and capturing the control and observation points of the system under test. The model must be sufficiently precise and formal to allow unambiguous interpretation to have a reproducible automatic generation [60].

The techniques of generation of tests from models have known for several years a considerable interest and growth. This attractiveness, on the part of the scientific and industrial circles, is explained in particular by the ever increasing complexity of computerized systems and the current need to guarantee a quality of service and optimum reliability. Indeed, generating tests from specification models ensures a high level of confidence in the respect of the requirements set out in the specifications. Application loads to be tested through the use of coverage criteria. This type of technique begins to find its place because of its

partial or complete automation made possible by the ability to reason on formal specification models. Finally, the use of the model can be initiated very early during the development cycle, allowing a system-level follow-up and validation of what is a point strong in a field where the development of a system can take several years.

Despite the many results, the field is still very active to meet the new requirements of the market: increasingly strict standardization, increased quality of service expected, willingness to control costs and of course, the arrival of new systems. more and more complex such as modern embedded systems. This is why guaranteeing the quality of a system still represents a dynamic sector for research and innovation, as evidenced by the large number of conferences and journals specializing in this area.

In addition to the problem of capturing system behaviors through a model, the MBT presents several problems in the actual test generation process. Indeed, the generation of tests must be relevant, that is to say to obtain tests ensuring a certain level of quality, while being controlled to avoid the combinatorial explosion.

3 Formal framework

3.1 Timed automata

Timed Automata (TA) [6, 13, 50] are a simple and expressive formalism for modeling computer systems that combine discrete and continuous mechanisms. These structures take the form of finite oriented graphs extended by a certain number of clocks, represented by real variables whose value evolves continuously with time.

3.2 Different types of tests

A given test can be seen as an interactive strategy between the tester and the system under test. The Tester sends inputs to the SUT and receives outputs from it and then checks whether the generated outputs are correct or not and produces either Pass or Fail verdicts correspondingly. Depending on the capabilities of the tester it is possible to distinguish three types of tests:

– Timed Automata Testers: In this case the tester itself is presented as a timed automaton. The latter is built in advance before test execution. It is deterministic in the sense that a given action leads to unique known state. However the construction of such a tester is not always possible since it is not possible to transform all non-deterministic timed[–] automata into deterministic ones.
– On-the-Fly Analog-Clock Testers: This type of tests have the capability to measure time with high precision using analog clocks. They are generated in parallel with test execution and can not be stored in memory since they are infinite.
– Digital-Clock Testers: These tests use digital-clocks to measure time. They can be generated either before or during test execution. They can be represented and stored as finite trees.

3.3 Conformance relations

In model based testing it is necessary to define a particular relation called *conformance relation* which allows to compare a given specification with its possible implementations. In general we assume that both the specification and the implementations are described using the same formalism. More precisely we assume that the model of the specification is known. However we assume that the model of every possible implementation exists but it is unknown. For the case of untimed systems we may use the *input-output conformance relation* ioco [54]. The latter allows to compare two input-output transition systems $Spec$ and Imp. Imp conforms to $Spec$ with respect to ioco if it accepts more inputs and generates less outputs than $Spec$. For the case of timed systems we may adopt the *timed input-output conformance relation* tioco [29, 30].

4 Different techniques for improving formal verification activities

4.1 Abstraction

As already mentioned formal verification is always limited by the state explosion problem. Fortunately, many properties of the considered system may only depend upon a small part of the system which needs to be accurately modeled while the other parts may be modelled at a very high level. In this manner abstraction plays a very important role in the verification process of complex systems. The abstraction procedure may be achieved either manually or automatically. Obviously manual abstraction is hard to make in general and is error-prone. Consequently, it is much better to rely on automatic abstraction whenever it is possible.

For instance in [53], the authors were interested in the formal verification of cyber-physical systems. These systems are very complex since they need to represent the physical environment, hardware and software. The paper presented an automatic abstraction procedure for simplifying the Labeled Hybrid Petri Nets (LHPN) models. The main

idea consists in applying LHPN transformations to eliminate details that are irrelevant to the aspects of interest. The considered transformations are inspired by similar transformations applied to ordinary Petri nets and by various static analysis techniques used during compilation. Similarly in [7] the authors proposed a number of languages in order to model hardware systems. They abstracted away wide datapaths and they kept the low-level details of the control logic. This results in an important reduction in the size of the space of states and makes the formal verification of the considered system possible. The proposed automatic abstraction in this work was achieved using the Vapor tool. Likewise in [56] the authors aimed to accelerate the verification process of a specific type of microprocessors. That was done by automatically applying a set of transformation rules for abstracting the register file. In [58] the authors presented the tearing paradigm as a methodology for automatically abstracting the behavior of a reactive system. Moreover they proposed algorithms which perform conservative formal verification.

4.2 Modularization and compositionality

An other way to deal with state explosion problem while achieving formal verification consists in adopting modularization and compositionality. These two techniques consist in decomposing the verification of large systems into relatively smaller subproblems of reasonable complexity. For instance the work of [59] concentrated on Cyber-physical systems. In this paper the authors separated functional aspects from adaptive aspects. Moreover the adopted a modular formal verification approach. In [14], the authors presented a methodology for the verification of C programs using finite state machines. In [25] the authors reviewed compositionality issues related to linear temporal logic (LTL). Moreover a technique for reducing infinite systems into a finite systems which can then be checked using formal verification techniques.

4.3 Symmetry detection

Symmetry detection [57] corresponds to an other technique which may be used to counter the state explosion problem in formal verification. This technique exploits symmetries that take place during the system execution, in order to minimize the corresponding state space. It consists in computing a mapping from states to representatives of equivalence classes. In [32] the authors proposed an approach based on symmetry reduction techniques related to probabilistic formal verification. In [17], the authors proposed an efficient method that computes representatives dynamically during fixpoint calculus. The authors of [21] presented a framework for defining and evaluating some symmetry reductions used in software formal verification. Furthermore, in [46] the authors exploited structural symmetries in the the system they are aiming to verify. For that purpose they introduced new elements to their description language. Then a verifier can automatically produce a reduced state space.

4.4 Data independence detection

Data independence detection [10] is an other technique that can be used for simplifying formal verification as well. This technique corresponds to the situation when the designer of the system to be verified detects that the behaviour of the considered system does not depend on some data inputs. In this case the model of the system can be simplified significantly. In [39] the authors considered Alloy which is an extension of the first-order logic. They proposed a theorem which calculates a threshold which may be used for the analysis of data-independent systems. In [48] the authors were interested in verifying specific protocols. For that purpose they considered data independence aspects in order to restrict the number of generated states of the system to be verified.

4.5 Eliminating functional dependencies

Many researchers have reported that the use of BDDS (Boolean Decision Diagrams) remarkably increases the size of the models to be formally verified. Moreover the presence of functional dependency among the state variables was classified as a common cause of inefficient BDD encoding in formal verification. Consequently removing such dependency reduces considerably the space of states. In [15] functional dependency is identified using SAT solving and Craig interpolation techniques. In [24] functional dependency is directly detected from the transition functions rather instead of being detected from reached state sets. In [20] the authors were interested in verifying hardware design.They identified functionally dependent variables in order to reduce the size of the considered BDDs.

4.6 Exploiting reversible rules

This technique [22, 23] consists in collapsing subgraphs of the graph of states into abstract states (called progenitors of the subgraphs). This task is achieved by identifying generation rules which can be inverted. These rules are called reversible rules in the sense that a subgraph can be produced from the progenitor via these rules, and each state can be associated with the progenitor by executing the considered reversible rules. Compared to classical abstraction procedures, this technique does not require the user to propose an appropriate abstract domain by his own.

5 Different techniques for improving testing activities

5.1 Refinement techniques

Refinement Approaches are well experimented in the field of hierarchical design of many types of systems. They consist in transforming high-level actions into lower-level actions. That is to start with a high-level abstraction and to move to a lower-level one until obtaining the implementation level. Adopting action refinement approaches in the field of testing seems to be very promising. In our previous work [11] we proposed an refinement based approach for testing real times systems. We assume that the system under test can be initially described using an untimed specification. For instance it can be modelled using a simple automaton which has no clocks and no time constraints. This untimed specification is then transformed into a timed one using refinement techniques.

5.2 Reducing the size of digital-clock tests

Digital-clock tests can grow very large because they may sometimes contain big numbers of consecutive tick actions. In order to reduce of these tests we may extend test cases with variables and more sophisticated data structures as proposed in our previous work [30].

5.3 Generating timed automata testers

Representing analog-clock test cases as deterministic timed automata allows to save a lot of computation time during computation time. However this alternative is not always possible since theoretically it is not possible to transform all non deterministic timed automata into deterministic ones using a finite number of clocks and nodes. To alleviate this problem we proposed in a previous contribution [30] a deterministic over-approximation of the considered non deterministic timed automaton. This approximation is obtained by fixing in advance the available resources to use and the adopted strategy for resetting clocks.

5.4 Updating test cases after system evolution

This technique is taken from our previous works [34]. It consists in optimizing the test generation phase after the occurrence of dynamic adaptation of the system under test. We assume that the model may of the system under test may evolve either partially or entirely after a dynamic behavioral adaptation occurs. Consequently, we need to update the set of available test scenarios either by generating new tests or updating old ones. For this reason we need to take advantage from selective regression testing and MBT techniques.

In order to reach that goal, we follow a four-step approach:

- Model Differentiation: This step aims to compare the initial and new models of the SUT. That is to detect differences and similarities between these two models. This allows to avoid creating the whole set of test cases from scratch after every behavioral adaptation of the SUT.
- Old Tests Classification: This step consists in deviding the set of old available test cases into three subsets of tests: partially valid tests; still valid tests; no longer valid tests.
- Test Update and Generation: This step allows to to update partially valid tests and to derive new test cases from the newly identified behaviors. To achieve this goal we may use techniques borrowed from MBT approaches [26, 30].
- TTCN-3 Transformation: This step consists deriving a set TTCN-3 tests from the set of abstract test cases produced during the previous steps [33].

5.5 Resource aware test component placement

This technique is inspired by our previous contributions [35, 36]. It consists in distributing the test components on the various nodes of the system under test while taking into account the available resources of these nodes. For this purpose, we need to monitor resources during the execution of the system: free memory, CPU load and battery state. Then we model the problem we have in hand as an instance of the *knapsack problem*. The latter corresponds to a classic application of integer programming. In the problems of the knapsack, there is a container (the “knapsack”) with a fixed capacity and a number of elements. Each element is associated with a weight and a value . The problem is to fill the without exceeding its capacity, while maximizing the overall value of its content. In our case we consider a *Multiple Multidimensional Knapsack Problem* (MMKP). That is we have different types of elements (CPU, RAM and Battery) that we need to place in a number of knapsacks (computer nodes). These different constraints can be solved using existing tools and heuristics in the literature [35, 36].

5.6 Coverage techniques

In MBT approaches, it is impossible to generate all possible tests that can be derived from the specification of the system under test. The number of possible is either infinite or very huge even for relatively very small specifications. Consequently, we need to find a way for reducing the number of generated tests. A first alternative consists in generating a fixed number of test cases randomly. A

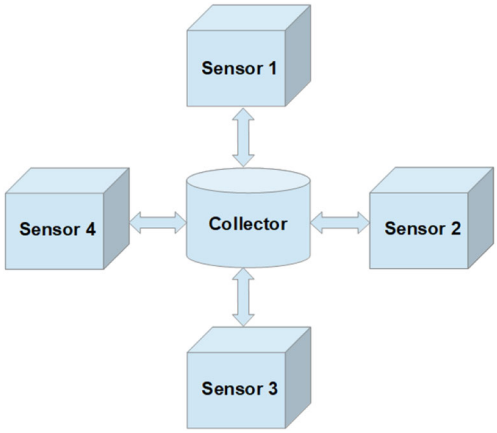

Fig. 1 A temperature measuring system with four sensors and one collector

second alternative consists in generating all possible tests up to a fixed depth of the specification. However both solutions present the risk of ignoring interesting behaviors of the system under test. For that, we adopt a third alternative which consists in using coverage techniques. Different coverage criteria are possible in the field of software testing such as branch coverage and statement coverage [42]. Similarly for the case of timed automata, existing approaches allow to cover finite abstractions of the space of states (like the region graph [52] and the time-abstracting quotient graph [44]) or structural elements of the model of the system under test (like locations or edges [19]).

In our previous work [30] we proposed a finite abstraction of the specification called the observable graph. The later is obtained by computing the product of the model of the specification and the model of the used digital-clock. Sequences of interest covering a specific criterion are extracted from this graph and then extended into valid test cases.

6 Case study: a temperature measuring system with four sensors and one collector

In this section, we propose a simple case study on which we illustrate how the previously introduced techniques can be used. As illustrated in Fig. 1, our case study is made of five objects: four sensors and one collector. The role of the sensors consists in measuring the ambient temperature and then send it to the collector. The collector receives the measured values sent by the sensors and stores them in an appropriate database for future use.

In Fig. 2, we propose a simplified model for the proposed case study made of eight finite state machines (FSMs). On the first hand the behavior of each sensor is given as a separate FSM made of three nodes and three transitions:

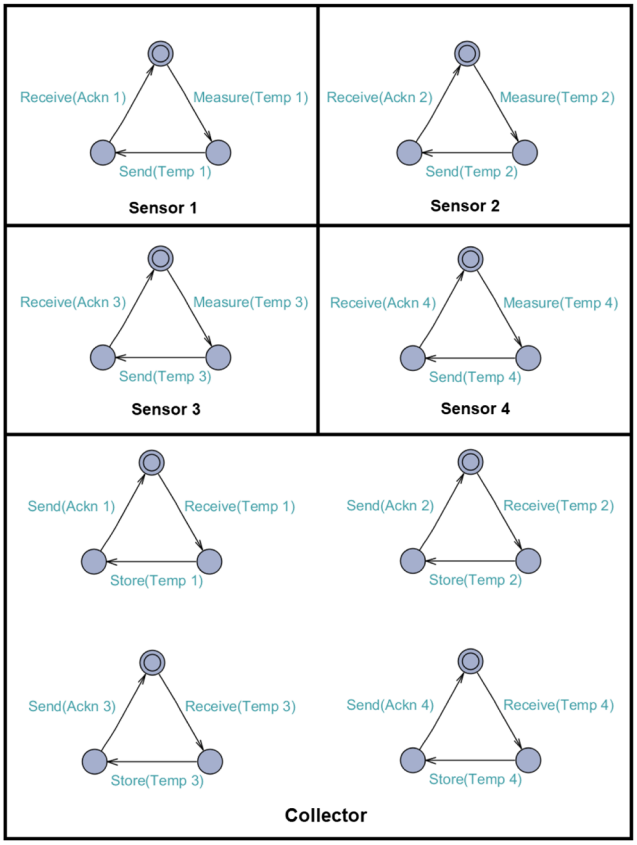

Fig. 2 A simplified model for the temperature measuring system made of eight finite state machines

1. Measuring temperature; 2. Sending temperature to the collector 3. Receiving acknowledgement from the collector. On the other hand the behavior of the collector is given as a product of four FSMs. Each of these FSMs is also made of three nodes and three transitions: 1. Receiving Temperature from the corresponding sensor; 2. Storing the Received Temperature; 3. Sending acknowledgment to the sensor.

Next we explain how the already proposed techniques in the previous sections can be used in order to improve the formal verification and formal based testing for this case study. First we start with formal verification aspects:

- Abstraction: At this level, it is quite easy to notice that the description of the different components of the considered system are given at a very high level. Moreover many details are abstracted away and interaction between the different sensors is not considered as well.
- Modularization and Compositionality: As already mentioned and as illustrated by Fig. 2 the model of the proposed system is given as a network of eight FSMs. Each FSM is made of three states and three transitions. By the way by computing the product of these different

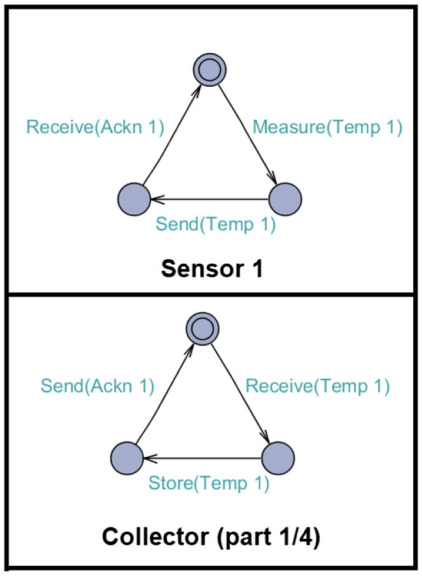

Fig. 3 A possible simplification of the considered model due to symmetry aspects

FSMs we will obtain a big FSM which has approximately $3^8 = 6561$ states. If we increase the number of sensors to 10 the number of states of the product may reach $3^{20} = 3486784401$ states which is a huge number. This illustrates the importance of considering Modularization and Compositionality in order to reduce the size of the considered models.

– Symmetry Detection: It is not difficult to see that the different sensors and the different parts of the collector play symmetric roles. Consequently, the formal verification of the whole considered system can be reduced to the verification of the product of only two FSMs: the first one corresponding to one of the four sensors and the second one corresponding to one of the four parts of the collector (as illustrated in Fig. 3).
– Data Independence Detection: We may assume that the different sensors of the system may measure other entities like pressure and humidity for instance. However by assuming that no correlation exists between temperature and these other entities then there is no need to take into account in the model of the system. This clearly allows to reduce the complexity of the considered model and to reduce its size.

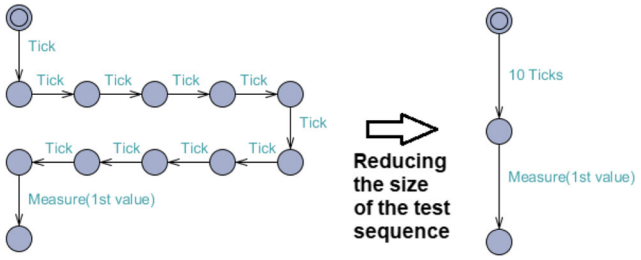

Fig. 5 Reducing the size of digital-clock tests

– Eliminating Functional Dependencies: Assume that the collector stores the average of the different values received from the different sensors. In this case we can clearly see that there is a direct dependency between this stored variable and the the values measured by the sensors. So in order to reduce the complexity of the verification procedure we need to take into account this correlation between these different variables by eliminating the new variable corresponding to the average value since it can be deduced from other variables.
– Exploiting Reversible Rules: Consider the FSM describing the behavior of sensor 1 for instance. This FSM can be simplified a bit more by collapsing the two nodes connected by the transition $Measure(Temp\ 1)$ since this action can be seen as an internal action and does not have any impact on the formal verification of the whole system. Similarly we can collapse the pairs of nodes of the collector FSMs which are connected by the $Store(Temp\ i)$ transitions. Once verification is achieved the collapsed nodes can be separated as they were initially.

Second we move to Model Based Testing aspects:

– Refinement Techniques: Recall that this technique consists in considering an untimed specification and a set of refinement rules which allow to transform each high-level untimed action into a sequence of low-level timed sequence. Test cases are extracted from the untimed specification then the obtained untimed test cases are refined into timed test cases according to the adopted refinement rules. For instance in Fig. 4 the action $Measure(Temp\ 1)$ is refined

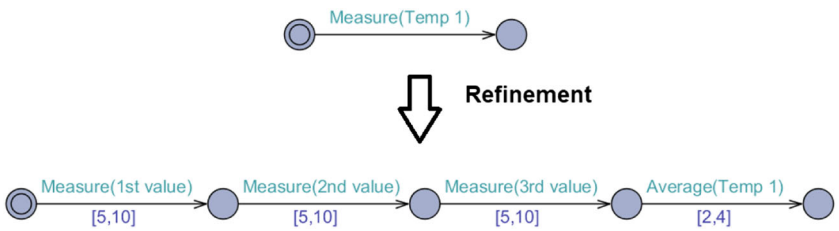

Fig. 4 A possible refinement of a high-level untimed action into a sequence of low-level timed actions

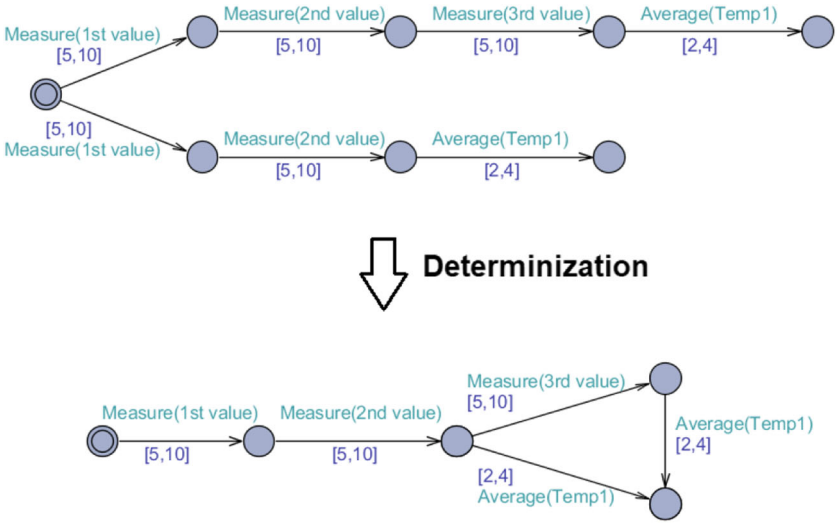

Fig. 6 Timed automata tester generation using determinization techniques

into a sequence made of four timed actions: the measurement of temperature is made by capturing three values consecutively and then computing the average of these three values. This clearly simplifies the test generation procedure and significantly reduces the required computation time and space.

– Reducing the Size of Digital-Clock Tests: As already mentioned a digital-clock test can be seen as a special tree which has a special $Tick$ action which models time progress. The goal of this step consists in reducing the size of the test tree by compacting sequences of $Tick$ actions. This idea is illustrated in Fig. 5 where a sequence of ten $Tick$ actions is replaced with only one transition labelled with 10 $Ticks$. In this manner the size of tests is reduced significantly.
– Timed Automata Testers Generation: When the specification of the system is given as a non-deterministic timed automaton then two options are possible for test generation. The first option consists in generating test cases on-the-fly. That is test generation and test execution are done in parallel. This first choice is difficult in general since it needs high performance calculators. The second choice consists in deteminizing the considered timed automaton off-line before test execution. For instance in Fig. 6, we propose a non-deterministic version of a portion of the model of the considered system. This automaton is non-deterministic since from the initial node the same action leads to different successor nodes. This non-determinism corresponds to the fact that the sensor may measure temperature either by computing the average of three captured values or only two values depending on some internal choices (available resources for example). In the same figure we propose the result of the determinization of the non-deterministic automaton. As already explained this procedure is not always possible in a exact manner. Consequently we may need to make some approximations.
– Test Cases Updating: As our system evolves we need to update the model accordingly. In this case we need to update the already generated test cases as well since it would be very expensive to regenerate them from scratch. In Fig. 7 we propose a possible evolution for our system. Initially the acknowledgement was sent by the collector to the corresponding sensor after storing the temperature in the database. In the new version, the acknowledgement is sent after the storage is achieved. The previously generated tests need to be updated

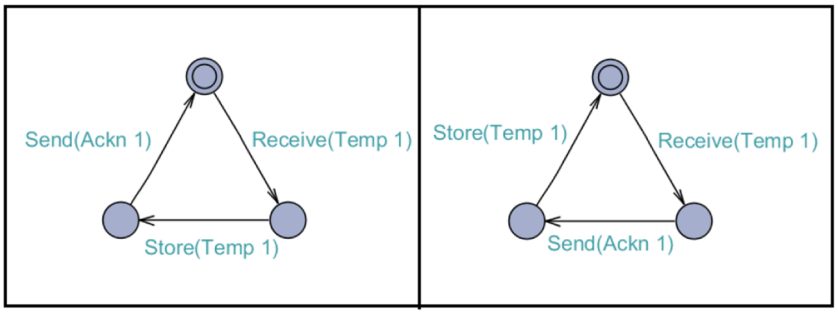

Fig. 7 An example of a possible Update of the Model of the considered System

accordingly in order to minimize the cost and the duration of the test regeneration phase. For this purpose we need to compare the two models of the system (the old and the new ones) and to make a classification of the available test cases as already explained.

– Resource Aware Tester Component Placement: In general the testing architecture may be either centralized or decentralized. In the second case, we need to find a strategy for placing the different test components over the computational nodes of the system. For instance if the collector has enough resources it can host some of the test components dedicated for testing some of the sensors. Similarly if the resources of one of sensors are sufficient then it can host the test component in charge of testing another sensor as well. Adequate optimization techniques need to be used for this purpose.
– Coverage Techniques: These techniques allow to reduce the number of generated tests in a smart way by defining specific selection criteria. For instance in our case we may consider a criterion which allows to cover the different nodes of the different FSMs of the considered model. Similarly, we may consider a second criterion which covers the set of transitions or the set of pairs of consecutive transitions, etc. This can be done by constructing the observable graph as already explained in the previous section.

7 Conclusion

In this work, we proposed several techniques to improve the quality of formal verification and model-based testing approaches for IoT and Smart Cities. These approaches generally suffer from the state explosion and are difficult to implement. On the first hand, the proposed techniques to remedy to problems related to formal verification are as follows: 1. Abstraction; 2. Modularization and Compositionality; 3. Symmetry Detection; 4. Data Independence Detection; 5. Eliminating Functional Dependencies; 6. Exploiting Reversible Rules. On the second hand the proposed techniques related to Model Based Testing are as follows: 1. Refinement Techniques. 2. Reducing the Size of Digital-Clock Tests; 3. Timed Automata Testers Generation; 4. Test Cases Updating; 5. Resource Aware Tester Component Placement; 6. Coverage Technique. In order to show how these different techniques can be used we proposed a simple case study which corresponds to a Temperature Measuring System made of one Collector and four Sensors. As a future work we need to implement the different proposed solutions and to define a set of criteria for selecting the set of appropriate techniques to adopt for each concrete situation. Moreover we may apply several other techniques to improve the testing approaches adopted for Iot and Smart Cities. For instance we may combine functional and load testing aspects as proposed in [28]. We may also consider other optimization techniques used in formal verification.

References

1. Alam F, Mehmood R, Katib I (2020) Comparison of decision trees and deep learning for object classification in autonomous driving. Springer International Publishing, Cham, pp 135–158. https://doi.org/10.1007/978-3-030-13705-2_6
2. Alam F, Mehmood R, Katib I, Albeshri A (2016) Analysis of eight data mining algorithms for smarter internet of things (iot). Proc Comput Sci 98:437–442. https://doi.org/10.1016/j.procs.2016.09.068. http://www.sciencedirect.com/science/article/pii/S187705091632213X. The 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/Affiliated Workshops
3. Alam F, Mehmood R, Katib I, Albogami NN, Albeshri A (2017) Data fusion and iot for smart ubiquitous environments: A survey. IEEE Access 5:9533–9554. https://doi.org/10.1109/ACCESS.2017.2697839
4. Alamoudi E, Mehmood R, Albeshri A, Gojobori T (2020) A survey of methods and tools for large-scale DNA mixture profiling. Springer International Publishing, Cham, pp 217–248. https://doi.org/10.1007/978-3-030-13705-2_9
5. Albino V, Berardi U, Dangelico R (2015) Smart cities: Definitions, dimensions, performance, and initiatives. J Urban Technol 22:3–21
6. Alur R, Dill D (1994) A theory of timed automata. Theor Comput Sci 126:183–235
7. Andraus ZS, Sakallah KA (2004) Automatic abstraction and verification of verilog models. In: Proceedings of the 41st annual design automation conference, DAC '04. ACM, New York, pp 218–223. https://doi.org/10.1145/996566.996629
8. Aqib M, Mehmood R, Alzahrani A, Katib I, Albeshri A, Altowaijri SM (2019) Rapid transit systems: Smarter urban planning using big data, in-memory computing, deep learning, and gpus. Sustainability 11(10). https://doi.org/10.3390/su11102736. https://www.mdpi.com/2071-1050/11/10/2736
9. Aqib M, Mehmood R, Alzahrani A, Katib I, Albeshri A, Altowaijri SM (2019) Smarter traffic prediction using big data, in-memory computing, deep learning and gpus Sensors 19(9). https://doi.org/10.3390/s19092206
10. Benalycherif L, McIsaac A (2009) A semantic condition for data independence and applications in hardware verification. Electron Notes Theor Comput Sci 250(1):39–54. https://doi.org/10.1016/j.entcs.2009.08.004. http://www.sciencedirect.com/science/article/pii/S1571066109003296. Proceedings of the Seventh International Workshop on Automated Verification of Critical Systems (AVoCS 2007)
11. Bensalem S, Krichen M, Majdoub L, Robbana R, Tripakis S (2007) A simplified approach for testing real-time systems based on action refinement. In: ISoLA, Revue des Nouvelles Technologies de l'Information, vol. RNTI-SM-1, pp. 191–202. Cépaduès-Éditions
12. Bertot Y, Castran P (2010) Interactive theorem proving and program development: Coq'Art the calculus of inductive constructions, 1st edn, Springer Publishing Company, Incorporated
13. Bornot S, Sifakis J, Tripakis S (1998) Modeling urgency in timed systems. In: Compositionality, LNCS, vol 1536. Springer

14. Chaki S, Clarke EM, Groce A, Jha S, Veith H (2004) Modular verification of software components in c. IEEE Trans Softw Eng 30(6):388–402. https://doi.org/10.1109/TSE.2004.22
15. Lee C-C, Jiang JR, Huang C-Y, Mishchenko A (2007) Scalable exploration of functional dependency by interpolation and incremental sat solving. In: 2007 IEEE/ACM international conference on computer-aided design, pp 227–233. https://doi.org/10.1109/ICCAD.2007.4397270
16. Clarke EM, Emerson EA (1982) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of programs, workshop. Springer-Verlag, Berlin, pp 52–71. http://dl.acm.org/citation.cfm?id=648063.747438
17. Emerson EA, Wahl T (2005) Dynamic symmetry reduction. In: Halbwachs N, Zuck LD (eds) Tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 382-396
18. Gordon MJC, Melham TF (eds) (1993) Introduction to HOL: A theorem proving environment for higher order logic. Cambridge University Press, New York
19. Hessel A, Larsen K, Nielsen B, Pettersson P, Skou A (2003) Time-optimal real-time test case generation using UPPAAL. In: FATES'03
20. Hu AJ, Dill DL (1993) Reducing bdd size by exploiting functional dependencies. In: 30th ACM/IEEE design automation conference, pp 266–271. https://doi.org/10.1145/157485.164888
21. Iosif R (2002) Symmetry reduction criteria for software model checking. In: Bošnački D, Leue S (eds) Software, model checking. Springer, Berlin, pp 22–41
22. Ip CN (1998) Generalized reversible rules. In: Proceedings of the 2nd international conference on formal methods in computer-aided design, FMCAD '98. Springer-Verlag, London, pp 403–420. http://dl.acm.org/citation.cfm?id=646185.758715
23. Ip CN, Dill DL State reduction using reversible rules. In: Proceedings of the 33st Conference on Design Automation, Las Vegas, Nevada, USA, Las Vegas Convention Center, June 3-7, 1996., pp 564–567 1996. https://doi.org/10.1145/240518.240625
24. Jiang JHR, Brayton RK (2004) Functional dependency for verification reduction. In: Alur R, Peled DA (eds) Computer aided verification. Springer, Berlin, pp 268-280
25. Kesten Y, Pnueli A, Zlatuška J (1998) Modularization and abstraction: The keys to practical formal verification. In: Brim L, Gruska J (eds) Mathematical foundations of computer science 1998. Springer, Berlin, pp 54–71
26. Krichen M (2012) A formal framework for black-box conformance testing of distributed real-time systems. IJCCBS 3(1/2):26–43. https://doi.org/10.1504/IJCCBS.2012.045075
27. Krichen M, Cheikhrouhou O, Lahami M, Alroobaea R, Jmal Maâlej A. (2018) Towards a model-based testing framework for the security of internet of things for smart city applications. In: Mehmood R, Bhaduri B, Katib I, Chlamtac I (eds) Smart societies, infrastructure, technologies and applications. Springer International Publishing, Cham, pp 360-365
28. Krichen M, Maâlej AJ, Lahami M (2018) A model-based approach to combine conformance and load tests: an ehealth case study. IJCCBS 8(3/4):282–310. https://doi.org/10.1504/IJCCBS.2018.096437
29. Krichen M, Tripakis S Black-box conformance testing for real-time systems. In: 11th international spin workshop on model checking of software (SPIN'04), LNCS, vol. 2989. Springer (2004). Available at http://www-verimag.imag.fr/PEOPLE/Stavros.Tripakis/papers/timetest.pdf
30. Krichen M, Tripakis S (2009) Conformance testing for real-time systems. Formal Methods in System Design 34(3):238–304
31. Kripke SA (1963) Semantical considerations on modal logic. Acta Philosophica Fennica 16(1963):83–94
32. Kwiatkowska M, Norman G, Parker D (2006) Symmetry reduction for probabilistic model checking. In: Ball T, Jones RB (eds) Computer aided verification. Springer, Berlin, pp 234-248
33. Lahami M, Fakhfakh F, Krichen M, Jmaïel M (2012) Towards a TTCN-3 test system for runtime testing of adaptable and distributed systems. In: Proceedings of the 24th IFIP WG 6.1 international conference testing software and systems (ICTSS'12), pp 71–86
34. Lahami M, Krichen M, Barhoumi H, Jmaiel M (2015) Selective test generation approach for testing dynamic behavioral adaptations. In: Testing software and systems - 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25, 2015, Proceedings, pp 224–239. https://doi.org/10.1007/978-3-319-25945-1_14
35. Lahami M, Krichen M, Bouchakwa M, Jmaïel M (2012) Using knapsack problem model to design a resource aware test architecture for adaptable and distributed systems. In: Proceedings of the 24th IFIP WG 6.1 international conference testing software and systems (ICTSS'12), pp. 103–118
36. Maâlej AJ, Lahami M, Krichen M, Jmaïel M (2018) Distributed and resource-aware load testing of WS-BPEL compositions. In: ICEIS (2), pp. 29–38. SciTePress
37. Mehmood R, Graham G (2015) Big data logistics: A healthcare transport capacity sharing model. Proc Comput Sci 64:1107–1114. http://www.sciencedirect.com/science/article/pii/S1877050915027015. Conference on ENTERprise Information Systems/International Conference on Project MANagement/Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2015 October 7–9, 2015, https://doi.org/10.1016/j.procs.2015.08.566
38. Mehmood R, Katib I, Chlamtac I, Bhaduri B (2018) Smart societies, infrastructure, technologies and applications: First international conference, SCITA 2017, Jeddah, Saudi Arabia, November 27–29, 2017, Proceedings, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering book series (LNICST, volume 224) Springer. https://doi.org/10.1007/978-3-319-94180-6
39. Momtahan L (2005) Towards a small model theorem for data independent systems in alloy. Electron Notes Theor Comput Sci 128(6):37–52. https://doi.org/10.1016/j.entcs.2005.04.003. http://www.sciencedirect.com/science/article/pii/S1571066105002355. Proceedings of the Fouth International Workshop on Automated Verification of Critical Systems (AVoCS 2004)
40. Muhammed T, Mehmood R, Albeshri A, Alzahrani A (2020) HCDSR: A hierarchical clustered fault tolerant routing technique for IoT-based smart societies. Springer International Publishing, Cham, pp 609–628. https://doi.org/10.1007/978-3-030-13705-2_25
41. Muhammed T, Mehmood R, Albeshri A, Katib I (2018) Ubehealth: A personalized ubiquitous cloud and edge-enabled networked healthcare system for smart cities. IEEE Access 6:32258–32285. https://doi.org/10.1109/ACCESS.2018.2846609
42. Myers G (1979) The art of software testing. Wiley
43. Neto ACD, Travassos GH (2010) A picture from the model-based testing area: Concepts, techniques, and challenges. Adv Comput 80:45–120
44. Nielsen B, Skou A (2001) Automated test generation from timed automata. In: TACAS'01. LNCS 2031, Springer
45. Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol 2283. Springer
46. Norris IPC, Dill DL (1996) Better verification through symmetry. Formal Methods in System Design 9(1):41–75. https://doi.org/10.1007/BF00625968

47. Queille JP, Sifakis J (1982) Specification and verification of concurrent systems in cesar. In: Proceedings of the 5th colloquium on international symposium on programming. Springer-Verlag, London, pp 337–351. http://dl.acm.org/citation.cfm?id=647325.721668
48. Roscoe AW, Broadfoot PJ (1999) Proving security protocols with model checkers by data independence techniques. J Comput Secur 7(1):147–190. http://content.iospress.com/articles/journal-of-computer-security/jcs120
49. Schlingensiepen J, Mehmood R, Nemtanu FC, Niculescu M (2014) Increasing sustainability of road transport in european cities and metropolitan areas by facilitating autonomic road transport systems (arts). In: Wellnitz J, Subic A, Trufin R (eds) Sustainable automotive technologies 2013. Springer International Publishing, Cham, pp 201-210
50. Sifakis J, Yovine S (1996) Compositional specification of timed systems. In: 13th annual symposium on theoretical aspects of computer science, STACS'96, LNCS, vol 1046. Spinger-Verlag
51. Söderström O, Paasche T, Klauser F (2014) Smart cities as corporate storytelling. City: analysis of urban trends 18. https://doi.org/10.1080/13604813.2014.906716
52. Springintveld J, Vaandrager F, D'Argenio P (2001) Testing timed automata. Theoretical Computer Science 254
53. Thacker RA, Jones KR, Myers CJ, Zheng H (2010) Automatic abstraction for verification of cyber-physical systems. In: Proceedings of the 1st ACM/IEEE international conference on cyber-physical systems, ICCPS '10. ACM, New York, pp 12–21. https://doi.org/10.1145/1795194.1795197
54. Tretmans J (1992) A formal approcah to conformance testing. Ph.D. thesis, University of Twente Twente The Netherlands
55. Utting M, Pretschner A, Legeard B (2012) A taxonomy of model-based testing approaches. Softw Test Verif Reliab 22(5):297–312. https://doi.org/10.1002/stvr.456
56. Velev MN (2001) Automatic abstraction of memories in the formal verification of superscalar microprocessors. In: Margaria T, Yi W (eds) Tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 252–267
57. Wahl T, Donaldson A (2010) Replication and abstraction: Symmetry in automated formal verification. Symmetry 2(2):799–847. https://doi.org/10.3390/sym2020799
58. Lee W, Pardo A, Jang J-Y, Hachtel G, Somenzi F (1996) Tearing based automatic abstraction for ctl model checking. In: Proceedings of international conference on computer aided design, pp 76–81. https://doi.org/10.1109/ICCAD.1996.568969
59. Zhang J, Goldsby HJ, Cheng BH (2009) Modular verification of dynamically adaptive systems. In: Proceedings of the 8th ACM international conference on aspect-oriented software development, AOSD '09. ACM, New York, pp 161–172. https://doi.org/10.1145/1509239.1509262
60. Zhu H, Belli F (2009) Advancing test automation technology to meet the challenges of model-based software testing - guest editors' introduction to the special section of the third IEEE international workshop on automation of software test (AST 2008). Inf Softw Technol 51(11):1485–1486