Towards a Framework for Automated Testing of Transaction-Based Real-Time Systems*

Robert Nilsson Sten F. Andler Jonas Mellin

Department of Compter Science
University of Skövde
Box 408, 541 28 Skövde, Sweden
phone: +46(0)500-448370, fax: +46(0)500-448399
email: {robert,sten,jonas}@ida.his.se

Abstract

In real-time systems, 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 of real-time applications 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. We propose a framework for testing of transaction-based real-time systems using automatic test-case generation and execution techniques. An overview of previously proposed methods for generating test cases for testing of temporal properties of real-time systems is also presented.

Keywords: Testing, Event-Triggered, Real-time systems, Test-case generation, Timeliness

1 Introduction

Modern real-time systems tend to be increasingly complex. Moreover, real-time systems generally must be dependable as they often operate in safety or business critical environments. These characteristics imply that there is a need for rigorous verification methods to detect errors that arise from temporal faults [1]. 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 ad-hoc. *Testing* is a method to dynamically verify software by execution in order to detect errors and failures. *Errors* are the internal states in application software which may lead to externally observable *failures* [1]. *Test-case generation* is the process of selectively generating test cases that exercise system behaviors likely to reveal such errors. Test-case generation is based on application knowledge such as specification models, code structure, and system design. This information is used to selectively generate sets of test cases, sometimes referred to as *test suites*, that have high probability of revealing specific classes of errors or deviations from the expected behavior [1]. An *execution environment*, in this context, is the architecture of the system in which the tested applications run. This includes real-time operating system services, communication primitives, scheduling algorithms and properties of the underlying hardware. We suggest that properties of the execution environment be considered in the test-case generation process. 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, executed and observed.

This paper focuses on generation and execution of test cases for event-triggered real-time systems. Real-time systems are often concurrent; this complicates generation and selection of suitable test cases as system behavior is dependent on the non-deterministic order in which tasks execute, e.g. due to varying execution times. The event-triggered paradigm of real-time systems design greatly complicates testing since the behavior of the controlled environment continuously influence the execution order trough sporadic requests [2]. We demonstrate how existing test-case generation methods take these factors into consideration for testing of *timeliness*, which is the ability of the system to meet its time constraints. Further, we present work in progress on a testing framework for

*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.

This article was published at RTCSA 2002, part of the material was also presented at last ARTES graduate student conference. New contributions regarding test-case specifications and execution is found in section 3

testing of event-triggered real-time applications implemented as transaction processing systems.

2 Target system model

The classical approach for achieving temporal correctness in real-time systems is off-line scheduling analysis with full a priori knowledge of resource requirements and time constraints. Scheduling analysis is typically performed in systems with static, or fixed priority, scheduling schemes to guarantee that deadlines are met for worst-case behaviors. Because of variations in release times and varying execution times in such system, there is still a need for rigorous testing (see for example Thane [3]). This kind of systems will be referred to as *time triggered* [4] in this article. The penalty of having time-triggered systems is that utilization, flexibility and extendibility suffer. For example extendibility is degraded in the sense that when some system parameters change (for example the execution time of some task) the whole system needs to be reanalyzed, re-scheduled and re-tested.

For applications where these penalties cannot be accepted the alternative is to have systems that are dynamically scheduled (using EDF or similar deadline scheduling approaches [5]) and whose behavior is determined by the current state of the system and the incoming requests. These systems are referred to as *event triggered* [4]. Event-triggered systems do not suffer from the penalties described above; instead, they impose problems to perform a complete off-line scheduling analysis, and hence, testing must be used to build confidence in their temporal correctness. Unfortunately, testing of event-triggered systems has been shown to be much harder than testing of their time-triggered counterparts [2]. A reason for the difference is that static schedules repeat themselves) whereas dynamic scheduling schemes often do not. A related reason is that tasks in time-triggered systems are released at well-defined points in time, whereas tasks in event-triggered systems are triggered by the sporadic arrival of events. Typical for an event-triggered application is some core functionality with associated hard time constraints and tasks that are executed to increase quality of service. For example, consider an air-traffic control system or a monitoring system of a chemical plant.

3 Testing framework for event-triggered real-time systems

The approach presented here aims at testing event-triggered systems. For increased testability we propose a framework where event-triggered applications are implemented on top of a transaction processing system, which provide resource locking, ECA-rules and event filtering, e.g., a real-time database. There are several benefits of using such a framework apart from the testability issue, e.g., synchronization and data management are supported. The testability in this framework is increased by the properties associated with transactions, e.g. isolation and atomicity properties imply that concurrently executing transactions will interact in well-defined ways. When testing an event-triggered application we would like to concentrate our test efforts on guaranteeing that the critical transactions meet their deadlines. A problem when testing real-time systems are the repeatability issues related to the non-determinism caused by varying execution orders and execution times of transactions. This leads to problems in stating that a particular behavior has been sufficiently tested. Most test-case generation methods for testing real-time systems aim at generating *event-sequences* i.e. sequences of events with associated relative time-stamps. However, the system-wide state in which these inputs occur is seldom considered. This implies that the same event-sequences have to be run multiple times to gain confidence on the system behavior (hoping that some particular execution order will occur). Consider the simple example when two transactions are executed on the same processor and in 70% of all cases the transactions are scheduled in one particular order. To detect an error when the reversed order occurs - the test case must be executed at least 13 times to have 99% probability of revealing the error. The alternative is to have some means of controlling the execution environment so that a particular execution order occurs. However, when performing timeliness testing (and time-dependent testing) probes cannot be inserted in the execution environment during test execution and then removed, as they will change the temporal behavior of the system. An approach to increase the value of test-case executions is to include information about the system-wide state in each test case, and hence, to achieve a more deterministic test execution. By using such test cases, the problem of finding timeliness violations is transformed into the problem of selecting (and executing) the right test cases. Adding system state-information to test cases has the consequence of dramatically increasing the total number of test cases possible to run in the system. This can be compensated for by adding constraints on the system behavior as described by Mellin et al. [6] [7]. In particular, the inclusion of designated preemption points reduce the number of possible behaviors of the system significantly and simplifies test execution. Test-case gen-

eration and selection can be automated, given structured or formal specification of the system and its execution environment. Test-case execution can also be automated, given a uniform test-framework and well-specified test cases. In the framework presented here, a non-intrusive event monitoring facility is provided by the underlying transaction processing system, and since the monitor will remain in the system during normal operation the temporal behavior is preserved. Further, all instrumentation to setup a test case is done prior to the actual test execution.

Test-case specification

Test cases are defined to contain an initial state description, an event sequence and an expected outcome. In a concurrent real-time system, the state is defined by the state of all currently active tasks (including their point in execution and used data objects), and hence, a sequence of input events is not sufficient to deterministically bring the system into a specified state. Here, we aim to bring the system to a specified initial state without explicitly defining a sequence of events that may lead to it. A test case for a real-time system should at least include an event sequence with associated time-stamps (or a script which implement such sequence). In simple control applications this may be enough, but in a typical event-triggered application, the events arrives with parameters that are fed to the processing task. The content of these parameters influence the execution behavior of the triggered task and need to be included in the test case. In this work, we assume that event-parameters are passed with the events and argue that their values should be part of the generated event-sequences (this is not assumed in the related work section). The expected outcome of a test case is a definition of the correct system behavior acquired by applying an event sequence on a system executing from the specified initial state. In a real-time system a time constraint is associated with each outcome. In this work, the expected results are limited to the time constraints, parameters of output events and changes to data objects in the underlying database.

Test-case execution scheme

The first step in executing a test case is to force the system into the specified initial state. Under the strict pessimistic concurrency control policy, the transactions that are active have also acquired locks on all required resources. Hence, serial execution up to the required transaction states are possible while no time dependencies are reflected in the data objects (in Figure 1; transactions T1, T2, and T3 are executed sequentially to specified preemption points). Obviously, the state of the resources will influence the execution time of the individual transactions; hence, classes of relevant initial states of resources should be selected. Once each active task is executed to the specified preemption point, they are combined to a concurrent initial state. Pending transaction tasks that have not been allowed to execute yet, but is part of the initial state specification, are added to the initial state. The dynamic scheduler is assumed to update its schedule as a direct response of incoming events and hence, one must allow it to build an initial schedule before the test execution begins. At this point the specified initial state has been enforced and can be saved so that multiple test cases with different event-parameters can be tested from this state (increasing performance of the testing). One possibility of saving the state is to have some means of replication of the database-node. In the second phase the system is executed from the initial state and inputs are externally injected during the execution according to the event-sequence part of the test-case specification. The behavior of the system is continuously monitored and logged using the event monitoring facilities present in the database architecture. The event logs are used after test case execution to verify that the correct behavior was observed.

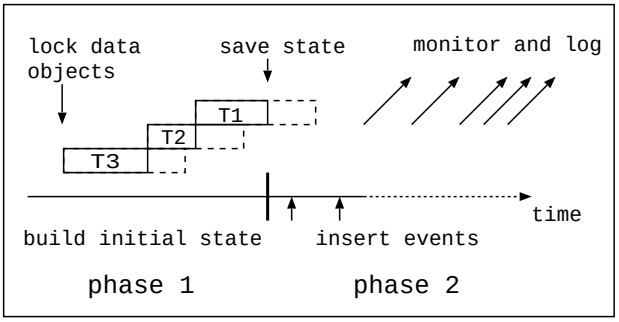

Figure 1: Test execution phases

4 Related work

Clarke and Lee [8] introduce a framework for testing time constraints of real-time systems. Time constraints are specified in a constraint graph, and the specification model is process algebra. This method is interesting for 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 time constraints on the input to the system. No method for generation of test cases for testing of time constraints on the output sig-

nals is presented.

There are a number of methods for testing time constraints based on timed automata models. In general, these methods supply interesting event sequences for testing time constraints. However, they do not consider the variations in behavior imposed by the execution environment. One such method is presented by Petitjean and Fochal [9] who propose a test architecture for testing of timed systems. A timed system is described as a timed automaton where time constraints can be expressed as a region graph. The specification model is the timed- I/O automaton theory proposed by Alur and Dill [10]. Another approach was presented by En-Nouaary et al [11]. Their approach exploits a sampling algorithm using grid-automata and non-deterministic finite-state machines as intermediate representations in the test generation process to reduce the test-effort and maintain test-coverage. However, the complete-testing assumption used in their work does not hold for the type of implementations we consider. Nielsen and Skou [12] use a subclass of timed automata called ERA (event recording automata model) for specifying time dependent applications. The main contribution with their method is a coarser equivalence partitioning of temporal behaviors over the time constraints expressed in the specification. However, here it is inconclusive if the degree of test coverage can be maintained. Laurencot and Castanet [13] recapitulate formal modeling languages which can be used to test real-time systems. The different specifications they compare are the automata of Alur and Dill [10], Timed Transition Models (e.g., [14]), and Extended Time Input Output State Machines. The method decomposes a timed formal model into sub-models for each timer and generates tests by traversing the sub-models. Cardell-Oliver and Glover [14] also starts the test generation process from a formal methods perspective. This requires that the system under test can be viewed as a deterministic finite state automaton. In our target model it is hard to construct such an abstraction that captures all aspects of the temporal behavior for the entire system (due to e.g. data-dependencies and race conditions).

Mandrioli et al. [15] suggest a method for testing of real-time systems based on specifications of system behavior in the temporal logic language TRIO. The elements of 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. For system level testing the number and complexity of logic formulas will increase rapidly. In a more recent work, the authors [16] expand their previous results to incorporate a high-level, structured specifications that can be combined with the low-level specifications proposed earlier for testing modular software applications. This refinement is valuable, but the result is only event sequences, and still no consideration is taken to internal states when events occur.

Morasca and Pezze [17] propose a method for testing concurrent and real-time systems that uses high-level Petri-nets for specification and implementation. 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 are not Petri-nets. Braberman et al. [18] also introduce a method for generating test cases for real-time systems based on timed Petri-net designs. The method uses the design notation SA/SD-RT for specifying the behavior of concurrent real-time systems. The design specification is translated to a timed Petri-net notation from which a timed reachability tree can be derived. Listed future work is to investigate how architectural information, such as scheduling policies, can be taken into account during test-case generation. We believe that the method by Braberman et al. is one of the most promising methods for our purpose since it has the potential to generate both initial state specifications and event-sequences.

Raymond et al. [19] present a method for generating event sequences for reactive systems. Their approach models environmental constraints and test requirements as external observers. However, the method does neither explicitly consider test case generation for testing temporal constraints nor the internal execution behavior of the tested system.

There exist several methods for finding input data that cause the maximum execution time of a task, e.g., static analysis of code and measurements. Wegener et al.[20] propose a method that uses genetic algorithms to generate test data for testing temporal properties of real-time systems. For a thorough analysis of the related work see [21]

5 Conclusions

In this article we briefly outlined a target system domain and stressed the necessity of verifying temporal correctness in that domain. We suggested a platform (and paradigm) for implementing event-triggered applications that increase testability. Problems with current testing methods for event-triggered real-time systems has been outlined and a framework for executing test cases that contain information about the initial system-wide state has been introduced. By incorporating state information in automatically gener-

ated test cases, we avoid the need of running test cases multiple times to find a faulty temporal behavior, and thus, we potentially decrease the test-effort while increasing confidence in temporal correctness.

We argued that by using a transaction-based system as a platform for implementing event-triggered applications, it is possible to exploit the isolation property and enforce system states by running each transaction to a specified designated preemption point. This can in turn be used for gaining control over the test execution and fully automate testing of timeliness.

We have presented an overview of related work, namely frameworks and methods for testing temporal properties of real-time applications, and highlighted their applicability for testing of event-triggered systems. Work is currently in progress to evaluate some of these methods and their associated tools and specification models for automatic test-case generation purposes in the presented framework. Ongoing research also includes further refinement and evaluation of the test-case execution scheme.

References

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

[2] W. Schütz, "Fundamental issues in testing distributed real-time systems," *Real-Time Systems*, vol. 7, no. 2, pp. 129–157, September 1994.

[3] H. Thane, *Monitoring, Testing and Debugging of Distributed Real-Time Systems*, Ph.D. thesis, Royal Institute of Technology. KTH, Stockholm, Sweden, 2000.

[4] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schütz, "An engineering approach to hard real-time system design," in *Proceedings of the Third European Software Engineering Conference*, Milano, Italy, 1991, pp. 166–188.

[5] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline scheduling for real-time systems*, Kluwer academic publishers, 1998.

[6] J. Mellin, "Supporting system level testing of applications by active real-time databases," in *In Proceedings 2nd International Workshop on Active, Real-Time, and Temporal Databases, ARTDB-97, number 1553 in LNCS. Springer-Verlag.*, 1998.

[7] 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)*, Hong-Kong, December 1999, pp. 212–215, IEEE Computer Society Press.

[8] 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.

[9] 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.

[10] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[11] R. En-Nouaary, Khendek F. Dssouli, and A. Elqortobi, "Timed test cases generation generation based on state characterization technique," in *Proceeding of the 19th IEEE Real-Time Systems Symposium (RTSS98)*, Madrid, Spain, December 1998.

[12] B Nielsen and A. Skou, "Automated test generation from timed automata," in *Proceedings of the 21st IEEE Real-Time Systems Symposium*, Walt Disney World, Orlando, Florida, 2000, IEEE.

[13] P. Laurencot and R. Castanet, "Integration of time in canonical testers for real-time systems," in *Proceedings of Int. Workshop on Object-Oriented Real-Time Dependable Systems*, California, 1997, IEEE Computer Society Press.

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

[15] D. Mandrioli, S. Morasca, and A. Morzenti, "Generating test cases for real-time systems from logic specifications," *ACM Transactions on Computer Systems*, vol. 4, no. 13, pp. 365–398, Nov. 1995.

[16] P. SanPietro, A. Morzenti, and S. Morasca, "Generation of execution sequences for modular time critical systems," *IEEE Transactions on Software Engineering*, vol. 26, no. 2, pp. 128–149, feb 2000.

[17] S. Morasca and M. Pezze, "Using high level Petri-nets for testing concurrent and real-time systems," *Real-Time Systems: Theory and Applications*, pp. 119–131, 1990, Amsterdam North-Holland.

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

[19] Pascal Raymond, Xavier Nicollin, Nicolas Halbwachs, and Daniel Weber, "Automatic testing of reactive systems," in *Proceeding of the 19th IEEE Real-Time Systems Symposium (RTSS98)*, 1998.

[20] J. Wegener, H. H. StHammer, B. F. Jones, and D. E. Eyres, "Testing real-time systems using genetic algorithms," *Software Quality Journal*, vol. 6, no. 2, pp. 127–135, 1997.

[21] R. Nilsson, "Automated selective test case generation methods for real-time systems," M.S. thesis, University of Skövde, September 2000.