2010 Sixth International Conference on Networking and Services

Testing Environment for Real-time Communications Intensive Systems

Diana Alina Serbanescu
Technical University of Berlin
Faculty of Electrotechnics and Informatics
Berlin, Germany
Email: diana.serbanescu@fokus.fraunhofer.de

Ina Schieferdecker
FOKUS Fraunhofer
Institute for Open Communication Systems
Berlin, Germany
Email: ina.schieferdecker@fokus.frauhofer.de

***Abstract*—In the engineering world, more and more accent is put on the real-time embedded applications. They are used almost everywhere, in domains as: automotive, robotics, health, avionics, spacecraft, industrial control, etc.. Often, on the proper functionality of real-time systems, depends the well functioning of a whole system and their mistakes might have fatal consequences. Often such real-time applications have additional requirements for a fast and reliable communication with the rest of the system they are part from. In order to prohibit misbehavior of such a critical application, proper tests should be developed and applied to it. Although real-time application development gained so much ground and a lot of tools and paradigms for real-time emerged, there are still little effort invested in the development of tools for software real-time testing. In this paper we propose a testing framework solution for real-time, based on a standardized testing language, that was used with success by now in many domains, as telecommunications, automotive, health, etc.. This language is named TTCN-3 – the acronym comes from Testing and Test Control Notation – and it was developed by European Telecommunications Standards Institute (ETSI). The purpose of this paper is to demonstrate that the proposed test framework achieves both a time deterministic behavior and high performance at the communication points with the System Under Test(SUT).**

***Keywords*-testing, real-time, embedded, TTCN-3, communications intensive**

I. Introduction

Real-time embedded systems play an important role in nowadays world. Many industrial processes relay on the good functioning of embedded systems. The area of applicability for real-time systems is very wide, ranging from industrial process controllers, to technical equipment used in the health sector, automotive, avionics and space control. As each sector develops technologically, more and more equipments and functionalities are added, and the complexity of interactions between different components increases. There are networks of interconnected embedded systems, communicating with each other through timed protocols, continuously exchanging data and synchronizing with each other. Such networks can represent the future car to car communicating networks or networks on wheels, monitoring and controlling systems, health assistance systems or disaster recovery systems. As we can imagine, the overall well functioning of the system depends on the well functioning of every each composing element. A comprehensive and systematic quality assurance of such complex system represents a real challenge. This challenge has to be assumed, since most of the real-time systems perform safety or other critical procedures, which might become fatal if not performed correctly. The quality assurance of software-intensive systems is still lacking full automatic techniques and is performed by a multiplicity of proprietary test systems and test platforms. Since the general tendency is to integrate systems that communicate with each other and have also strict timing constraints, a good approach for testing would be to provide a standard framework for asserting good functionality of all those systems. In real-time communications intensive applications, the simultaneous achievement of the required performance and determinism is a difficult problem. Also the communication aspect indicates that the focus should be on the I/O communication ports and timely features of these.

Our approach is to take an already standardized language, that is practiced in the industry for some years, and to design a test framework that is also suitable for real-time. The aim is to provide a manufacturer-independent test environment consisting of modular service components (e.g. test management component, sender component, receiver component, platform adapters, etc.). TTCN-3 [4], [6], [5] provides this type of modularity, it is a well structured language developed specially for testing, it is standardized and was used with success in domains as telecommunication and automotive. TTCN-3 has powerful concepts and efficient mechanisms for dealing with ports and for emphasizing the communication aspects between the Test System(TS) and the System Under Test (SUT). The drawbacks of TTCN-3, as it is right now, are that it was designed for testing only functionality-related requirements of the systems, and not real-time aspects. As it is well known, for a real-time system the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed.TTCN-3 lacks

978-0-7695-3969-0/10 $26.00 © 2010 IEEE
DOI 10.1109/ICNS.2010.58

of proper mechanism for dealing with real-time applications, both at conceptual and implementation level. It has no proper mechanism for dealing with time and for imposing time limits. A comprehensive discussion on the missing issues of TTCN-3 regarding real-time handling is done in [14], [3]. On the conceptual level, research was performed to eliminate those shortcomings and several extensions to the language were already proposed [8], [9], [7], [11], [10], [2], [3]. In this paper we discuss how an appropriate implementation for the proposed extensions would look like and present an abstract implementation design. The implementation design makes the connection between the abstract real-time TTCN-3 specification and possible different real-time operating system platforms. It introduces an intermediary step of abstraction, making a bridge between the abstract and test-oriented concepts and their translation into concepts from the operating systems world. The latter are immediately implementable on any real-time operating system platform, using the specific API for that platform and the design guide that is provided in this paper. In second section we refer the source for the extensions that were chosen for the framework. In the third section we present the implementation design and in the fourth section we present a real hardware-software platform that was chosen as basis for the implementation. We also present some parameters indicating the performance of the platform and from which general performance estimations can be deduced. Nevertheless, the entire platform evaluation can be made only when the whole testing system will be implemented and used for testing real real-time applications. The entire work - both the concepts and the design - is concentrating on the parts from the TS that are evolving around communication ports with the SUT, around sending and receiving of messages. It is important to perform those operations efficient and to have accurate time estimations of them. The approach we present is inspired by the theory of the real-time operating systems [15], [16], [17], [18]. The Real-time Test System (RTTS) is seen as a complex environment, composed from different elements with real-time deadlines, elements which have to be managed and scheduled. In this context, we can say that the test cases written using the abstract TTCN-3 specification are to be translated into a real-time application, running on a real-time operating system (RTOS).

II. General Discussion About TTCN-3 And Its Extensions

As a basis for the conceptual level, we are going to follow the approach taken by the TEMEA project [8], approach that is also presented in [3]. They provide conceptual means for dealing with time, and for relating time to incoming and outgoing events at the communication ports between the TS and the SUT. The majority of concepts are taken over mostly as they are defined in TEMEA project, but there are some additional observations and restrictions to those concepts that are discussed in the following.

As a correction to the [3], it should be mentioned that for the sending of messages only the temporal predicate `at` makes sense. It imposes a precise time at which TS should take some action. The other time predicates are not good in this context, because they imply a random decision to be taken by the TS (e.g. send a stimuli at a random point in time within the specified interval). This would only increase the complexity of the TS, possibly leading to non-deterministic situations (makes the tests non-repeatable).

Nevertheless, they are safe to be used with the receive operation, because only an evaluation operation based on the timestamp of the event would be performed here. Logical operators as `and` and `or` can be used here in conjunction with temporal predicates in order to form complex expressions.

As in the case of `send` operation, in conjunction with `break` only the usage of the temporal predicate `at` makes sense (correction to the [3]).

```
alt{...}break at datetime;
```

III. Implementation Design For The Real-time Extensions

A. Dealing with Time

At the TTCN-3 specification level, time points are regarded as positive real numbers. Nevertheless, on a real machine they will be translated in terms of number of ticks of the inner clock of the system. The values and expressions attached to the temporal predicates from the real-time TTCN-3 specification are calculated using the internal representation of time.

From a time perspective, the `send`, `receive` and `break` operations in combination with the time predicates and time measurement mechanism can be divided in two categories:

- Control instructions:

```
p.send(msg) at tx;
alt{...}break at ty{...}
```

The control instructions are using internal mechanisms, as the clock of the system and the time interrupts, for imposing constraints to parts of the execution of the TS. The control instructions are the instructions that have to be performed at precise points in time. When those points in time are encountered, the clock system generates a timer interrupt. The timer interrupt is treated by an interrupt handler, which interrupts any running activity and performs the code associated with that instruction.

Such instructions are `send` and `break` instructions with `at` time predicate. At the initialization of the test system, all the time values given as parameters to the `at` predicate are saved into a list maintained by the kernel (see Figure 1). This list is ordered in the increasing order of time values. Another step performed in the initialization phase is that of creating

time handlers for each value in the list. The handlers have a behavior associated with the type of instruction to which the temporal predicate is applied: `send` or `break`. When the clock value equals one value from the list, a time interrupt is generated and the associated event handler is invoked. Because the interrupts are not generated periodically, the clock should run in one shot mode.

- Verification and log instructions:

```
p.send(msg)→timestamp;
p.receive(msg)→timestamp;
p.receive(msg) at tx/
               before tx/
               after tx/
               within(tx,ty);
```

The log instructions save the time when the message is received into the timestamp variable. The value is transomed from the internal representation into a `datetime` value, and handed to the upper abstraction layer of the TTCN-3. The verification instructions evaluate if a message was received in the expected time frame and the evaluation is based on the timestamp associated with the message.

These mechanisms are presented in the following sections, when each instruction is analyzed in detail.

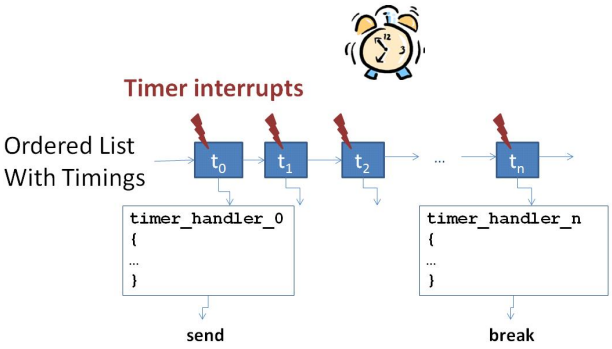

Figure 1. Generating timer event and triggering the event handling routine for a ”send” or a ”break” statement

B. Dealing with Events

We are dealing with two types of events: internal generated events, for time determinism, also called time events, and external generated events, which are triggered by the I/O ports whenever a new message has arrived.

1) Time Events: For the time events, that are generated as described in Subsection III-A, we have two types of event handlers. The handler associated with a `send` operation performs sending of a message when it is invoked. The other one executes the code associated with the `break` instruction in TTCN-3. In TTCN-3 the `break` operation is always used in relation to the `alt` statement. The `break` instruction is used to impose an upper time limit to the execution of `alt` statement (see Figure 2). Therefore, the invoked handler is also used to kill the task associated with the `alt` statement. The handler associated with the `send` operation is triggered in a similar way with the one for `break`.

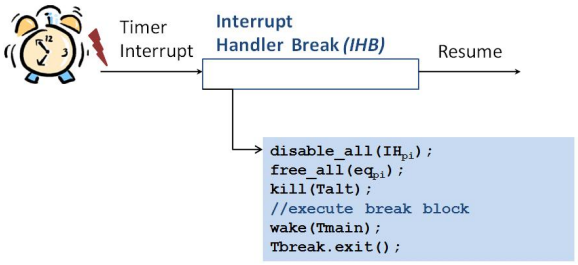

Figure 2. Time event handling routine associated with ”break”

2) External Events: When a message is received from the SUT, an I/O interrupt is generated on the respective port. At the TS initialization we create for each incoming port a virtual queue. The interrupt activates the handler. The handler takes the message, takes the clock time, and saves the message together with the timestamp into the queue associated to the port. The procedure is presented in Figure 3.

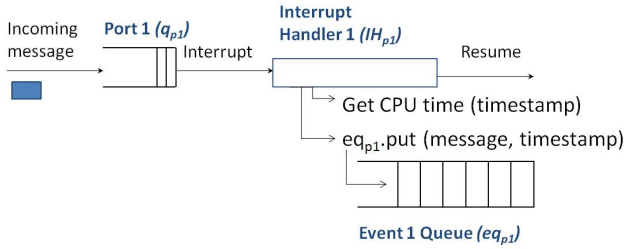

Figure 3. Event handling routine triggered by an external event

C. Verification and Log Instructions

Verification and log instruction do not use interrupts to influence the behavior of the TS. They do not influence the scheduling mechanism, they do not cause the preemption of the current running thread. Their effect is just to keep a timestamp of the time when the associated operation was performed. In the case of the `receive` operation, the timestamp is taken directly from the value saved by the interrupt handler associated with the incoming event.

In the case of `send` operation, there are two possibilities. Send with temporal predicates are implemented using interrupt handlers(see Figure 4). Send without temporal predicate executes just a normal send operation in the context of the thread that calls it. For saving the time at which the message is send, a time primitive reads the time immediately after. The thread must not be preempted between those operations. therefore, the interrupts have to be disabled, and maximum

priority level for the task should be assured during this region.

D. Component Task with Timed Send

We regard test components from TTCN-3 specification as tasks at the RTOS level. It is interesting to analyze how a test component task is influenced by certain operations contained within the component, and which represents the component's behavior. The description of the component's behavior in TTCN-3 is described in Listing 1.

```
testcase SendAtTime() runs
on SenderComponent
{
    datetime tx;

    // other computations
        senderPort.send(msg) at tx;;
    // other computations

}
```

Listing 1. Component task with timed send. TTCN-3 code.

In Figure 4 presents one possible execution flow, if the "send task" is the main prioritized task running in the system at that time. The `send` instruction is a control instruction and therefore, it has impact on the state of the "send task" that is running in the system at the moment at which the `send` operation should be executed. When the time interrupt is generated, it triggers the handling routine, which preempts the "send task" and puts it in the ready queue. Ready queue is a queue maintained by the system, and which contains the tasks that are ready for being scheduled next. After the routine finish the sending operation, the scheduler gets the first available task from the ready queue and dispatches it. Usually, the ready queue is an ordered queue, sorted on different criteria as task priority or task deadline.

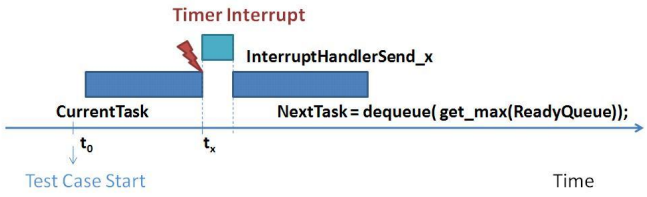

Figure 4. Execution time line for executing a "send" at a given time

E. Component Task with Send and Timestamp

In the execution time line presented in Figure 5 it can be observed that the `send` with `timestamp` operation does not preempt the execution of the component itself. Nevertheless, after sending, the task should not be preempted by another task before keeping the timestamp. Basically, it locks the processor for a short period.

TTCN-3 code associated with this behavior is in Listing 2.

```
testcase SendTimestamp() runs on SenderComponent
{
        datetime tx;

    // other computations
            senderPort.send(msg) ->
        timestamp tx;
    // other computations

}
```

Listing 2. Component task with send and timestamp. TTCN-3 code.

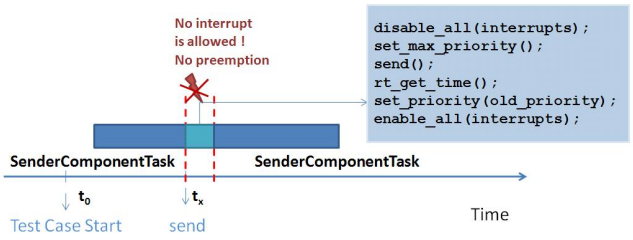

Figure 5. Execution time line for a "send" with recording the timestamp

F. Alt Operation with Receive Branches and Break Condition

A more complex interaction is encapsulated into the `alt` statement. One example of one `alt` waiting on two ports is described in Listing 3. All the elements described before are used in this scenario. The proposed solution, with interrupts for events, with handlers, and tasks represents a real-time alternative to the snapshot semantic proposed in the TTCN-3 standard. Our intention is to split the block of the snapshot into real-time tasks that can be efficiently scheduled at the level of the RTOS.

In our approach, the `alt` is associated with a task that manages two queues. The queues are created also by the parent component, altogether with the creation of the "alt handler". When a message is received on one of the ports, an interrupt is generated and a handler is called. The handler takes the timestamp and save it together with the message in one of the abstract queues. There is a convention by which each queue port has also an abstract queue associated with it. The handler of the queue is awaken and starts processing. As a manager of the queues, the "alt task" takes the last coming message and compares it against time and structural patterns. Function Filter presented in Figure 8 encapsulates the generic algorithm performed for matching. If the message matches, then the behavior associated with that branch is executed next in the context of the current task. If it doesn't match, then the "alt task" blocks waiting for the arrival of another message. The "alt task" can be interrupted while executing by a timer handler or by an I/O interrupt handler, preempted and moved to ready queue. It can be preempted also by tasks with a higher priority. When

the processor becomes available and there is no other task with a higher priority running in the system, then it gets the processor and continue its computation (see Figure 7). State transitions for the ”alt task” are presented in Figure 6.

```
testcase InTimeReceive() runs on ReceiverComponent
{
    datetime tx;
    datetime ty;
    datetime tz; datetime tt;

        alt {
        [] pa.receive(tmpla1) within(tx, ty) {
                        setverdict (pass);
        }
        [] pb.receive(tmplb1) before tz {
                setverdict (pass);
                }
        [] pa.receive  {
           // any other message,
           // any other time
                setverdict(fail);
                }
        [] pb.receive  {
                    // any other message,
           // any other time
                setverdict(fail);
                }
        } break at (tz  + 10*millisec){
                        setverdict(error);
            log( );

        }

}
```

Listing 3. TTCN-3 code sample for an alt with two ports.

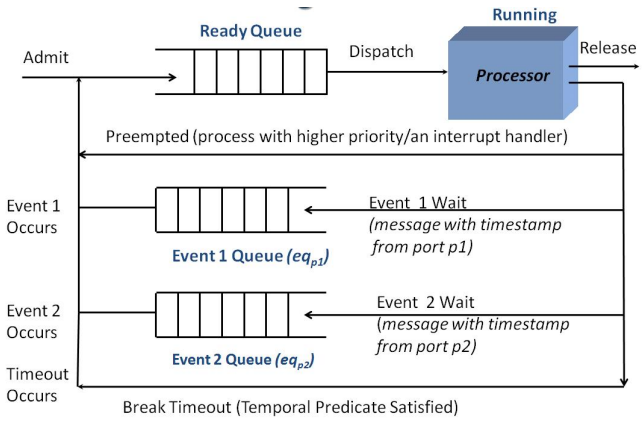

Figure 6. Alt task transition states

In Figures 9 and 10, two possible execution flows are presented. In the first one, two messages arrive, generating interrupts. The ”alt” tasks preempted twice, one of the message correspond with one pattern and is received in time, therefore the execution is continued with the behavior of the associated branch. In the second case no message is received in time, a timer interrupt is generated and the ”alt task” is killed. An error behavior is executed in this case.

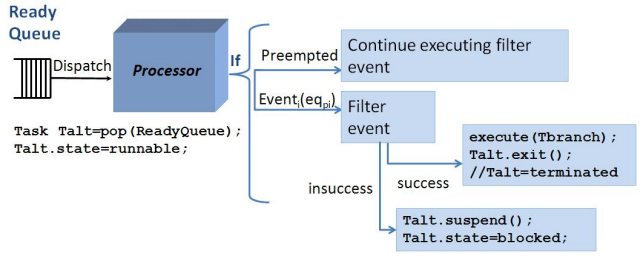

Figure 7. Behavior of ”alt” task at runtime

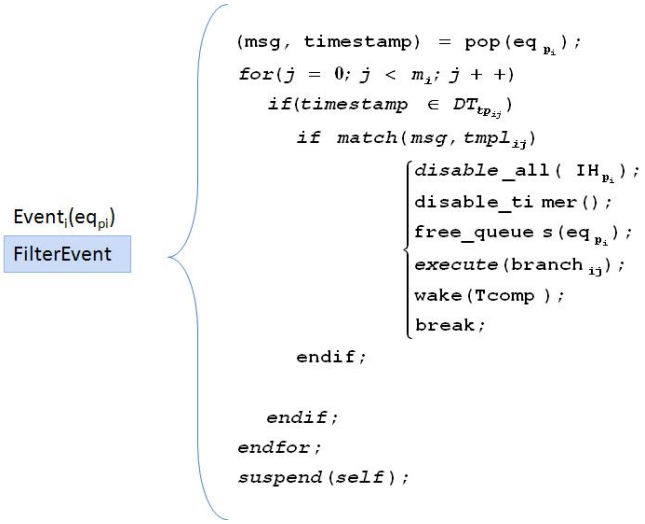

Figure 8. Filter function after receiving the event

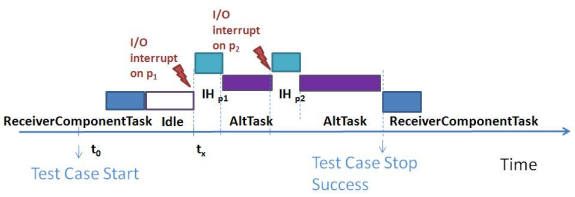

Figure 9. Possible flow of events

It is interesting to imagine how the system would look like if we have more than one component. An algorithm for assigning priorities to different tasks associated with different components will be a good choice. There are also situations depending on the misbehavior of the SUT that can induce a malfunctioning of the test system itself. Such a situation is a flooding of the TS with input events. Handling endless coming events at the highest priority, may lead to deadline overrun on the test system side. Those situations should be envisioned and taken into consideration at the design phase.

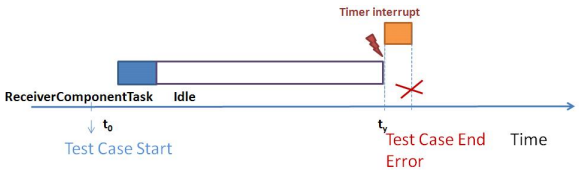

Figure 10. Possible flow of events

IV. Results

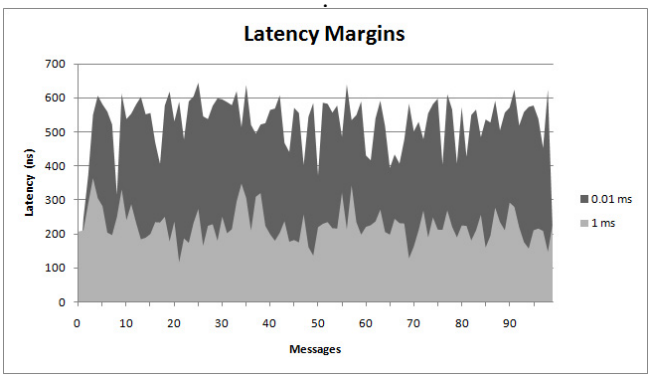

Figure 11. Latency Margins for the ”send” operation with strict timing

For the actual implementation of the proposed model we chose a Linux platform (Ubuntu 8.10) on which we recompiled the vanilla kernel 2.6.24, which was previously patched with RTAI [19] version 3.6. The systems are installed on a machine with a GenuineIntel(R) Duo Core CPU T2500 @ 2.00 GHz. The maximum latency values, when the system is loaded, are under 20 microseconds; those values are obtained by running the calibration tests that come together with the patch for RTAI. Using the RTAI we are going to implement the concepts presented above and study the latencies for sending operations at certain time points. The RTAI use as a hard real-time timer, a timer of type APIC which has a frequency of 10361250(Hz). We implemented the `send` operation with timing constraints, as it is described in the previous section. We performed several sending of messages with different frequencies, at established timed points. We evaluated the skew time between the imposed time for sending and the actual time when the operation was actually performed. The skew might be positive or negative. It represent the latency of the system, caused by scheduling and context shifting of tasks. The results we obtained indicate that de latency is limited within certain bounds. The bounds are also small, so the performance of the system is quite good. Figure 11 presents the results for the tests. Streams of messages where sent periodically, with variation of periods rank of 0.01 ms, 1 ms, and some values in between. The maximum latency for the stream with the period of 0.01 ms is of 638 ns and the maximum latency for the stream with the period of 1 ms is of 365 ns. As it can be observed from the graphic, the latency decreases as the period grows. When the period is very small, it becomes comparable with the time required for context switching. The latencies cumulates and might overlap the period, increasing that way, the latencies for the next sending operations. Therefore, for achieving a real-time behavior, it is important to establish first, the parameters within which it can be obtained. One such parameter is the time granularity in which the systems has the required responsiveness. We variate the period of sending the messages to see how system behaves for each stream. Each stream is characterized by a certain period inter messages. We decrease the period, from 1 ms to 0.01 ms, to consequently increase the stress on the system. We chose this range of values for the periods, because most of the intensive real-time applications have timing constraints within this time range. We are also limited by the frequency of our system's clock, and therefore, the frequency of the operations we programm, should be set according to it.

In our approach, we considered that each test component has access to the clock of the CPU on which it runs, and it relates its timing behavior to it.

V. Conclusion

In this paper we envisioned a test framework based on the extended version of TTCN-3. First, the extensions added to the language are summarized. Then, the relation between those concepts and existing TTCN-3 statements is studied, an implementation is proposed, and an interaction at runtime between those elements is envisioned. The framework is seen as a whole, from the conceptual level to the implementation paradigm and is intended to become a standardized platform for testing real-time applications.

Further work should address a distributed testing platform, where different components can be assessed to different CPUs. Different timing properties of the component (e.g. starting time, ending time, deadline) should be studied and developed. Also, a schema based on component's property, and a scheduling mechanism based on the prioritization scheme can be conceived.

Acknowledgment

The authors would like to thank to Petre Razvan for the support and review.

References

[1] International Organization for Standardization, *ISO 8601:1988. Data elements and interchange format — Information interchange — Representation of dates and times*. Geneva, Switzerland: International Organization for Standardization, 1988, see also 1-page correction, ISO 8601:1988/Cor 1:1991. [Online]. Available: http://www.iso.ch/cate/d26780.html

[2] D. A. Serbanescu, V. Molovata, G. Din, I. Schieferdecker, and I. Radusch, "Real-time testing with TTCN-3," in *TestCom/-FATES*, 2008, pp. 283–301.

[3] J. Grossmann, D. Serbanescu, and I. Schieferdecker, "Testing embedded real time systems with ttcn-3," Los Alamitos, CA, USA, pp. 81–90, Dec. 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICST.2009.37

[4] ETSI: ES 201 873-1 V3.2.1, "Methods for Testing and Specification (MTS). The Testing and Test Control Notation Version 3, Part 1: TTCN-3 Core Language," Sophia Antipolis, France, Febr. 2007.

[5] ETSI: ES 201 873-5 V3.2.1, "Methods for Testing and Specification (MTS). The Testing and Test Control Notation Version 3, Part 5: TTCN-3 Runtime Interfaces," Sophia Antipolis, France, Febr. 2007.

[6] ETSI: ES 201 873-4 V3.2.1, "Methods for Testing and Specification (MTS). The Testing and Test Control Notation Version 3, Part 4: TTCN-3 Operational Semantics," Sophia Antipolis, France, Febr. 2007.

[7] J. Großmann, I. Schieferdecker, and H.-W. Wiesbrock, "Modeling property-based stream templates with ttcn-3," in *TestCom/FATES*, 2008, pp. 70–85.

[8] TEMEA Project, "web site of TEMEA (TEst specification and test Methodology for Embedded systems in Automobiles)," Dec. 2009. [Online]. Available: http://www.temea.org

[9] Z. Dai, J. Grabowski, and H. Neukirchen, "Timed TTCN-3 – a real-time extension for TTCN-3," Dec. 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.953

[10] H. Neukirchen, "Languages, Tools and Patterns for the Specification of Distributed Real-Time Tests," Ph.D. dissertation, Georg-August-Universitt Gttingen, 2004.

[11] I. Schieferdecker and J. Gromann, "Testing of Embedded Control Systems with Continous Signals," in *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme II*. TU Braunschweig, 2006, pp. 113–122.

[12] I. Schieferdecker, E. Bringmann, and J. Grossmann, "Continuous ttcn-3: Testing of embedded control systems," in *SEAS '06: Proceedings of the 2006 international workshop on Software engineering for automotive systems*. New York, NY, USA: ACM Press, 2006, pp. 29–36.

[13] G. J. Myers, *The Art of Testing*, 1st ed. John Wiley & Sons, 1979.

[14] R. Sinnott, "Towards more accurate real-time testing," in *The 12th International Conference on Information Systems Analysis and Synthesis (ISAS 2006)*, Orlando, Florida, 2006.

[15] L. E. Leyva-del Foyo, P. Mejia-Alvarez, and D. de Niz, "Predictable interrupt scheduling with low overhead for real-time kernels," in *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 385–394.

[16] F. M. Proctor and W. P. Shackleford, "Real-time operating system timing jitter and its impact on motor control," in *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. National Institute of standards and Technology, 2001.

[17] K. Ramamritham and J. A. Stankovic, "Scheduling algorithms and operating systems support for real-time systems," in *Proceedings of the IEEE*, 1994, pp. 55–67.

[18] A. Mohammadi and S. G. Akl, "Scheduling algorithms for real-time systems," School of Computing Queen's University, Tech. Rep., 2005.

[19] R. M. Team, "Web pages of RTAI - Real Time Application Interface for Linux," Dec. 2009. [Online]. Available: https://www.rtai.org/