Six Issues in Testing Event-Triggered Real-Time Systems

Birgitta Lindström
University of Skövde
Box 408, 541 28 Skövde, Sweden
E-mail: birgitta.lindstrom@his.se

Robert Nilsson
University of Skövde
Box 408, 541 28 Skövde, Sweden
E-mail: dr.robert.nilsson@gmail.com

AnnMarie Ericsson
University of Skövde
Box 408, 541 28 Skövde, Sweden
E-mail: annmarie.ericsson@his.se

Mats Grindal
University of Skövde
Box 408, 541 28 Skövde, Sweden
E-mail: mats.grindal@his.se

Sten F. Andler
University of Skövde
Box 408, 541 28 Skövde, Sweden
E-mail: sten.f.andler@his.se

Bengt Eftring
University of Skövde
Box 408, 541 28 Skövde, Sweden
E-mail: bengt.eftring@his.se

Jeff Offutt
George Mason University
Fairfax, VA 22030, USA
E-mail: offutt@gmu.edu

Abstract

Verification of real-time systems is a complex task, with problems coming from issues like concurrency. A previous paper suggested dealing with these problems by using a time-triggered design, which gives good support both for testing and formal analysis. However, a time-triggered solution is not always feasible and an event-triggered design is needed. Event-triggered systems are far more difficult to test than time-triggered systems.

This paper revisits previously identified testing problems from a new perspective and identifies additional problems for event-triggered systems. The paper also presents an approach to deal with these problems. The TETReS project assumes a model-driven development process. We combine research within three different fields: (i) *transformation of rule sets between timed automata specifications and ECA rules with maintained semantics,* (ii) *increasing testability in event-triggered system, and* (iii) *development of test case generation methods for event-triggered systems.*

1. Introduction

In 1994 Schütz wrote a paper describing several problems for testing of real-time systems [51]. That paper led to an international research collaboration with multiple research projects pursued by the TETReS group at the University of Skövde, Sweden and George Mason University, USA. This paper summarizes progress on several problems described by Schütz with particular attention to real-time systems with event-triggered semantics. We also discuss progress on other problems.

The TETReS research projects cover many problems in testing event-triggered real-time systems. Combining concepts from testing and formal verification is increasing collaboration between normally separated areas, enhancing synergy. All of these projects are increasing software quality by addressing problems described here.

Section 2 describes the background and introduces testing and real-time systems concepts. Section 3 summarizes the problems described by Schütz. This overview is followed by a detailed discussion of which problems have a greater impact on testability when the tested system is event-triggered. Some of these problems are new to this paper. Section 4 describes how our research targets the problems. Finally, Section 5 discuss the value of the combined research in the TETReS project.

2. Background

The main contributions of this paper lie in the intersection of software testing and real-time system development. Concepts from these fields are introduced in the following subsections.

2.1. Testing Concepts

Software testing exercises a software program with input values [34]. Embedded real-time software has strict reliability requirements and maintenance is expensive, thus Developers spend a lot of effort during testing.

Testing is used to check for conformance and finding faults [34]. *Conformance testing* tries to verify that an implementation corresponds to its specification, and *fault-finding testing* tries to find implementation and design faults. Common to both purposes is the underlying desire to gain confidence in the system behavior.

A fundamental limitation of software testing is that even small programs cannot be tested exhaustively. The reason is that the number of program paths grows quickly with nested loops and branch statements [7] and for concurrent programs, the number of behaviors increase even quicker due to interactions between concurrent tasks. Consequently, testing cannot show the absence of faults (to prove system correctness), it can only show the presence of faults [16].

As a complement to testing, formal verification can be used. Figure 1 shows the relation between formal verification and testing in a development project.

This paper assumes a model driven development (see figure 2). This does not mean that all aspects of a system design or implementation must be formalized and expressed in a model. However, there are several advantages of using a model driven approach.

One advantage is that model driven development allows test cases to be automatically generated. Hence, effort is moved from generating specific test cases to building a model. Another advantage is that some properties of the model can be automatically verified. Models can be built before the system is implemented and used as a reference or part of a specification of the system. It is also possible to build the model only for testing and verification purposes, after or concurrently with development. Often, design errors and ambiguities in a specification can be detected when creating models. A *test case* is a set of test inputs, execution conditions, and expected results developed for a particular objective,

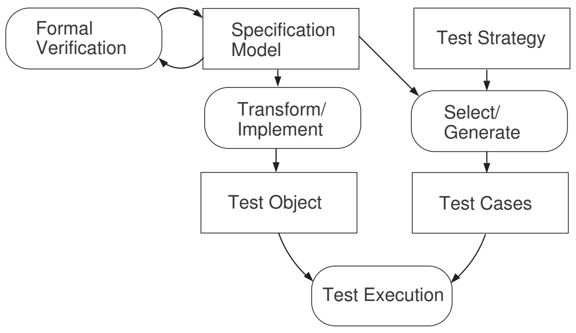

Figure 1. Overview of Testing activities

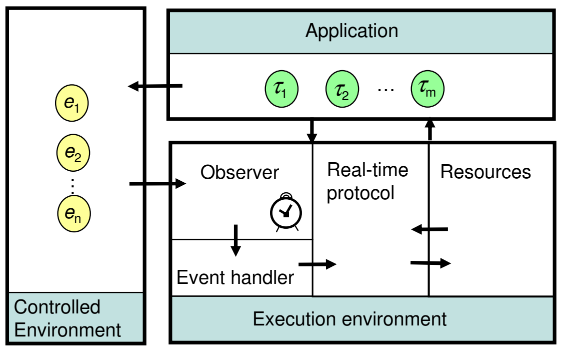

Figure 2. Overview of a real-time system

such as to exercise a particular program path or to verify compliance with a specific requirement [29]. A *test object* is the system (or portion) that is addressed by the test case. Hence, a test object may consist of pure software or a combination of hardware and software. Further, a test object may consist of a single software module or a set of software modules.

2.2. Real-time Systems

Figure 2 depicts an overview of a real-time system. Real-time applications are often modeled as a set of tasks (pieces of sequential code) that compete for system resources such as processor-time, shared data, etc. The response times of the tasks depend on the order in which they are scheduled to execute. This is controlled by real-time protocols such as scheduling and concurrency control protocols and properties of the execution environment such as the real-time operating system, programming language and hardware. Real-time systems typically interact with other (sub-)systems and processes in the physical world, the *environment* of the real-time system. For example, the environment of a real-time system that controls a robot arm may contain items on a conveyor belt and messages from other robot control systems on the same production line.

Real-time software observes the environment periodically or in response to some triggering event. In particular, *periodic* tasks are activated with fixed inter-arrival times. Thus, all the points in time when such tasks are activated are known beforehand. In contrast, *sporadic* tasks are activated by events from the environment. Assumptions about the event occurrence patterns, such as *minimum inter-arrival times*, are used in analysis. Each real-time task must meet a set of time constraints on its activation and completion. A *deadline* is a time constraint on the response time of a task. A system in which all timing constrains are met is *timely* [49].

Kopetz and Verissimo [31] describe two ways to design real-time systems: time-triggered and event-triggered. The primary difference is that time-triggered systems only observe the environment and perform actions at pre-specified points in time, whereas event-triggered systems detect and act in response to events in the environment.

A pure *time-triggered* real-time system operates with a cyclic behavior. At an observation point, the system observes all events that have occurred since the last observation point. Tasks that correspond to the occurred events are executed during the next time period in a pre-scheduled order. The computations that are scheduled in one period must finish before the next period starts regardless of which events have been observed. This implies that all tasks that can be executed in a period must be known beforehand.

An *event-triggered* real-time system reacts to an event by reconsidering the current schedule. A task corresponding to the occurred event is identified. A decision is made based on the scheduling policy, current state of the system, resource requirements, and task priorities. The task may be dropped, scheduled for execution some time in the future, or started immediately by preemption of the currently executing task. The priority of an individual task can, for example, be decided by its period, criticality or urgency.

A time-triggered design is based on resource adequacy and therefore requires full knowledge about resource usage. Moreover, resource adequacy means that there will always be enough resources to meet the timing requirements. Hence, if the difference between worst case and average case resource demands is large, then a time-triggered solution will lead to low resource utilization. For example, assume a system with several sporadic events that occur seldom but have short, hard deadlines (such as alarm signals or sudden requests for evasive action). A time-triggered system would have to execute a periodic task that polls the environment frequently enough to be able to detect and respond to such events in a timely manner. An event-triggered system would only have to execute a sporadic task as a response to the real occurrences of the event, which leaves computation resources free to be used for other purposes. This is why event-triggered designs are sometimes preferred.

3. Issues in Testing Real-time Systems

In 1993, Schütz presented problems in testing real-time systems [50]. Schütz focused on issues related to testing time-triggered real-time systems [32]. The issues are organization, representativity, host/target approach, environment simulation, observability, and reproducibility [50]. This paper revisits Schütz's problems with an event-triggered perspective and complements these with problems that arise when testing event-triggered systems. Some of Schütz's problems are unaffected by the choice of paradigm. Problems that have been addressed by the TETReS project are described in Section 3.1.

The first problem is *organization*. Testing activities must be organized into well-defined test phases that should harmonize with the design phases of the development methodology. Moreover, real-time systems are inherently complex in comparison with non-real-time systems due to time constraints and interaction with hardware. These problems may impact the organization of testing as well as development [50].

The second problem is *representativity*. Exhaustive testing is in general impossible; partly because of large input domains, partly because of insufficient knowledge about the system's environment, and partly because of the number of execution paths in the implementation of the system. The consequence is that the system can only be tested by a sample of potential test inputs. Hence, it is very important that this sample is representative with respect to real-world scenarios [50].

The third problem is the *host/target approach*. Real-time systems are usually developed and verified on two types of computer systems, the host and the target. Real-time systems are often embedded with low support for testing on the target system. Moreover, testing on the target system might be hazardous with respect to human injures or material damage. Therefore, testing is to a large extent performed on the host. The drawback is, of course, that the behavior may differ between host and target. Therefore the final test phases must be performed on the target system.

The fourth problem is *environment simulation*. Before the final field tests, the target system is usually tested in a safe, or controlled, environment. These tests can be open-loop or closed-loop. In *open-loop testing*, test cases are constructed and then executed on the target system. In *closed-loop testing*, the real-time system is connected to a simulation of the system's environment. The environment simulator reacts to the output of the tested real-time system according to pre-defined rules and, based on that output, it calculates new sets of stimuli for the tested control system. A test case in closed-loop testing can consist of parameters that influence the behavior of the simulator in different ways (for example, change the wind-conditions in the environment of an aircraft).

The correctness of an environment simulator is important for the confidence in correctness of the tested system. If the tested system has time constraints, then the environment simulator needs to supply inputs to the tested system in a timely and realistic manner. The problem is elevated in event-triggered designs as opposed to time-triggered because the environment simulator must be more accurate to trigger events at the correct times and in the right order.

3.1. Problems for Event-Triggered Real-time Systems

Testing an event-triggered system is harder than testing a time-triggered system [50]. This section describes six problems that have been addressed by the TETReS project. We explain the impacts of the event-triggered design. The two remaining problems identified by Schütz, i.e., observability and reproducibility, are covered by problem 1.

Problem 1 : Test Execution, Monitor and Control. Effective test execution, either manual or automatic, requires the test object to satisfy some basic requirements. Most test cases have specific goals. Thus, test cases should be executed exactly as described and responses from the test object should be captured. This translates into the three properties controllability, observability, and reproducibility [50].

Controllability is the ability to (re)execute selected test cases. This includes both choosing the state to start executing from and injecting sequences of events at specified points in time. Enforcing a starting state is significantly easier with a time-triggered design because their cyclic behavior ensures they always return to their initial state. Injecting events at specified points in time is also much easier with a time-triggered solution due to its coarse clock granularity. The finer the clock granularity is, the higher is the required time precision for the injection.

Observability is the ability to observe internal and external system behavior during test execution. Low observability makes it difficult to distinguish between correct and erroneous behavior. Traditional techniques to achieve observability such as auxiliary outputs are less useful for real-time systems since these techniques introduce a probe effect. The *probe effect* is described as a phenomenon where the behavior of a system may be affected due to the attempt of observing it [19]. For example, extra statements added to produce auxiliary output will add some execution time, which can affect the response time. Changing the timing behavior may even change the execution order. This, in turn, can sometimes alter the final results produced. Therefore testers must consider the probe effect when carrying out performance measurements or testing timeliness [50].

Event-triggered systems are more prone to the probe effect than time-triggered systems. An event-triggered system will deliver the result as soon it is ready. Thus, even small changes can introduce probe effects. In time-triggered systems the time the final result is delivered is predefined. If the probe effect is smaller than the time available to the predefined time of result delivery, the probe effect will not cause any detectable consequence.

Reproducibility is the property that the system repeatedly exhibits identical behavior when stimulated with the same test case. The is hard to achieve in event-triggered systems since the system behavior partly depends on elements that have not been expressed explicitly as an input to the system. That is, the behavior is non-deterministic when just the software is considered. Hence, what we judge to be a repeated test case might lead to different behaviors due to elements that testers cannot control, including hardware components.

Problem 2 : Input Space. For non-real-time systems, the input space is the set of all possible values that can be provided to the system through any of its interfaces. During testing, each element in the input space constitutes a potential test case input. A fundamental problem in testing is that most real-life systems have huge input spaces. These are often orders of magnitude larger than the number of test cases possible to execute in the time allocated for testing.

A common approach to this problem is to split the input space into partitions of values where all values in a partition are assumed to have similar behavior. One or more samples are selected from each partition. Examples of test methods based on partition testing are Equivalence Partitioning [39] and Boundary Value Analysis [39].

The behavior of a real-time system depends both on the values when these values are observed by the system. The input space thus contains a temporal dimension in addition to the value domain.

In time-triggered real-time systems, the system observes new inputs at predefined, periodic, points in time, so called *observation points*. Events in the system environment that lead to inputs to the system will be observed by the system at the next observation point. The observation points thus result in a natural partition of the temporal domain since the result will be exactly the same regardless of when the event occurred, as long as it is observed at the same observation point.

In contrast to time-triggered real-time systems, the input spaces of event-triggered real-time systems do not have natural partitions of the temporal domain. An event may influence the behavior of the system at any point in time. The lack of natural partitions has several consequences for the tester. It produces a larger input space than in the time-triggered case, it makes controllability more difficult and it places higher demands on an environment simulator.

Problem 3 : Formal Verification. To enhance the level of analysis and prevent faults from being introduced early in development, formal methods can be used to analyze critical parts of a system [9]. Timeliness typically needs to be both tested and formally verified. Testing and formal verification are therefore viewed as both complementary and necessary activities. Formal verification and analysis are necessary since testing is applied only to a sample of all behaviors. Hence, applications must be thoroughly analyzed and time, in the form of execution times and deadlines, must be included in the analysis.

Model checking [17, 47] is a powerful technique for automatic formal verification. This technique is used in tools for real-time systems modeled as timed automata such as UPPAAL [35] and KRONOS [15]. The timed-automaton model enables analysis of reachability properties and global invariants, which makes them especially useful to verify timeliness. In timed-automata CASE tools, the computing power needed for automatic model checking grows exponentially with the number of states and clocks in the specification. The dynamic behavior of the event-triggered real-time system increases the risk of *state-space explosion* [20], that is, the problem of exponential growth of state space with respect to the size of the input model.

Semantic differences between a verified model and the real implementation might invalidate any proof. The risk for such a semantic gap increases rapidly with the flexibility and complexity in event-triggered systems, due to the state space explosion problem. This problem usually causes developers to simplify the model, by doing things like increasing the level of abstraction.

Problem 4 : Timeliness. Testing of timeliness is general in the sense that it applies to all system architectures and does not rely on the accuracy of models and estimations; instead the target system is executed and monitored so that faults leading to timing failures can be detected. Consequently, the goal of timeliness testing is to provoke the system to miss a deadline by selecting appropriate test inputs. In a time-triggered system, the worst case situation with respect to timeliness is when the highest load and worst case execution time for all involved activities occur. In such systems, testing of timeliness becomes equivalent to finding the input conditions that maximize resource consumption of each task. In an event-triggered system the problem is more complex. The worst case is a combination of current state (e.g., program counter and other information in the process control blocks (PCB)) and the time and order of input events (e.g., interrupt signals).

Most existing methods for timeliness testing generate test cases from abstract models of the software behavior. These models do not include resource requirements of tasks or the assumed behavior of the execution environment [44]. Scheduling theory tells us that it is difficult or impossible to characterize the worst case sequence of triggering events without considering the effect on the set of active tasks and real-time protocols [53]. Consequently, new test criteria and test case generation methods are needed that produce sequences of triggering events and use information of the execution environment to stress event-triggered real-time systems.

Problem 5 : Design Trade-offs. Section 2.2 describes the fundamental differences between time-triggered and event-triggered systems. An advantage of time-triggered systems, in particular when designing safety-critical systems, is predictability. Its cyclic behavior and static scheduling of CPU and other resources enhances predictability, especially in time. However, time-triggered systems are generally less efficient and hence more expensive than than the corresponding event-triggered system. This illustrates the general design dilemma of finding suitable trade-offs when two or more system properties conflict with each other. The problem is that most work on testability focus on time-triggered solutions and therefore, we know little about supporting testability in event-triggered systems. Not much is known about the relation between real-time system design decisions and testability. Hence, it is difficult to find the optimal trade-off between testability and efficiency as well as predictability and performance.

Problem 6 : Validation of Methods. Validating solutions to real-time problems present several problems. First, it is hard to find appropriate software objects for experimentation. Consider validation of a test case selection method for timeliness in distributed event-triggered systems. The study object needs to be distributed and event-triggered. Second, if a suitable study object is found, it may also be a big economic risk for a company to allow the researcher to try a previously untried solution or concept within its production. Third, if the researcher is allowed into the company, the company may restrict how the study is performed and how the results are disseminated.

Whenever a real-life study is performed, another problem is the representativity of the investigated settings. Without knowledge of the size and distribution of the goal population it is impossible to say how results generalize [36].

One consequence of these problems is that many suggested methods and solutions have not yet been sufficiently validated in practical settings. Another consequence is, for conducted studies, the difficulty of judging the results with respect to applicability.

4. Method

The following subsections describe several lines of research within the TETReS project. The research problems being addressed and the results thus far are explained.

4.1. Specification of reactive behavior

This section describes results on the formal verification problem in Section 3.1. Previous researchers suggested implementing event-triggered systems with event condition action (ECA) rules [48]. ECA systems execute an action in response to an event if a specified condition is true. The behavior of an ECA system is the same as an event-triggered system in which the system must react (execute action) to external stimuli (event occurrence), and the condition part is usually optional. The ECA rules are flexible in that the behavior can be changed simply by adding or removing a rule. ECA rules are currently used in various areas, such as active (real-time) databases ([46, 4, 12]), complex event processing systems ([5, 52, 55]) and the semantic web [1, 11].

It is hard to analyze the behavior of rule based systems because of run time interactions between rules. Executing one rule may trigger a new event or affect the outcome of condition evaluation of another rule. A chain of rules triggering other rules may cause an undesirable non-terminating behavior.

We use the development tool REX, designed for specifying rule based systems [18]. REX is built as a front end to the timed-automaton CASE-tool UPPAAL. Rules and requirement properties of the rule-based system are specified in the graphical user interface of REX. The specification and requirement properties are automatically transformed to UPPAAL. The rules are transformed to a timed-automaton specification and the requirement properties are transformed to CTL expressions that can be verified in the UPPAAL model-checker. Hence, the specified requirement properties of the set of rules are seamlessly verified by UPPAAL's timed-automata model-checker. REX hides the details about formal verification, which allows non-experts to use model-checking.

As noted in Section 3.1, state space explosion can be a problem when timed automata are used to formally verify event-triggered systems. The computing power needed for automatic model checking heavily depends on the number of states and clocks in the specification. However, as suggested by Hoare [27], large and complex projects can be split into small components that are designed and implemented independently. Moreover, as pointed out by Bowen and Hinchey [9], applying formal methods to all system aspects is both unnecessary and costly. It might, for example, be preferable to use the formal verification capability of UPPAAL for critical parts of the system and to use the simulator to verify less critical parts.

The goal of this research is to support development of predictable rule based systems. Given algorithms that transform rule sets between timed automata specifications and ECA rules with maintained semantics, a set of executable rules is implicitly formally analyzed in timed automata. This approach increases the ability to develop and maintain event-triggered systems built with ECA rules. Moreover, the high-level specification can be re-used for test case generation, thus supporting testing.

4.2. Improving Testability

This section describes results on the problems of test execution and design trade-off (1 and 5 from Section 3.1), and how we support testability in event-triggered real-time systems. Section 3.1 explains that an event-triggered system is harder to test than a corresponding time-triggered system [50, 8]. This because time-triggered systems have cyclic behavior and highly constrained *execution environments*. The execution environment is viewed as a set of hardware and software components that the system is built on top of (e.g., processors, task dispatchers, device drivers). The execution environment imposes constraints of the resources (e.g., communication bandwidth) and on the mechanism (e.g., data locking, task dispatching). This limits the state space as well as the number of execution orders.

The designers do not have to choose between using expensive time-triggered semantics and having no testability. We can combine properties of time-triggered and event-triggered systems and build event-triggered systems with higher testability than traditional event-triggered systems by introducing a set of constraints (see **C1** to **C3** below) on the execution environment [38, 8, 37]. The constraints that we investigate are proposed by Mellin [38].

C1–*An upper bound on the number of concurrently executing tasks of the same task type.* Increased concurrency can increase the number of execution orders, thereby increasing the number of potential behaviors with respect to time and order. It is possible to derive this bound from the minimum inter-arrival time and the maximum response time for the task type. To do this, we need a mechanism that guarantees them by doing things like filtering new tasks that arrive too early or aborting tasks that have not finished in time.

C2–*Designated preemption points*. Preemptions are delayed until the next preemption point. By only allowing preemptions at specific points, the number of execution orders is significantly reduced [37]. Nilsson, Andler and Mellin demonstrated how designated preemption points may support controllability [41]. Preemption points have been used before [6, 54], but we also require an upper bound on the number of preemptions. Hence, if this bound is smaller than the number of preemption points, a mechanism for handling the situation where a task has reached this upper bound is needed. This can be done by running the task in non-preemptive mode or allowing abortion if necessary.

C3–*Predefined observation points*. When an event occurs, the system is notified at the next observation point. With coarse observation granularity, the number of event sequences that the system can distinguish is decreased. The reason is that events occurring during the same interval are considered simultaneous. Hence, their ordering will not affect the behavior [37]. Moreover, coarse observation granularity enhances controllability. To execute a test case with a given event sequence, it is only necessary to input the events in given time intervals instead of specific points in time. Observation points are used in time-triggered, static systems [30] but can be introduced in dynamic systems without violating the dynamic semantics [37].

C1 to **C3** are currently being empirically evaluated We hope to show that it is possible to build event-triggered systems that can be tested for timeliness in a structured way.

4.3. Test case generation

Test case generation strategies are essential for effective and efficient testing. We target test generation in two ways. First, a family of general software testing methods is refined and evaluated for handling large input spaces and for testing correctness properties. This includes surveying the state-of-art for this type of testing and the state-of-practice of testing commercial systems. Second, a model-based testing method is developed for testing timeliness, a property that requires special consideration during testing and design.

4.3.1. Combination Strategies. This line of our research primarily focuses on the problem of large input spaces. A secondary focus is on validating the proposed methods through experiments in realistic settings and industrial case studies. As was described in section 3.1, problem 2, the input space of most real-life systems is enormous. *Combination strategies* is a class of test case selection methods that use combinatorial strategies to select test sets that have reasonable size. Our previous paper [26] surveys a large number of combination strategies.

A common property of all combination strategies is that they require an input parameter model. The *input parameter model* (IPM) is an abstract description of the input space of the test object. The IPM may also include representation of other properties of system under test, such as state. Any property that can be expressed in terms of input variables with values can be represented in an IPM. This makes combination strategies general enough so that combination strategies can be applied to different types of testing. For functional testing, including timeliness, a small set of normal and boundary values [3, 13, 14] can be used. A larger number of non-valid inputs may be used for robustness testing [33, 10]. A large number of valid values may be used for stability.

One challenge for the tester when using combination strategies is how to represent the input space in an IPM. Questions within this area include how to identify a suitable set of parameters, how to partition the values of each parameter, and how to handle conflicts in the input space, i.e., impossible or invalid combinations of parameter values. These questions have been addressed in our suggested input parameter modeling method [24]. It is based on the category partition method [45] and adapted for combination strategy specific features. Recent results also show that conflicts in the input space can be effectively and efficiently resolved later in the test case generation process. Thus, the tester need only to indicate potential conflicts in IPMs [25].

Another challenge for the tester is selecting which combination strategy to use. Grindal [21] describes a selection method that accounts for the priorities with respect to time, cost and quality of the current project and the importance of the system under test. The method is based on number of evaluation criteria [23], such as types of targeted faults, level of coverage, and size of the generated test suite.

The proposed methods for input parameter modeling [24], handling conflicts [25] and selecting combination strategies [21] are integrated into a complete testing process, which has been validated in a proof-of-concept study [22]. The study has been performed in three industrial settings and results show that the combination strategy testing process can successfully be applied to a large set of testing problems. Further, the results indicate that combination strategies, while providing a structured approach to testing and objective measurements of test quality, also are at least as effective and efficient as currently employed test methods.

4.3.2. Mutation-based Testing of Timeliness. This research focuses on effective testing of timeliness in event-triggered real-time systems. This also contributes to handling the large input space resulting from the time domain.

We have developed a framework for open-loop timeliness testing of real-time systems using mutation-based test techniques. This framework uses estimated temporal properties and resource requirement as specified in a formal model. As opposed to other approaches for timeliness testing, properties of the execution environment are part of the model from which test cases are generated. Hence, effects of scheduling and platform overheads can be captured and exploited for guiding automated test case generation.

Test criteria for timeliness and robustness are based on mutation operators [2]. These criteria allow us to automatically generate test cases as sequences of triggering events and specifications of dangerous execution orders. These tests stress some of the worst-case scenarios with respect to timeliness. The test cases can be generated automatically using formal model-checking techniques for critical portions [43]. We have also developed techniques to use heuristic-driven simulation tools to automatically generate mutation-based test cases for timeliness and other time dependent properties such as stability of a control system [44, 42].

The testing framework also allows test case executions to be more deterministic. The semantics of transaction processing real-time systems can be exploited to realize a form of prefix-based test execution [28]. This execution scheme helps concentrates testing on critical execution orders, allowing assumptions and estimations to be tested [41].

The proposed framework has been used for testing timeliness of a small real-time control system running on Linux/RTAI. The mutation-based test cases revealed significantly longer response times than both manually constructed stress tests and suites of randomly generated test cases that were approximately ten times larger [40]. We plan to evaluate the testing framework in an industrial field study and evaluate the effectiveness of the automated test execution scheme.

5. Summary and Conclusions

This paper has presented progress on six problems that arise when testing event-triggered real-time systems.

Compared to the problems presented by Schütz ten years ago, this paper emphasizes problems that are more important in event-triggered real-time systems. In addition, progress on several new problems of testing event-triggered real-time system is discussed.

The TETReS research group at University of Skövde, Sweden and George Mason University, USA, has been studying these problems in several research projects for several years. The substantial progress we have made is described in Section 4.

Collectively, the TETReS projects have yielded results in most of the problems in testing event-triggered real-time systems. The multi-pronged approach of this project has led to very positive synergy among disparate ideas. The specification constructed for formal verification (Section 4.1), can be reused for test-case generation (Section 4.3.2). The input parameter modeling and combination strategies (Section 4.3.1) can be combined with the sequences of triggering events generated for for timeliness testing (Section 4.3.2). Further, the system constraints for improved testability (Section 4.2) effect the computational complexity of formal verification and model-based test case generation.

An expected consequence of using these constraints is that the response times of tasks increase compared to classic event-triggered systems. Adding observation points means that the most urgent real-time task must wait until the next observation point before it is scheduled. The consequence of adding preemption points is that all tasks potentially have to wait for the task with the longest period between designated preemption points. Hence, when choosing the size of these intervals, the designer must consider whether the increased worst case response time can violate timeliness. However, the purpose of the constraints is that the variance of response times and the number of system behaviors decrease, while the worst case response times remain shorter than they would if using a time-triggered design.

A common goal for the TETReS project is to test event-triggered real-time systems as well as time-triggered systems. All problems described here need to be addressed. By definition, this type of research requires multiple areas of competence. The international coperation among the TETReS group has enabled us to apply multiple solutions that complement each other, resulting in a holistic approach to the development of event-triggered real-time systems.

References

[1] J. J. Alferes, R. Amador, E. Behrends, M. Berndtsson, F. Bry, G. Dawelbait, A. Doms, M. Eckert, O. Fritzen, W. May, P.-L. Pătrânjan, L. Royer, F. Schenk, and M. Schroeder. Specification of a Model, Language and Architecture for Reactivity and Evolution. deliverable I5-D4, Centro de Inteligncia Artificial - CENTRIA, Universidade Nova de Lisboa, 2005.

[2] P. Ammann, P. Black, and W. Majurski. Using model checking to generate tests from specifications. In *Proceedings of the Second IEEE International Conference on Formal Engineering Methods*, pages 46–54. IEEE Computer Society, December 1998.

[3] P. E. Ammann and A. J. Offutt. Using formal methods to derive test frames in category-partition testing. In *Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS'94),Gaithersburg MD*, pages 69–80. IEEE Computer Society Press, June 1994.

[4] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring. Deeds towards a distributed active and real-time database system. *ACM SIGMOD Record*, 25:38 – 40, 1996.

[5] A. B. Asaf. Adi, D. Botzer, O. Etzion, and Z. Sommer. Context awareness in amit. In *Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03)*, pages 160–167, 2003.

[6] P. Barrett, A. Hilborne, P. Verissimo, L. Rodrigues, P. Bond, D. Seaton, and N. Speirs. The delta-4 extra performance architecture (XPA). In *International symposium on fault-tolerant computing*, pages 481–488, June 1990.

[7] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 2nd edition, 1990.

[8] R. Birgisson, J. Mellin, and S. Andler. Bounds on test effort for event-triggered Real-Time Systems. In *The 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 212–215, December 1999.

[9] J. P. Bowen and M. Hinchey. Ten commands of formal methods. In *High-integriyt system specification and design*, pages 215–230. Springer-Verlag, 1998.

[10] R. Brownlie, J. Prowse, and M. S. Phadke. Robust testing of AT&T PMX/StarMAIL using OATS. *AT&T Technical Journal*, 71(3):41–47, May/June 1992.

[11] F. Bry and P.-L. Patranjan. Reactivity on the web: paradigms and applications of the language xchange. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1645–1649, New York, NY, USA, 2005. ACM Press.

[12] A. Buchmann, H. Branding, T. Kudrass, and J. Zimmermann. Reach: A real-time, active and heterogeneous mediator system. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, 15(1-4):44–47, 1992.

[13] K. Burroughs, A. Jain, and R. L. Erickson. Improved quality of protocol testing through techniques of experimental design. In *Proceedings of the IEEE International Conference on Communications (Supercomm/ICC'94), May 1-5, New Orleans, Louisiana, USA*, pages 745–752. IEEE, May 1994.

[14] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[15] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[16] E. W. Dijkstra. Notes on structured programming. In O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structured Programming*, pages 1–82. Academic Press, 1972.

[17] E.M.Clarke and A.Emerson. Synthesis of synchronization skeletons for branching time temporal logic. *In Logic of Programs: Workshop, Lecture Notes in Computer Science*, 131:52–71, 1981.

[18] A. Ericsson and M. Berndtsson. Detecting design errors in composite events for event triggered real-time systems using timed automata. In *First International Workshop on Event-driven Architecture, Processing and Systems (EDA-PS 06) Chicago*, pages 39–47, 2006.

[19] J. Gait. A probe effect in concurrent programs. *Software Practice and Experience*, 16(3):225–233, March 1986.

[20] G.J.Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[21] M. Grindal. *Handling Combinatorial Explosion in Software Testing*. PhD thesis, Department of Computer and Information Science, Linköping University, 2007. Dissertation No. 1073.

[22] M. Grindal, Å. G. Dahlstedt, A. J. Offutt, and J. Mellin. Using Combination Strategies for Software Testing in Practice - A proof-of-concept. Technical Report HS-IKI-TR-06-010, School of Humanities and Informatics, University of Skövde, 2006.

[23] M. Grindal, B. Lindström, A. J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Software Engineering*, 11(4):583–611, Dec. 2006.

[24] M. Grindal and A. J. Offutt. Input parameter modeling for combination strategies. In *Proceedings of the IASTED International Conference on Software Engineering (SE2007), Innsbruck, Austria, Feb. 13-15, 2007*, pages 255–260, Feb. 2007.

[25] M. Grindal, A. J. Offutt, and J. Mellin. Managing conflicts when using combination strategies to test software. In *Proceedings of 18th Australian Conference on Software Engineering (ASWEC2007), Melbourne, Australia, 10-13 Apr. 2007*, Apr. 2007.

[26] M. Grindal, J. Offutt, and S. Andler. Combination testing strategies: A survey. *Journal of Software Testing, Verification, and Reliability*, 15(3):167–199, September 2005.

[27] C. A. R. Hoare. An overview of some formal methods of program design. *IEEE Computer*, 32(1):85–91, 1987.

[28] G. Hwang, K. Tai, and T. Hunag. Reachability testing: An approach to testing concurrent software. *International Journal of Software Engineering and Knowledge Engineering*, 5(4), December 1995.

[29] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[30] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R.Zainlinger. Distributed fault-tolerant real-time systems: The MARS approach. *IEEE Micro*, 9(1):25–40, 1989.

[31] H. Kopetz and P. Verissimo. *Distributed Systems*, chapter 16, pages 411–446. Addison Wesley, 1993.

[32] 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*, pages 166–188, Milano, Italy, 1991.

[33] N. P. Kropp, P. J. Koopman, and D. P. Siewiorek. Automated robustness testing of off-the-shelf software components. In *Proceedings of FTCS'98: Fault Tolerant Computing Symposium, June 23-25, 1998 in Munich, Germany*, pages 230–239. IEEE, 1998.

[34] J. C. Laprie. *Dependability: Basic Concepts and Terminology*. Springer Verlag for IFIP WG 10.4, 1994.

[35] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, Oct. 1997.

[36] B. Lindström, M. Grindal, and A. J. Offutt. Using an existing suite of test objects: Experience from a testing experiment. *ACM SIGSOFT Software Engineering Notes, SECTION: Workshop on empirical research in software testing papers, Boston*, 29(5), Nov. 2004.

[37] B. Lindström, J. Mellin, and S. Andler. Testability of dynamic real-time systems. In *Proceedings of Eigth International Conference on Real-Time Computing Systems and Applications (RTCSA2002)*, pages 93–97, Tokyo, Japan, March 2002.

[38] J. Mellin. Supporting system-level testing of applications by active real-time database systems. In *Proceedings of the 2nd International Workshop on Active, Real-Time, and Temporal Database Systems*, volume 1553 of *Lecture Notes in Computer Science*, pages 194–211. Springer, 1998.

[39] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, 1979.

[40] R. Nilsson. *A Mutation-based Framework for Automated Testing of Timeliness*. PhD thesis, Linköping University, 2006. Dissertation No. 1030.

[41] R. Nilsson, S. Andler, and J. Mellin. Towards a Framework for Automated Testing of Transaction-Based Real-Time Systems. In *Proceedings of Eigth International Conference on Real-Time Computing Systems and Applications (RTCSA2002)*, pages 109–113, Tokyo, Japan, March 2002.

[42] R. Nilsson and D. Henriksson. Test-case generation for flexible real-time control systems. In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 723–731, Catania, Italy, September 2005. IEEE Computer Society.

[43] R. Nilsson, J. Offutt, and S. F. Andler. Mutation-based testing criteria for timeliness. In *Proceedings of the 28th Annual Computer Software and Applications Conference (COMPSAC)*, pages 306–312, Hong Kong, September 2004. IEEE Computer Society.

[44] R. Nilsson, J. Offutt, and J. Mellin. Test case generation for mutation-based testing of timeliness. In *Proceedings of the 2nd International Workshop on Model Based Testing*, pages 102–121, Vienna, Austria, March 2006.

[45] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, June 1988.

[46] Polyhedra. http://www.polyhedra.com/, 2005.

[47] J. P. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proc. 5*th *Int. Symp. on Programming*, number 137 in Lecture Notes in Computer Science, pages 195–220, Berlin, 1982. Springer–Verlag.

[48] K. Ramamritham. Real-time databases. *International Journal of Distributed and Parallel Databases*, 1(2):199–226, 1993.

[49] K. Ramamritham. The origin of TCs. In M. Berndtsson and J. Hansson, editors, *Proceedings of the First International Workshop on Active and Real-Time Database Systems (ARTDB 1995)*, pages 50–62, Skovde, Sweden, June 1995. Springer.

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

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

[52] M. Seiriö and M. Berndtsson. Design and Implementation of an ECA Rule Markup Language. In *Proceedings of the 4th International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005)*, volume 3791 of *LNCS*, pages 98–112. pringer, 2005.

[53] J. A. Stankovic. Admission control, reservation, and reflection in operating systems. IEEE Bulletin of the Technical Committe on Operating Systems and Application Environments (TCOS), Summer 1998.

[54] J. A. Stankovic, K. Ramamritham, D. Niehaus, M. Humphrey, and G. Wallace. The spring system: Integrated support for complex real-time systems. *Real-Time Systems*, 16(2-3):223–251, May 1999.

[55] TIBCO. Complex event processing, framework for operational visibility and decisions, 2007.