Bounds on Test Effort for Event-Triggered Real-Time Systems

Ragnar Birgisson  Jonas Mellin  Sten F. Andler
Department of Computer Science, University of Skövde
rbirgiss@baan.nl, {jonas.sten}@ida.his.se

Abstract
The test effort required for full test coverage is much higher in event-triggered than in time-triggered real-time systems. Thus, it is hard to attain sufficient confidence in the correctness of event-triggered real-time applications by testing. We present a general upper bound on the test effort of constrained event-triggered real-time systems, assuming multiple resources (a refinement of previous results). The emphasis is on system level testing of application timeliness, assuming that sufficient confidence in its functional correctness has been attained. Covered fault types are mainly incorrect assumptions about temporal attributes. An analysis of our approach shows that designated preemption points are required. A key factor in this approach is the ability to reduce the test effort while maintaining full test coverage.

1 Introduction
Real-time systems must respond to stimuli in a timely manner and, thus, verification of timeliness is of paramount interest. Testing is a necessary complement to other verification methods, because testing relies less on assumptions about the execution environment and the application. This work focuses on testing of timeliness, which entails the revealing of deviations from the expected behavior caused by incorrect assumptions about temporal attributes.

Full test coverage, i.e., testing of all anticipated behaviors, is strongly desirable in a safety critical environment. However, the test effort required to achieve full test coverage may be very high. Therefore, methods to bound this effort are needed in order to enable testing of timeliness with sufficient test coverage. An approach to bound the test effort for transaction-based event-triggered real-time systems is presented by Mellin [11], where application behavior is constrained so that: (M1) the number of times a transaction can be preempted is bounded; (M2) the number of currently executing transaction instances are bounded; and (M3) the points in time where the environment can be observed are constrained. In this paper, we refine the upper bound to account for multiple resource locks. Further, the effects of these constraints are analysed from a scheduling perspective.

2 Testing Real-Time Systems
According to Beizer [2], the aim of software testing is to show the presence of faults. A fault is the hypothesized cause of a deviation from the expected behavior [9].

Central concepts in testing are test effort and test coverage. Test effort refers to the effort associated with testing of a particular test object. This includes everything from designing test cases to analyzing the results. Test coverage is a metric indicating how well test cases cover anticipated real-world scenarios. For example, achieving full test coverage requires testing of the complete input space.

The chosen design paradigm has a significant impact on the required test effort. Time-triggered systems are inherently easier to test than event-triggered systems [12]. Processes in time-triggered systems work in lockstep with time and require rigid assumptions about the behavior of the environment. In contrast, event-triggered systems react to events in the environment as they occur. In a comparison made by Schütz [12], the test effort is considered proportional to the number of states that the environment can be perceived to be in during an (action granularity) interval $g_a$, because all possible combinations of events must be introduced to the system to achieve full test coverage. If, during this interval, $n$ independent events may arrive and the environment can be observed $s$ times, then the number of perceivable environment states is: $ESTAT(s,n) = \sum_{k=0}^{n} s^k = \sum_{k=0}^{n} s^k 1^{n-k} = (s+1)^n$. For time-triggered systems, the environment is only observed once during each interval and, thus, $s = 1$, whereas for event-triggered systems $s >> 1$. Further, $ESTAT$ only expresses a lower bound for event-triggered systems, because the internal state is not considered.

*This work was supported by NUTEK (The Swedish National Board for Industrial and Technical Development)

†R. Birgisson is currently affiliated with Baan in Iceland
3 Assumptions and Models

In this work, we assume the constraints (on application behavior, M1–M3 in section 1). Further, we assume that all tasks execute as transactions communicating via protected shared variables, e.g., a database object, which reduces the number of synchronization sequences. The concurrency control policy is assumed to be conservative and pessimistic, i.e., a transaction must lock all resources that it requires before executing. Our work focuses on system level testing of timeliness and, hence, it is assumed that sufficient confidence in functional correctness has been attained. Observability, which attained when the probe effect is avoided [12], is assumed to be obtained with a predictable built-in event monitor [10]. The probe effect [6] is the difference in behavior between an instrumented system and its uninstrumented counterpart. That is, to attain observability an operational system must execute as if it is being tested.

Execution Model: The execution model assumes dynamic preemptive scheduling that can handle a mixed load of soft and hard deadline transactions with periodic or sporadic arrival patterns. All hard transactions are guaranteed to meet their deadlines. In addition, we assume that the priorities of transaction instances are unique [11]. Two additional distinct assumptions are made: (E1) A currently executing soft transaction, that has been preempted the maximum number of times, is always aborted if a hard transaction arrives; and (E2) the time to abort a soft transaction is bounded.

System Level Testing Model: Figure 1 schematically depicts a test case. It describes an initial state of the system and events that should be presented during the test case. The initial state of the system represents the past, i.e., the state before test execution begins. This includes information about all active transactions, how many times each transaction has been preempted, and currently blocked transactions. Test execution is performed between begin and end in figure 1. The objective of testing for timelines is to check if the system meets its deadlines after begin. The test driver software starts the system in the initial state and presents the events from the test case to the system. The (triggered) transactions are monitored to see if any deadlines are missed. Test execution ends when all events have been presented to the system and all transactions triggered by these events have completed.

4 A General Upper Bound

The upper bound on the test effort for event-triggered real-time systems, as specified by Mellin [11], is formulated by combining perceivable environment states (ESTAT in section 2), potential blockings (BSTAT), and potential preemptions (PSTAT). This results in the total number of states, $FSTAT = ESTAT \times BSTAT \times PSTAT$. Together, $BSTAT$ and $PSTAT$ represent all possible previous states of active transactions in the system. At each observation point, the number of future states is based on all possible previous states multiplied by all perceivable environmental states. The equations are as follows:

$$BSTAT(q,t) = 1 + \sum_{k=0}^{p+1} \binom{p+1}{k} q^k t^{p+1-k} = 2^{p+1} = q \cdot t$$
$$PSTAT(p,q,t) = \left(\sum_{k=0}^{p+1} \binom{p+1}{k} t^k\right) = ((p+1)^{p+1} - 1)/p^t$$

where $p > 0$.

$BSTAT$ is based on finding all combinations of two or more transactions out of a total of $q \times t$ executing transactions, where $t$ is the number of event types and $q$ the allowed maximum number of concurrently executing transactions triggered by the same event type. Only combinations of two or more transactions are included because, at each point in time, all transaction instances are assumed to have a unique priority, and all resources are allocated to a transaction before its execution. The number of potentially preempted transactions presented in $PSTAT$ is found by considering all possible combinations of preempted transactions out of $q \times t$ transactions, when each transaction may be preempted at most $p$ times. As pointed out by Mellin [11], the upper bound represented in $FSTAT$ is conservative and pessimistic because impossible states are included.

$BSTAT$, as given here, only represents cases where a single resource is locked. A less restrictive approach is to allow multiple resources to be locked simultaneously. For this purpose, we introduce a notation for representing blockings. The notation $x \vdash y$ ($x$ blocks $y$) represents a situation where transaction $x$ blocks transaction $y$. A situation where $x$ blocks more than one transaction is represented by $x \vdash y_1, y_2, \ldots, y_m$. Transaction $x$ and all transactions blocked by $x$ participate in a blocking. A blocking scenario is a non-empty set of blockings. A blocking template is a bag (or multiset) of elements, consisting of the lengths of blockings in a blocking scenario. For example, the blocking template $[3,2]$ represents all blocking scenarios with blockings of length three and two, respectively.

Assume a system with five transactions named $a$, $b$, $c$, $d$, and $e$. Each transaction has a unique priority, such that transaction $a$ has the lowest priority and transaction $e$ has the highest. The possible blocking templates are: $[2]$, $[3]$, $[4]$, $[5]$, $[2,2]$, $[3,2]$. Each of these templates represents a number of blockings scenarios. For example, the tem-
plate \([3,2]\) represents, among others, the blocking scenario \(a \rightarrow b, c \text{ and } d \rightarrow e\). Since a blocking scenario is a set, the total number of blocking scenarios represented by this template is calculated by finding all combinations of three transactions from the set of transactions \(\{a, b, c, d, e\}\) multiplied with combinations of two transactions from the remaining two transactions, i.e., \((\frac{5}{2}) \times (\frac{3}{2}) = 10 \times 3 = 30\) scenarios. This is because, for example, \(a \rightarrow b\) and \(c \rightarrow d\) are incorrectly not considered equal to \(c \rightarrow d\) and \(a \rightarrow b\). They are equivalent because each transaction is assumed to have a unique priority.

If the elements in a blocking template are distinct (as in the template \([3,2]\)), then the total number of scenarios is calculated using binomial coefficients as shown in equation (1). The variable \(T\) represents the total number of transactions, and the variables \(c_1 \ldots c_r\) are the elements of bag \(c\) and represent the lengths of blockings in a blocking scenario such that \(\sum_{c_j \in c} c_j \leq q \times t\).

\[
\binom{T}{c_1} \times \binom{T-c_1}{c_2} \times \cdots \times \binom{T-(c_1+\cdots+c_{r-1})}{c_r} \tag{1}
\]

If the elements in a blocking template are not distinct, then (1) gives more scenarios than there actually are. For example, the template \([2,2]\) would represent \((\frac{4}{2}) \times (\frac{2}{2}) = 10 \times 3 = 30\) scenarios. This is because, for example, \(a \rightarrow b\) and \(c \rightarrow d\) are incorrectly not considered equal to \(c \rightarrow d\) and \(a \rightarrow b\). They are equivalent because each transaction is assumed to have a unique priority. If the transactions would not have unique priority, then the order in which transactions are introduced to the system is important. In this case, the two blocking scenarios should be considered different, because they would represent different orderings.

In this particular example, each blocking scenario consists of two blockings of the same length (2). Therefore, we need to divide the number of scenarios, given by equation (1), by the number of permutations of two elements (representing the two blockings), which gives \((\frac{4}{2}) \times (\frac{2}{2}) / 2! = 15\). Extending (1) to account for redundant permutations results in:

\[
\frac{\binom{T}{c_1} \times \binom{T-c_1}{c_2} \times \cdots \times \binom{T-(c_1+\cdots+c_{r-1})}{c_r}}{\prod_{n \in \text{elem}(c)} \text{card}(n, c)!} \tag{2}
\]

Here, we have defined a function \(\text{card}(n, c)\) to be the number of occurrences of \(n\) in \(c\), and a function \(\text{elem}(c)\) to be the set of distinct elements in \(c\).

Equation (2) describes the number of scenarios represented by a single blocking template. For a given number of transactions and resources, there are several possible blocking templates. Assume that set \(C\) contains all possible templates. An algorithm to generate \(C\) is found in [3]. The blocking templates represented by each blocking template are distinct, because no two templates are identical. Therefore, the different scenarios may be summed up as shown in (3).

\[
\sum_{c \in C} \frac{\binom{T}{c_1} \times \binom{T-c_1}{c_2} \times \cdots \times \binom{T-(c_1+\cdots+c_{r-1})}{c_r}}{\prod_{n \in \text{elem}(c)} \text{card}(n, c)!} \tag{3}
\]

Substituting \(q \times t\) for \(T\) in (3) gives \(BST\ AT^t\):

\[
BST\ AT^t(q, t, C) = \sum_{c \in C} \frac{\prod_{j=1}^{r} (q_{t_j} - \sum_{e_j} c_e)}{\prod_{n \in \text{elem}(c)} \text{card}(n, c)!} \tag{4}
\]

5 Effects of Applying the Constraints

This section discusses impact that the constraints (M1–M3) have on scheduling.

5.1 Maximum Number of Preemptions

As we assume that soft transactions can be aborted within a bounded time (E2), the major problem is when at least two hard transactions \((T_1, T_2)\) execute in the system and one of them \((T_1)\) has reached its preemption maximum. If transaction \(T_2\) has a higher priority, then we face a non-preemptive scheduling problem. To solve this problem we propose the use of designated preemption points. A preemption point marks a point in the execution of a transaction where it may be preempted. Each transaction is thereby split into one or more non-preemptive intervals bounded by preemption points. By bounding the worst-case execution time of such non-preemptive intervals we can predict the worst-case delay due to these intervals. Further, the constraint on maximum number of preemptions can be easily enforced by ensuring that the number of times a transaction may reach a preemption point does not exceed the number of allowed preemptions. One way to do this is to guarantee a minimum execution time between preemption points.

5.2 Maximum Concurrent Transaction Instances

To handle the bound on concurrently executing transactions, there must be an offline guarantee that the number of executing hard transactions never exceed the maximum. An admission controller and an overload resolver, for example as presented by Hansson et al. [8], is required to enforce this maximum. These two mechanisms can be used to control the amount of transactions in the system.

5.3 Observation Points

The result of constraining observation of the environment is that asynchronous interrupts are delayed. Periodic transactions do not have to be delayed, because the observation period can be set properly with respect to all periodic transactions. However, for sporadic transactions, the observation period must be accounted for in the release time, because an event may occur between observation points.
6 Applicability

The constraints discussed in this paper bound the test effort for event-triggered real-time systems. By using our approach, it is possible to estimate the test effort at an early stage. This provides an opportunity to refine the design of an application in order to reduce the required test effort. Further, better planning of the testing process is possible, since the test effort is bounded. Automating test case generation, selection, and execution in a system that uses our constraints enables full test coverage.

7 Related Work

In contrast to the earlier work [11], our work allows multiple resources. Moreover, the effects that the constraints for reducing the required test effort have on scheduling have been studied in this article.

In contrast to the work done by Schütz [12], who emphasizes time-triggered systems, this work addresses how the required test effort of event-triggered real-time systems can be reduced in order to bring it closer to that of a time-triggered system. In particular, an upper bound is described whereas Schütz’ work only describes a lower bound for event-triggered systems.

Test effort can be reduced by omitting certain test cases. All such methods are based on defining the fault coverage, and deriving test cases from a specification describing the test object. These range from general methods such as category partitioning [1,7], specifying systems by using, e.g., finite state machines [5], to statistical methods [4] that are used to randomly select inputs from an input space. The underlying assumption in these methods is that the selected test cases are representative. In contrast, our approach defines an upper bound for full coverage when testing timeliness of an event-triggered real-time system. Our approach assumes a static selection of test cases based on well-defined previous states of a system and future events.

8 Future Work

A further improvement, analysis and validation of the behavior of our upper bound is needed. For example, the tightness of the upper bound has not been investigated. Test case generation and selection for our method would be required to automate testing. In particular, a method for finding the most failure-prone scenarios is of paramount interest. The latter could also be combined with a specification method.

9 Conclusions

We have formulated a new expression for potential blockings, which allows multiple resources to be locked simultaneously. This has resulted in a refined formula that gives an improved upper bound on the test effort for event-triggered real-time systems.

We conclude that Mellin’s constraints require the use of designated preemption points to avoid unpredictable durations of non-preemptable periods, the ability to abort soft transactions in bounded time, and incorporation of observation interval length into scheduling which employs admission control as well as overload resolution.

The test effort for constrained systems can be estimated at an early stage which enables better planning of the test process. Further, since the test effort is bounded, it is possible to certify the level of test coverage that has been reached.

Acknowledgements: We would like to thank our colleagues for their valuable comments.

References