Test-Case Generation for Testing of Timeliness*

Robert Nilsson, Sten F. Andler and Jonas Mellin

February 27, 2001

Abstract

In complex distributed real-time systems the temporal correctness is imperative for dependability. Industrial practice has few methods for testing of temporal correctness and the methods that exist are often ad-hoc. A problem associated with testing real-time is that their timeliness depends on the execution order of tasks. This is particularly problematic for event-triggered real-time systems where the system continuously is notified of events that influence the execution order. Further, real-time systems may behave differently depending not only on time dependencies in program logic but also on differences in execution times. This paper investigates how existing test-case generation methods take these factors into consideration and explains the opportunity to construct better methods and metrics for their evaluation.

An analysis of current methods for generating test cases for testing temporal properties of real-time systems is presented. The methods are classified according to their characteristics and analyzed in relation to the requirements of an event-triggered system model. The aim of the classification is to detect similarities between different test-case generation methods so that the characteristics that have an impact on the applicability of the method when testing timeliness can be determined. We conclude that existing work only consider a subset of the factors that influence timeliness, and therefore propose research activities that take these factors into consideration during test-case generation.

1 Introduction

Modern real-time systems tend to be increasingly complex. This is particularly true for distributed real-time systems, where the complexity of both distribution and real-time issues need to be considered. Moreover, real-time systems generally must be dependable and therefore temporal correctness of the software is imperative. These characteristics imply that there is a need for rigorous verification methods to detect temporal faults that could lead to critical failures. Industrial practice has few methods for verifying temporal correctness of complex systems such as distributed real-time applications, and the methods that exist are often case-specific or ad-hoc [Sch94, BJ00].

Testing is a method to dynamically verify software by execution in order to detect errors and failures [Lap94]. *Test-case generation* is the process of selectively generating test cases that exercise system behaviors likely to reveal errors.

This paper focuses on selective generation of test cases for distributed real-time systems. There exist generally accepted methods for generation of test cases for sequential software, based on, for example, specifications or code structure [Bei90]. Real-time systems are often concurrent; this complicates generation and selection of suitable test cases because system behavior is dependent on the non-deterministic order in which tasks execute (cf. [TH99]). Further, real-time systems may behave differently depending on time dependencies in program logic and differences in execution times. In particular, we investigate how existing test-case generation methods to a varying degree take these factors into consideration for testing *timeliness*, which is the ability of the system to fulfill its time constraints.

The investigated test-case generation methods have been classified according to their characteristics and analyzed in relation to the requirements of an event-triggered system model.

*This work is funded by the national Swedish Real-Time Systems research initiative ARTES (www.artes.uu.se), supported by the Swedish Foundation for Strategic Research.. Submitted for publication

More specifically, our model of the target system adopts the event-triggered design paradigm as presented by Kopetz et al. [KZF+91], and constrains the execution environment to increase testability, as described by Birgisson et al. [BMA99]. The aim of our classification is to reveal similarities between different test-case generation methods and allow determination whether a certain type of characteristics has an impact on the applicability of the method to testing of timeliness. Furthermore, the result of the classification is analyzed against the requirement of our model of the target system in order to highlight issues that have not been covered in existing research.

2 Test-Case Generation Model

Test-case generation is based on application knowledge such as specification, code structure, and system design (see figure 1). This information is used to selectively generate a set of test cases, sometimes referred to as a *test suite*, that has high probability of revealing a specific class of errors or deviations from the expected behavior [Lap94].

An *execution environment*, in this context, is the architecture in which the tested applications run. This includes real-time operating system services, communication primitives, and properties of underlying hardware. The execution environment gives certain constraints on applicability of different test methods, e.g. required contents of the test cases. Hence, we suggest that properties of the execution environment should be considered in the test-case generation process.

Test cases are generated in accordance with the test-case selection criterion, which depend on the specific test method that is being used. In this paper a *test coverage criterion* denote the level of ambition for a test method, i.e., test coverage criterion sets a goal of when an application has been tested sufficiently. A *test-case selection criterion* is related to this – a criterion for selecting test cases that eventually will fulfill the test coverage criterion.

When a test suite has been prepared, it is executed on the system under test. The test-case execution phase requires that the execution environment is sufficiently controllable and observable so that the desired test scenario can be enforced and executed. The term *observability* denotes how well the system provides facilities for monitoring or observing the execution of application programs during testing [Sch93]. *Controllability* is associated with how well a tester can control the behavior of the system under test [Sch93].

Test evaluation is the process of comparing the result of test-case execution with the expected result. Test evaluation also incorporates evaluation against the test coverage criterion to determine if sufficient test coverage has been attained.

3 Generating Test Cases for Testing of Timeliness

This section presents the state-of-the-art in test-case generation for testing of time constraints in real-time systems. We argue a study of this area is interesting and what problems are solved by the analysis. The methods that are relevant, from a timeliness perspective, are classified according to a set of attributes in section 4. These attributes and the criteria for classification are explained in section 3.2.

3.1 The need for Testing of Timeliness

There is a plethora of articles describing different methods for generating or selecting tests for various target systems and frameworks. As a developer in the domain of complex distributed real-time systems, it is hard to know what test-case generation methods are meaningful for the properties that one wants to test in this domain. Timeliness is one of the most important properties of real-time systems. Few articles consider testing of timeliness, but refer to formal proofs, static analysis and scheduling theory to guarantee timeliness. Such analysis often requires exact estimations of worst-case execution times, known load patterns, and off-line scheduling. However, event-triggered real-time systems often allow a task load with mixed criticality and dynamic scheduling policies to be more flexible. This complicate analysis and proofs while testing becomes necessary in order to achieve confidence in the temporal correctness. By investigating and classifying methods for generating test cases from the timeliness perspective it is possible to determine what aspects already have

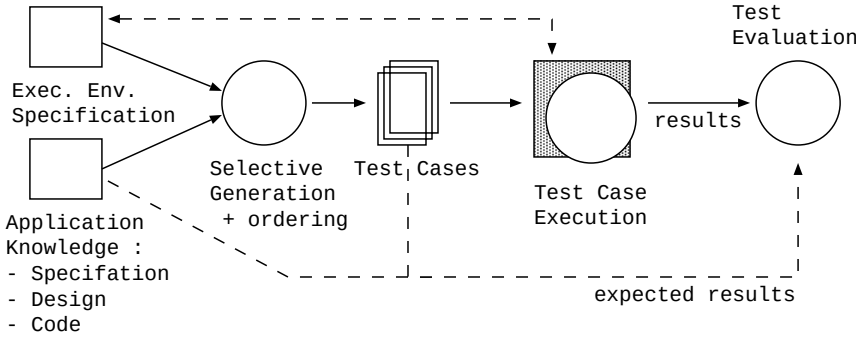

Figure 1: High-level testing model

been addressed and what aspects needs further consideration and research when generating test cases for dependable real-time systems (see also [Nil00]).

3.2 Classification Attributes

The methods that have been encountered in literature are classified in relation to each other, forming a taxonomy over test-case generation methods for testing of temporal properties. The attributes have been selected so that they should apply to all test generation methods that consider time constraints.

3.2.1 Specification Type

This attribute classifies the methods by the type of specification from which test cases are derived. A specification in this context is any model that incorporates the expected temporal behavior of a real-time system. Four main categories of specification type have been identified; process algebra, finite state machines, temporal logic, and Petri-net specification. The aim for this classification is to identify if a test-case generation method requires a specification type that has inherent advantages or problems when used for testing of timeliness.

Some test-case generation methods, used for testing temporal properties, are independent of the specification. These methods often rely on some other, possibly manual, method for verifying temporal correctness. Examples of such methods are included in this article for completeness but will not be classified by this attribute.

3.2.2 Test-Case Contents

Different methods supply various degree of test-case detail. Some methods supply only "test-data" and rely on some other mechanism to verify the implementation against a temporal specification. These methods are not specifically aimed at testing temporal correctness; thus when the execution time of a task depends on the input data, appropriate input data for such testing must be specified.

Other methods aim at generating complete execution sequences with series of input events and expected output events. This class of test data is denoted as "event sequences" in this work. A special version of event-sequence representation is executable "test processes", which are run in parallel with the application, supplying inputs at certain points in the execution and responses to outputs from the system. A test process can potentially incorporate timed input events, input data, and expected results.

Another category is test cases that include a specification of the initial internal state of the system from which the event sequence should be applied as discussed by Birgisson et al. [BMA99].

3.2.3 Time Base

From the timeliness perspective, the time base used in test-case generation is interesting. A continuous time base makes it possible to specify time constraints in a more precise way than in its discrete counterpart. However, this ability makes the number of test cases infinite.

Some methods assume a discrete time base, where there is a constant period between time

instants, which simplifies testing. This may be a valid assumption, since all currently used computer systems operate with a discrete time base at the lowest level, i.e., at the level of hardware clock cycles.

3.2.4 Degree of Automation

From a timeliness testing perspective a high degree of automation is desirable to reduce the test effort, and thus, to be able to execute more system-level tests in a limited time. Most methods for test-case generation recognize the need for automation and tool support for generating test-suites. In some cases, the methods support fully automated test-case generation. In other methods a tool allows the tester to specify aspects that needs to be tested, and the generation of test cases is then performed in semi-automated interaction with this tool. Some methods do not consider automation explicitly but do have the required properties needed to support an automated solution. Hence, four possible categories have been identified; "Automated", "Semi-Automated", "Potentially Automated", and "Not Automated".

4 Test-Case Generation Methods

This section presents the most significant test-case generation methods from the timeliness perspective. The layout of this section follows the categories of the "specification type" attribute.

4.1 Methods Based on Process Algebra Specifications

An example of a method based on process algebra is presented by Clarke and Lee [CL97]. In this paper they introduce a framework for testing time constraints of real-time systems. The time constraints are specified in a constraint graph that specifies intervals in which events can occur and dependencies between events. The graph is translated to processes in the algebra of communicating shared resources (ACSR). Test processes produced from the process algebraic specifications can be used to verify the specification model of the system as well as be converted to test cases for the actual software implementation. The test processes are run in parallel with the system under test to supply timed inputs to the system and intercept outputs. The article provides a taxonomy of timing-constraint faults and a set of coverage criteria for detecting various types of implementation faults. The recommended test case generation method focuses on deriving test cases close to the end-points of the intervals at which events are specified to occur.

The target system for the presented is real-time systems and protocols. The generated test cases are primarily aimed at testing of ASCR specifications, but the authors conjecture that test cases can be reused for testing of actual software implementations. This method is interesting from the perspective of our target model since input occurs continuously and time constraints are specified in a well-formed way. However, the focus of this test-case generation method is to test "behavioral constraints" – time constraints on the input to the system. No method for generation of test cases for testing of time constraints on the output values is presented.

Cleveland and Zwarico [CZ91] propose a method that incorporates a framework for generating "behavioral preorders", which is a specification in process algebra that is suitable for reasoning about real-time characteristics of reactive systems. They are mainly concerned with the problems arising when describing timed systems in a process algebraic notation. No concrete method for test-case generation for real-time software systems is provided.

4.2 Methods Based on State Machine Specifications

A finite state machine is a well-known abstraction of computer systems. An interesting property of real-time systems is that a point in time can be viewed as an event that may cause a transition in a finite state model. When testing systems for non-temporal properties, methods based on finite-state-machine representations have already been exploited for test-case generation, e.g., [FBK+91]. In this section, methods for testing temporal properties based on three different flavors of state machine models are presented.

A common problem in finite-state-machine representations is the state explosion problem. This problem often occurs when the number of states in the actual implementation is greater than the number of states in the finite state

model. Another problem with these models are events that could lead to more than one state, i.e., non-determinism [FBK+91].

4.2.1 Timed I/O Automaton

Petitjean and Fochal [PF99a] propose a test architecture for testing of timed systems. A timed system is described as a timed automaton that can be expressed as a region graph. This work is built on the timed- I/O automaton theory proposed by Alur and Dill [AD94]. In their model every automaton has a set of states and a set of clocks. The clocks are represented by real numbers that proceed at the same rate and can be reset by any transition in the system. The refinement of this model by Petitjean and Fochal are clock regions that allow time constraints to be represented by a region graph where the number of clock regions becomes smaller than in the original model by Alur and Dill.

The Petitjean and Fochal paper further describes how test cases are exercised in the system under test. The architecture takes the management of clocks away from the implementation and lifts it to the level of the tester, because it must use timers that correspond to the real-valued clocks in the specification. Thus, real-value clocks are kept within a read-only module of the test-driver, equivalent with the clock representation in the system. When the system under test generates output events, the test driver verifies these against the clock zones. However, it is inconclusive how efficient this test case generation method is

In order to get sufficient test coverage, test cases are selected from each vertex that makes up a clock region in the region graph. This policy guarantees that at least one test case has been exercised for each combination of time constraints. We believe this approach to be promising for generating test suits for testing timing constraints. However, the method abstract away from properties in the execution environment and it is inconclusive if the generated test cases can be used in our target model, this issue needs investigation.

4.2.2 Timed Transition Systems

Another method which takes a finite-state-machine approach is Cardell-Oliver and Glover [COG98]. The purpose of their article is to present a practical and complete method for generating conformance tests for real-time systems. The paper starts from a formal-methods perspective, where software specifications is iteratively refined and formally verified at each step. The authors note that the last step from symbol manipulation to an actual hardware and software implementation requires testing since the implementation no longer is a provable description.

According to Cardell-Oliver and Glover, a formal test method has four stages; checking of the test hypothesis, generating test cases, running the tests, and evaluating the test results. The test hypothesis defines the assumptions about the property under test so that correct conclusions from the test results can be drawn. This method requires that the system under test can be viewed, at some level of abstraction, as a deterministic finite state automaton. According to the paper, a test method is complete if "its test generation algorithm determines a finite set of test cases sufficient to demonstrate that the formal model of the implementation under test is equivalent to its specification". The specification language used in this method is Timed Transition Systems (TTS), but the authors claim that the method is adaptable to other formal languages such as timed CSP.

The basic idea of the method is to generate test cases that exercise all transitions between states and compares observable variables with their specification counterparts. Each transition in the TTS specification has an associated clock guard in discrete time units that is used for expressing time constraints. The required input for this method is a specification of the system as a TTS process as well as a specification of its environment. From these models, a timed action automaton is derived by symbolic execution. The test-case generation algorithm takes the timed action automaton associated with a TTS specification and constructs a finite set of test cases that will test each timed action edge of the automata. Each test is a sequence of timed actions.

The authors note that their method results in a large number of test cases, but suggest that other methods should be applied to eliminate redundancy. They also claim that if a unique sequence of output events can identify each state, the number of required test cases for conformance testing can be decreased.

4.2.3 Extended Timed Input-Output State Machines

Koné [Kon95] introduces a method for designing tests for time constraints. However, the main contributions of his work are a formal approach to modeling and a theory for testing of time dependencies in system behavior. The approach will not be described in detail here since it is similar to the methods described above and does not handle test-case generation for software explicitly. However the article is mentioned in this context since it is one of the seminal works in the area.

Laurencot and Castanet [LC97] show different existing methods to integrate time in existing formal modeling languages so that it is possible to test time constraints. The article also aims to show how the notion of time can be integrated in test cases for distributed systems and protocols. The importance of time in test cases for time-dependent systems is highlighted. Three formal specifications from other authors are presented in their overview and the notion of time in each of them is discussed. The different specifications are the automata of Alur and Dill [AD94], Timed Transition Models (e.g., [COG98]), and Extended Time Input Output State Machine (ETIOSM) [Kon95]. Based on the latter notation, a canonical "tester" that handles time is presented. The method decomposes timed formal models, in this case ETIOSM, into sub-models for each real-valued timer. These are then analyzed according to a set of rules and used for building the tester that incorporates time. According to the paper, test cases given by this method can be formulated in TTCN test language. An interesting issue is it that the authors claim that their tester is very well suited for testing distributed systems or protocols, because of the use of a transition for each transmission and reception. In addition, propagation delay is part of the models.

4.3 Methods Based on Temporal Logic Specifications

Mandrioli et al. [MMM95] suggest a method for functional testing of real-time systems based on specifications of system behavior in TRIO. TRIO is an extension of a temporal-logic language defined to deal explicitly with strict time requirements. A specification of behaviors is decomposed into elemental test-case fragments that later can be recombined to form test cases consisting of combinations of timed inputs and corresponding outputs. A test generation tool helps a test engineer to device test cases and includes history generator/checker components. The history generator is used for deriving all elementary test cases satisfying a TRIO formula at a given time instant. The elemental test cases are timed input-output pairs. These pairs can be combined and shifted in time to create a large number of partial test cases. In order to get a manageable test suite, a heuristic function combined with human guidance is used for selecting which elemental test cases should be combined in order to satisfy some test coverage criterion. The authors claim that most coverage criteria can be used with this method. A problem with this method, however, is how to relate actual output events instances to multiple specified output events of the same type. This problem is partly solved by a history checker that analyzes the generated history from the system under test. Further, this method assumes that all system behavior can be specified with logic specifications; for system level testing the number and complexity of logic formulas will increase and it is inconclusive how the method scales.

In a more recent work, SanPietro et al. [SMM00] expand the work by Mandriolli et al. to incorporate high-level, structured specifications that can be combined with the low-level specifications proposed in the earlier work. The main difference in this work is that the language allows modularization and presents a graphical notation of modules and components. The structured model is translated to a graph which is traversed by an algorithm to construct execution sequences for the overall system (cf. [MMM95]). The test-case generation semi-automatically constructs execution sequences in accordance with some coverage criterion. In our opinion, this extension increases the applicability of the method presented by Mandrioli et al. It would be interesting to further evaluate this method in the context of testing of timeliness and for example the effect of constraining the observation granularity.

4.4 Methods Based on Timed Petri-Net Specifications

Braberman et al. [BFM97] introduces a method for generating test cases for real-time systems based on timed Petri-nets. The method aims at

extracting knowledge not only from the specification or code, but from all steps during development, e.g. design. The article suggests the use of a design notation SA/SD-RT for specifying the behavior of a real-time system. For test purposes, this design specification is translated to a high level, timed Petri-net notation from which a timed reachability tree (TRT) can be derived. Each path from root to leaf in this tree represent a "situation" that in itself represent a potentially unlimited number of test cases due to a continuous time base. These "situations" are the base for generating test cases. The authors present different reduction schemes that can be applied to reduce the number of test cases resulting from each situation. The higher-level adequacy criteria presented in the paper are based on "situation" coverage, e.g. "one is enough" is satisfied if one test execution from each situation is executed. Listed future work is to create a tool which helps to manage and automate the method, an extension for applying the method on isolated components, and also to investigate how architectural information, such as scheduling policies, can be taken into account during test-case generation. An identified but unsolved problem with this method is that there is no way for deriving actual input test data, only temporal information about sequences of events can be produced, e.g. what timed input sequences are needed to test all situations. We believe that the method by Braberman et al. is one of the most promising methods in this survey. We share the opinion with the authors that it is of great importance to take advantage of the additional architectural design information during test-case generation.

In an earlier paper, Morasca and Pezze [MP90] propose a method for testing concurrent and real-time systems that uses high-level Petri-nets for specification and implementation. The Petri-nets proposed in their article are Er-nets, which can handle concurrency, where time is modeled as a special property associated with the transitions in the model. Problems with testing of concurrent and real-time systems are presented, high-lighting the problems with non-determinism and combinations of test-scenarios. Also, this paper presents a number of test criteria for selecting test cases for this kind of systems. This article takes problems with concurrency into consideration, but testing of the time constraints is not explicitly considered. Further, it is hard to determine how applicable it is for implementations that do not use Er-nets.

4.5 Methods for Enforcing Temporal Behaviors

Methods that are not based on a specification are generally not applicable for test-case generation aimed at testing temporal properties. The reason for this is that in order to determine if some transaction violates a time constraint, e.g. misses its deadline, the constraint must be specified in some way.

From our perspective, such test-case generation methods can still be helpful in finding test data that influence the temporal behavior of a transaction in a specific way. There exist other methods than test-case generation for finding data that cause the maximum execution time of a task, for example static analysis of code [PS91] and measurement [PF99b]. However, in static analysis, the value of the estimated maximum execution time is often pessimistic [MW98] and assumes that components such as caches and pipelines are turned of.

Wegener et al.[WSJE97] propose a method that uses genetic algorithms to generate test data for testing temporal properties of real-time systems. The heuristic method aims to find the longest and shortest execution paths of real-time programs. The paper shows that the genetic algorithm method easily outperforms a random method in finding the longest and shortest execution paths, measured in the number of clock cycles of program execution. The fitness function is supplied by a simulation tool, which counts the number of executed machine level instructions and summarizes the associated cycle times. This is only an approximation of the real execution time and the article states that temporal testing must be repeated for each platform where the software will run; this further emphasize the importance of automation. One problem is that there is no guarantee that a heuristic approach can find the extremes, the decision on when to stop the search for them is arbitrary.

4.6 Analysis and Classifications

Papers describing methods for test-case generation aimed for testing temporal properties are listed above. The result of our classification of these methods is shown in table 1.

Assuming that these papers represent all or most current methods aimed at testing time

Authors [Reference]	Specification	Test Cases	Time Base	Automation
Clarke and Lee [CL97]	Process Algebra	Test Process	Discrete	Automated
Cleaveland and Zwarico [CZ91]	Process Algebra	Test Process	Discrete	Not Automated
Mandrioli et al. [MMM95]	Temporal logic	Event Sequence	Discrete	Semi-Automated
SanPietro et al. [SMM00]	Temporal logic	Event Sequence	Discrete	Semi-Automated
Braberman et al. [BFM97]	Petri-nets	Event Sequence	Continuous	Semi-Automated
Morasca and Pezze [MP90]	Petri-nets	Event Sequence	Continuous	Potentially Automated
Laurencot and Castanet [LC97]	ETIOSM	Test Process	Continuous	Semi-Automated
Petitjean and Fochal [PF99a]	Timed Input Output Automata	Test Process	Continuous	Potentially Automated
Cardell-Oliver and Glover [COG98]	Timed Transition Systems	Test Process	Discrete	Automated
Koné [Kon95]	ETIOSM	Test Process	Continuous	Not Automated
Wegener et al. [WSJE97]	N/A	Test Data	Discrete	Automated

Table 1: Classification of Test-case Generation Methods

constraints, an observation is that the type of specification has influence on the contents of produced test cases. For example, temporal logic and Petri-net specifications generally generates test cases that consist of event sequences whereas test cases generated from some finite state machine or process algebra often are specified as test processes. None of the methods include information about internal states of the execution environment in the test cases and, thus, it is impossible to get full test coverage of non-deterministic internal behaviors.

Another observation from the table is that the time base used in these methods appears to be unrelated to the specification language or test case-type. A note to this classification is that in the method that use genetic algorithms for deriving input data [WSJE97], we have assumed that they use a discrete time base, e.g., clock cycles for measuring execution times. However, it is not relevant to these methods, since they rely on some other mechanism to verify time constraints and are only concerned with forcing a task to a specific temporal behavior.

One mayor conclusion that can be drawn from the presented methods is that very few consider internal states of the system when generating test cases. The only approaches that consider any form of such states at all are methods based on finite-state machines or Petri-nets. However, in these representations it is difficult to maintain a relationship between the states of the automata and the internal states of the computer system because the approach with these modeling methods often abstract too far away from mechanisms in the execution environment. Further, if non-determinism is allowed, finite-state-machine methods seldom apply due to state explosion problems.

Other methods consider the system as a black-box and only verify that the time from when input events are supplied to the time where output is signaled corresponds to the constraints in the specification. This is indeed an applicable method in some systems, but for dependable systems with non-deterministic execution times and execution orders, it is generally not sufficient to supply different sets of timed inputs to guarantee timeliness. Parameters and internal states are bound to have an impact on timeliness, i.e. different paths in the code are executed for different classes of input data, and thus, must be tested.

A related, important, observation is that very few of the specification based testing methods supply input parameters. The only input that is assumed is events and timing between events. This may be sufficient in some control applications, but in general, a real-time system has to read data values from sensors in its environment and may act differently upon different data values.

An interesting observation is that most methods assume an event-triggered paradigm; none of the test-case generation methods assume that input events are constrained in any way. The only constraint on system behavior in this context is that some of the works assume a discrete time base. A disadvantage with this is that it is hard to determine the impact of a constrained execution environment in existing methods. Hence, it would be interesting to investigate if constraints, such as the ones presented in [Mel98] could be used with these test-case generation methods to reduce the size of the test suite. For example, a lot of input sequences may be considered equivalent in a more sparsely observed environment.

The only method that considers distributed systems explicitly is Laurencot and Castanet [LC97]. Their approach is finite state machine based. It is inconclusive how well this method scales to large real-time systems and how the states in the specification can be mapped to internal states of the system. Nevertheless, it is in our opinion a promising research direction. The fact that this is the only test-case generation method found in the survey that considers distribution explicitly is surprising, since distribution is an inherent aspect of real-time systems.

5 Conclusions

This section summarizes the material presented in this article. First, the state-of-the-art and general impressions of the area are briefly discussed. Second, the contributions from the classification and analysis are highlighted. Third, some future research directions are suggested that would remedy a few of the identified problems.

5.1 Discussion

The area of testing of real-time properties is slowly beginning to get more attention in academia and industry; the reasons for this we believe is that real-time systems are becoming increasingly complex. The static analysis methods and formal verification methods have trouble keeping up with the required flexibility, diversity of applications, immense parallelism, and distribution of the next generation of embedded systems. Hence, practitioners and researchers turn to testing in hope of finding a simpler way to verify overall correctness. Unfortunately, testing cannot provide absolute guarantees of correctness; the number of test cases needed to completely verify even a medium-sized real-time system is too large to even consider. However, if we take testing in consideration when designing systems by avoiding mechanisms that result in unnecessary test complexity, we believe that it is possible to construct test-case generation methods that exploit this to give confidence in any given degree of correctness while not loosing the desired properties of flexible real-time systems.

One may ask if testing is a useful technique for verifying the correctness of real-time systems in safety-critical environments. In such environments, it can be argued that functional and temporal correctness is so important that well-known, time-triggered systems with static off-line scheduling and formal proofs should always be used to guarantee timeliness. It is true that static architectures are more suitable for safety-critical environments. However, we believe there are many application areas of dependable systems where timeliness is important, but where flexibility and performance requirements call for a flexible, event-triggered architecture.

5.2 Contributions

This article presents an analysis of approaches for generating test cases for testing of time constraints. All encountered methods for testing of time constraints seems to be based on some structured or formal specification. Different specification notations give different advantages but also limitations and problems. Methods based on different notations are analyzed and their properties evaluated from the perspective of testing timeliness in distributed event-triggered real-time systems.

A classification of test-case generation methods relating to testing of temporal properties has been presented, complementing the classification of test methods presented by Laprie et al. [Lap94]. The classification is general enough so that all methods that aim to test time constraints should be easy to add as they arise.

The characteristics and contents of different test-cases produced by the different methods are described and evaluated against the requirement of our target model. The conclusion is that none of the methods consider internal states of the execution environment in the

test-cases, and hence, they allow temporal non-determinism. That is, the same test case may result in very different response-times in consecutive executions.

A related contribution is the identification of a need for special test criteria for selecting test cases for testing of real-time systems. None of the encountered selection criteria take advantage of constraints in the execution environment. Instead selection criteria are often based on the properties in the specification notation, e.g., transition coverage.

Furthermore, the methods has been classified according to their degree of automation and time-base, these attributes indicate which of the methods that are most probable of producing test suites with a realistic associated test-effort.

5.3 Future Research Directions

This section summarizes, in our opinion, the most important of the identified problems and proposes future research directions in this area that aim to solve these problems.

5.3.1 A Selective Test-Case Generation Method for Real-Time Systems

A test generation method for real-time systems must take time constraints into consideration, both for testing that input events are handled in a correct way and that the output of a system always is within a specified interval, i.e., timeliness. To verify the latter property, we believe that testing must be conducted on two levels. On a higher level, the tested event scenario must cause the worst-case internal behavior in blocking, synchronization, and arrival time of events. On a lower level the input parameters and data from shared resources must cause the worst-case execution time of transactions.

Our analysis has concluded that existing methods for test-case generation produce either event sequences or test data, but there exist very few methods that can supply both. An interesting problem for research is to combine a specification-based testing technique, where event sequences and their time constraints are generated from a formal specification, with a temporal behavior enforcing testing technique, such as described in section 4.5.

Furthermore, the test cases produced by the method should support test execution under specific internal conditions, e.g., by providing the internal state from where test execution begins (cf. [Nil99]).

5.3.2 A Test Coverage Criterion for Testing of Timeliness

No work encountered in the survey takes advantage of constraints in the execution environment for limiting the number of test cases. We believe that it is possible to eliminate many redundant test cases by taking such constraints into consideration. Hence, a future work is to further refine these ideas into more formal coverage criteria for testing timeliness in real-time systems.

Preferably, this should result in a hierarchy of coverage criteria attaining increasing degrees of test coverage. Such criteria must be evaluated against other approaches on real-life applications in order to determine it the criteria ensures high quality applications.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Bei90] B. Beizer. *Software Testing Techniques.* Von Nostrand Reinhold, 1990.

[BFM97] V. Braberman, M. Felder, and M. Marré. Testing timing behavior of real-time software. International Software Quality Week, 1997.

[BJ00] B. Bereza-Jarocinski. Automated testing in daily build. Technical Report ISSN 1493-6444, Swedish Engineering Industries, 2000.

[BMA99] R. Birgisson, J. Mellin, and S. F. Andler. Bounds on test effort for event-triggered real-time systems. In *In Proc. 6th Int'l Conference on Real-Time Computing, Systems and Applications (RTCSA'99)*, pages 212–215. Department of Computer Science, University of Skövde, December 1999.

[CL97] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *Proceedings of the Third International Workshop on Object-Oriented Real-Time Dependable Systems*, Newport Beach, California, February 1997.

[COG98] R. Cardell-Oliver and T. Glover. A practical and complete algorithm for testing real-time systems. *Lecture Notes in Computer Science*, 1486:251–261, 1998.

[CZ91] R. Cleveland and A. E. Zwarico. A theory of testing for real-time. *IEEE Computer Society Press*, pages 110–119, 1991.

[FBK+91] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on software engineering*, 17(6):591–603, june 1991.

[Kon95] O. Koné. Designing tests for time dependent systems. Seoul, South Korea, 1995. IFIP International conference on Computer Communications.

[KZF+91] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schütz. An engineering approach to hard real-time system design. pages 166–188, Milano, Italy, 1991. In proc. of the Third European Software Engineering Conference, ESEC ' 1991.

[Lap94] J. Laprie, editor. *Dependability : Basic Concepts and Terminology.* Springer-Verlag for IFIP WG 10.4, August 1994.

[LC97] P. Laurencot and R Castanet. Integration of time in canonical testers for real-time systems. California, 1997. 1997 Workshop on Object-Oriented Real-time dependable Systems, IEEE Computer Society Press.

[Mel98] J. Mellin. Supporting system level testing of applications by active real-time databases. In Proc. 2nd Int'l Workshop on Active, Real-Time, and Temporal Databases, ARTDB-97, number 1553 in LNCS. Springer-Verlag., 1998.

[MMM95] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real-time systems from logic specifications. *ACM Transactions on Computer Systems*, 4(13):365–398, Nov 1995.

[MP90] S. Morasca and M. Pezze. Using high level petri-nets for testing concurrent and real-time system. *Real-Time Systems : Theory and Applications*, pages 119–131, 1990. Amsterdam North-Holland.

[MW98] F. Müeller and J. Wegener. A comparison of static analysis and evolutionary testing for the verification of timing constraints. 1998.

[Nil99] R. Nilsson. Automated test case execution for real-time systems that are constrained for improved testability. Technical Report HS-IDA-EA-99-118, Department of Computer Science, University of Skövde, 1999.

[Nil00] R. Nilsson. Automated selective test case generation methods for real-time systems. Master's thesis, University of Skövde, September 2000.

[PF99a] E. Petitjean and H. Fochal. A realistic architecture for timed testing. In *Proc. of Fifth IEEE International Conference on Engineering of Complex Computer Systems*, USA, Las Vegas, October 1999.

[PF99b] S. M. Petters and G. Färber. Making worst case execution time analysis for hard real-time tasks on state of the art processors feasible. Hong Kong, 1999. RTCSA99.

[PS91] P. P. Puschner and A. V. Schedl. Computing maximum task execution times - a graph based approach. 1991. Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

[Sch93] W. Schütz. *The Testability of Distributed Real-Time Systems.* Kluwer Academic Publishers, 1993.

[Sch94] W. Schütz. Fundamental issues in testing distributed real-time systems. *Real-Time Systems*, 7(2):129–157, September 1994.

[SMM00] P. SanPietro, A. Morzenti, and S. Morasca. Generation of execution sequences for modular time critical systems. *IEEE Transactions on Software Engineering*, 26(2):128–149, feb 2000.

[TH99] H. Thane and H. Hansson. Towards deterministic testing of distributed real-time systems. Swedish National Real-Time Conference SNART'99, August 1999.

[WSJE97] J. Wegener, H. H. StHammer, B. F. Jones, and D. E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6(2):127–135, 1997.