Schedulability Analysis of an Event-Based Real-Time Protocol Framework*

João Rodrigues
INETI
joao.carlos@mail.ineti.pt

João Ventura
Skysoft Portugal
jcv@skysoft.pt

Luís Rodrigues
Universidade de Lisboa
ler@di.fc.ul.pt

Abstract

This paper presents a method to analyze the timing behavior of an event-based real-time protocol composition framework. The framework, called *RT-Appia*, allows the development and implementation of configurable real-time protocol stacks. The method performs the schedulability analysis for events flowing in a real-time channel and is based on the holistic theory for distributed real-time systems. To illustrate the use of the model, a stack of modular reliable group communication protocols for the CAN field-bus is analyzed and the collected results are compared with previous work.

1 Introduction

A fundamental goal in the design of a real-time system is that it must behave in a predictable manner with respect to timing requirements. This means that it must be possible to show, demonstrate or prove that the timing requirements are met for the lifetime of the system [6].

With the increase of processing power and network bandwidth it is possible to build sophisticated distributed hard real-time systems. The construction of communication systems using the composition of several micro-protocol objects, encourages the re-use of protocol components and allows applications to configure protocol stacks exactly tailored to their needs. This aspect is particularly relevant in the context of real-time applications where, due to memory and power consumption constraints, it is interesting to execute in each component just the protocol layers required to support the intended functionality.

A protocol kernel is a software package that supports the composition and execution of micro-protocols. In terms of protocol design, the protocol kernel provides the models and the tools that

*Parts of this report will be published in the Proceedings of the Seventh IEEE International Workshop on Object-oriented Real-Time Dependable Systems, San Diego, CA, USA, 7-9 January 2002. This work has been partially supported by the project POCTI/33127/99, MOOSCO.

allow the application designer to compose stacks of protocols according to the application needs. In run-time, the protocol kernel supports the exchange of data and control information between layers and provides a number of auxiliary services (such as memory management for message buffers and timer management). A real-time extension of a protocol kernel should be able to balance the flexibility and efficiency of micro-protocols with the predictability requirements of real-time applications. To demonstrate that timing requirements are met, a hard real-time protocol kernel should provide the set of tools necessary to perform schedulability analysis.

This paper presents a method to perform the schedulability analysis for compositions of micro-protocols in a protocol kernel, including message transmission. As usual in this type of methods, we do not aim to perform an exact analysis, just to obtain an upper bound of the response time, which can be used to demonstrate and prove the timing guarantees of a given application.

We have implemented a software tool able to execute the proposed analysis. This tool is a companion design tool for the *RT-Appia* protocol kernel [4]. It should be noted that the model (and the tool) can be adapted to other protocol kernels such as CORDS [9] or RT-Cactus [1, 2].

The paper is organized as follows. Section 2 presents the motivation for the work and a small survey of the related work. Section 3 contains an overview of the *RT-Appia* system. In Section 4, the theory underlying the selected timing analysis method is summarized. Section 5 shows preliminary schedulability analysis results of a case-study protocol stack; the stack was developed to provide fault-tolerance communication over the CAN field-bus. Finally, Section 6 concludes the paper.

2 Motivation and Related Work

We are building *RT-Appia* [4], a framework for the development and implementation of configurable real-time protocol stacks. The goal of *RT-Appia* is to allow the construction of specialized protocol stacks for real-time communication, through the composition of micro-protocols. *RT-Appia* is an event-based kernel where individual micro-protocols are implemented as protocol objects that subscribe and produce events; interactions among adjacent protocols are modeled by the exchange of these events.

It is our goal to include in the framework mechanisms to extract the timing behavior of protocols compositions. The first model and tool that has been developed for this purpose [10] was based on the timing offset model proposed in [8].

The current work intends to explore alternative methods to extract the timing behavior of event-based protocols compositions in the *RT-Appia* protocol kernel. In particular we intend to exploit the holistic schedulability analysis [7] since it offers greater flexibility than the timing offset model when applied to distributed systems. Additionally, we feel that it is interesting to make available tools based on different models, compare them and let the system designer choose the one that better suits the requirements of his work.

Therefore, in this work we explore holistic schedulability analysis for distributed hard real-time systems presented in [7]. As a case-study, we use the set of modular fault-tolerant group communication protocols designed for the CAN network (EDCAN and RELCAN) described in [5]. This same set of protocols has been already analyzed using the timing offset model in [10]. The goal is to demonstrate the flexibility of the holistic analysis and to compare the results of both techniques when extracting the timing behavior of protocol compositions in *RT-Appia*.

3 RT-Appia Overview

RT-Appia,[4] is a real-time extension to the *Appia* protocol composition framework [3]. In *RT-Appia* each stack is composed of one or more real-time channels. Each real-time channel is an ordered sequence of sessions, instances of a specific protocol layer. The session maintains state that is used by the layer to process events. The sequence of layers associated with a given channel defines the quality of service implemented by the channel.

Communication between layers is made by exchange of events. For each subscribed event, the protocol layer must provide a handler to process that event. During the quality of service definition, each layer declares the set of events the layer produces and that the layer is interested in subscribing. Using this information, *RT-Appia* constructs event routes that maintain the exact set of sessions that need to be visited by each event, optimizing the time needed for the event to traverse the channel.

Within each real-time channel, the routing of events is implemented by an EventScheduler, an active component responsible for selecting the next event to be processed. The programmer may also associate a different priority to each event. It should be noted however that all events of the same channel are processed by a single RealtimeThread. This strategy tries to minimize the overhead associated with concurrency control and context switching. The scheduler maintains all events that flow in the channel in a data structure that we call the schedulable event set. The internal structure of the set takes into consideration the priorities of the events but also the requirement to preserve the FIFO order of events of the same priority exchanged between any pair of layers. When invoked, the event scheduler selects the schedulable event of highest priority and delivers it to the session that it should visit in its route. The processing of the event by a session is performed by the handle method of that session. During the processing of the handle method, the session may insert new events, possibly of higher priority, in the set of schedulable events. However, the invocation of the method handle is not preempted by the insertion of other events in the set. In particular, if the session decides to propagate the current event in the channel it must re-insert the event being processed to the schedulable set. Only when handle returns, the scheduler picks a new (or the same) event to be consumed. Naturally, the processing of events of a channel can always be preempted by an event of a higher priority channel.

A communication channel interfaces the network and the application at well defined layers. Typically, every channel has an application layer and a network layer. These layers are responsible for inserting events in the channel in response to user stimuli or messages arriving from the network.

To perform schedulability analysis of the *RT-Appia* objects, it is necessary to derive the Worst Case Execution Time (WCET) of any given channel activation. As previously noted, *RT-Appia* stores an event route for each event. The event route defines the set of layers visited by a given event. From the event route and the WCET of the handle methods it is possible to derive the worst case computation time of any given event. However, the processing of an event at one session may generate other events, so it is also necessary to capture the *related chain* of events with the longest computation time. A *related chain* of events terminates when the last event is delivered to the application, network or a timer.

The execution time for a chain of n events can be computed with the following equation:

$$C_{chain} = \sum_{n} C_{H_n} + n.C_{Sch} \tag{1}$$

C_{H_n} is the execution time of the handle method of session n and C_{Sch} the time the event scheduler spends inserting (C_{SchIn}) and removing (C_{SchRem}) an event in the schedulable event set and is computed by:

$$C_{Sch} = C_{SchIn} + C_{SchRem} \tag{2}$$

We assume that all events in each related chain of events have the same priority, equal to the priority of the event that generate the chain, so we can model each chain of events has a task. Note that all the events in a channel are executed by the same task: the EventScheduler.

4 Schedulability Analysis of a Distributed Hard Real-Time System

4.1 Single Processor Schedulability Analysis

The worst-case response time of a task on a single processor can be computed using the schedulability analysis for arbitrary deadlines (deadlines can be greater than task periods). The following equation gives the worst-case response time of a given task i:

$$r_i = max_{q=0,1,2,\cdots}(J_i + \omega_i(q) - qT_i) \tag{3}$$

$$\omega_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \lceil \frac{J_j + \omega_i(q)}{T_j} \rceil C_j + \tau_i(\omega_i(q)) \quad (4)$$

Where r_i is the worst-case response time of task i, J_i is the task i release jitter and T_i is the task period. C_i is the worst-case computation time of task i and B_i is a blocking factor representing the longest time that task i could be blocked by lower priority tasks. The summing factor represents the interference from higher priority tasks and $hp(i)$ is the set of tasks of higher priority than i. The tick scheduling overheads for a task i over a window of width ω is represented by $\tau_i(w)$ and the following equation:

$$\tau_i(\omega_i(q)) = LC_{clk} + min(L, K)C_{QL} + max(K - L, 0)C_{QS} \quad (5)$$

Using Equation 3 we can analyze the worst-case response times of event chains in a channel, by substituting the execution time of a task, C by C_{chain} in Equation 1.

As noted before, the execution of an session handler cannot be preempted by a higher priority event being inserted in the schedulable event set. We account this as release jitter (release jitter is the worst-case delay between a *task* arriving and being released [7]). We will consider that higher priority events suffer from a release jitter equal to the handler with the highest worst-case execution time, when executed by lower priority events. We have made some experiments accounting for release jitter and concluded, that its effect is significant.

An important aspect of an *RT-Appia* stack is that different real-time channels may share sessions at one or more layers. Multi-channel sessions can easily suffer priority inversion problems due to: *i*) concurrency control in the access to the shared data structures; *ii*) inter-channel coordination features that are specific of each session. The support for multi-channel sessions is done using the blocking factor B_i in Equation 4.

4.2 Holistic Scheduling Theory

In [7], the authors extended the schedulability analysis for arbitrary deadlines on a single processor to a distributed hard real-time system where tasks with arbitrary deadlines communicate using a TDMA protocol. According to the holistic scheduling theory presented, a destination task of a message m inherits a release jitter equal to:

$$J_{d(m)} = r_{s(m)} + a_m + r_{deliver} \quad (6)$$

In which $r_{s(m)}$ is the worst-case response time of the sender task of message m and can be computed using Equation 3; a_m is the worst-case time m takes to arrive at the destination processor communication adapter; and $r_{deliver}$ is the worst-case time to deliver the message to the destination task. In our analysis, $r_{deliver}$ is the worst-case time an interrupt handler and/or a task handler assigned to the network controller spends inserting an event in a channel.

Our goal in using this approach is to easily integrate other models of real-time communication networks by replacing the computation model for a_m. To illustrate this idea, we replaced the computational model of the TDMA protocol with the computational model for computing the worst-case latency of a given hard real-time message in a CAN field bus. In the following paragraphs we provide a brief description of the CAN analysis.

4.3 CAN

The analysis that bounds the worst-case latency of a given hard real-time message in CAN was derived by [8] and based on the following assumptions:

1. The deadline of a message must be less or equal to its period.
2. The message queuing window (i.e. the message queuing jitter) must be less than its period.
3. The bus controller must not release the bus to lower priority messages if there are higher priority messages pending.

The worst-case response time of a given message m is defined as the longest time between the queuing of m and the latest time the message arrives at the destination processor and is defined as:

$$R_m = \omega_m + C_m \tag{7}$$

The queuing delay ω_m is given by:

$$\omega_m = E(\omega_m + C_m) + B_m + \sum_{\forall j \in hp(m)} [\frac{\omega_m + J_j + \tau_{bit}}{T_j}] C_j \tag{8}$$

The set $hp(m)$ is the set of messages of higher priority than m and the summing represents the interference from higher priority messages. T_j is the period of a message j and J_j is the queuing jitter of the message j. The term τ_{bit} is the bit time of the bus.

The time spent by the message in the queue is given by the sum of the error recovery overheads (modeled by the function E).

B_m is the longest time that the message m can be delayed by lower priority messages (this is equal to the time taken to transmit the largest lower priority message because once transmission of a message has begun it cannot be preempted by transmission of a higher priority message) and is given by:

$$B_m = max_{\forall k \in lp(m)} C_k \tag{9}$$

Where $lp(m)$ is the set of lower priority messages.

C_m is the upper bound on frame transmission time with maximum insertion of stuffed bits and is given by:

$$C_m = ([\frac{53 + 8s_m}{5}] + 66 + 8s_m)\tau_{bit} \tag{10}$$

The term s_m denotes the size of message m in bytes. The factor gives the maximum number of stuff bits. This equation takes in account the minimum three bit bus idle period that necessarily precedes any data or remote frame transmission (the values used are for CAN 2.0B).

By using Equation 7 for computing a_m in Equation 6, we can extend the holistic theory to a CAN network.

5 Case Study

To illustrate our analysis technique and compare results, we have adapted the example presented in [10]. The example uses RELCAN, a reliable multicast protocol for real-time applications that has been proposed by Rufino et al. [5].

5.1 Protocol Overview

RELCAN and EDCAN provide a totally ordered atomic broadcast service to overcome the problem of CAN field bus being in an inconsistent state under particular circumstances [1]. The protocol stack of RELCAN, illustrated in Figure 1, is composed of RELCAN, EDCAN and CAN itself. CAN is required by both RELCAN and EDCAN, and EDCAN is required by RELCAN.

The RELCAN protocol assures a reliable communication service, but with less transmission time overhead, in the best case, than EDCAN. The sender uses a two phase protocol, the first phase consisting of putting a DATA message on the CAN bus, after which it waits for the confirmation of correct transmission from the CAN controller. The second phase consists of sending a CONFIRM message signaling that no retransmissions are required. The receiver delivers the message to the upper layer when receiving the message for the first time from the CAN layer, saves a copy for possible retransmission and starts a timer alarm. If the CONFIRM message is received before the alarm expires, then the alarm is canceled. If not, then it means that the sender has probably failed, and the receivers should retransmit the message using the EDCAN services. In this example we assume RELCAN in the best case (no alarm and EDCAN services are required).

[1] For a more detailed explanation of this problem and of RELCAN and EDCAN, please refer to [5].

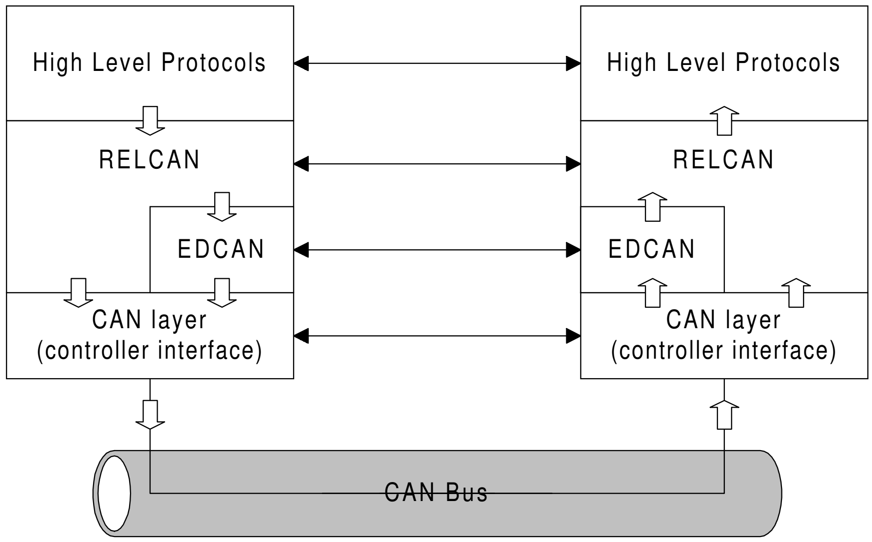

Figure 1: Protocol Stack of RELCAN

In the first place we must define the set of events required. The RELCAN sender handles two events and generates three events:

- handles one event from the upper layer (RELCAN.REQ) and generate one event for the CAN layer (DATA.REQ).
- handles one event from the CAN layer (DATA.CNF) and generate two events: RTR.REQ for the CAN layer and RELCAN.CNF to the upper layer.

The recipient handles two events and can generates one event:

- handles the DATA.IND event from the CAN layer and generates the RELCAN.IND event to the upper layer.
- handles the RTR.IND event from the CAN layer.

As mentioned in section 3, a task is assigned to each *related chain* of events. Three tasks for the sender and two tasks for the recipient are used:

- RS1 - The first sender task handles the RELCAN.REQ event and sends the message in a CAN data frame (DATA.REQ event).
- RS2 - The second sender task handles the DATA.CNF event and sends the CONFIRM message (RTR.REQ).
- RR1 - The first recipient task (RR1) handles the reception of the message from the CAN layer (DATA.IND event) and delivers it to the upper layer (RELCAN.IND).

cpu 1				cpu 2				cpu 3			
task	*Prio*	*J*	*WCRT*	*task*	*Prio*	*J*	*WCRT*	*task*	*Prio*	*J*	*WCRT*
RS1	0	0	150	RS1	0	0	150	RS1	0	0	150
RS2	1	456	756	RS2	1	685	985	RS2	1	761	1061
RC1	2	456	906	RC2	2	686	1135	RC3	2	761	1211
RR12	3	685	1285	RR11	3	456	1056	RR11	3	456	1056
RR13	4	761	1511	RR13	4	761	1511	RR12	4	685	1435
RR22	5	1596	2496	RR21	5	1138	2038	RR21	5	1138	2038
RR23	6	1748	2798	RR23	6	1748	2798	RR22	6	1596	2646
msg	*Prio*	*C*	*WCRT*	*msg*	*Prio*	*C*	*WCRT*	*msg*	*Prio*	*C*	*WCRT*
Data.req	0	153	306	Data.req	2	153	535	Data.req	4	153	611
Rtr.req	1	76	382	Rtr.req	3	76	611	Rtr.req	5	76	687

Table 1: RELCAN task set results using the extended holistic theory.

- RR2 - Handles the reception of the CONFIRM message (RTR.IND event).

We also defined a third sender task RC to generate the RELCAN.CNF event. Please note that in handling DATA.CNF event, two events are generated (RTR.REQ and RELCAN.CNF), respectively.

5.2 Timing Analysis Results

In order to obtain the timing analysis tool, we have started by implementing a software component that computes the worst-case latency of a given hard real-time message in CAN (Equation 7). Then we integrated this component in the software tool used by Tindell in [7], by replacing the TDMA protocol (a_m in equation 6). The resulting tool implements the holistic scheduling analysis presented in Section 4.

The developed scenario consists of three message transmissions by 3 nodes. Each node sends its message to the others two nodes on the bus, which receive it correctly, the same happening with the confirmation message of the second phase.

The results presented in Table 1 refer to our case-study, computed with the tool introduced above. The table presents Worst Case Response Time (WCRT) for all tasks (the values are presented in μ seconds). The following parameters were used: The following parameters were used, for all tasks: $T_{clk} = 1$, $C_{clk} = 0$, $C_{QL} = 0$, $C_{QS} = 0$ (no scheduler overheads), $T_i = 3000$, $D_i = T_i$ (task deadline equal to its period) and a worst-case computation time $C_i = 150$, for each task. We also consider $J_i = 0$[2], $B_i = 0$ (no initial jitter, nor blocking time) and an $r_{deliver} = 0$, for all tasks. For the message transmission delays, a 1 *Mbit* extended CAN network was considered, without errors. The *prio* column is the priority assigned to tasks and messages and J is the computed jitter suffered by each task. As you can see from Table 1, all tasks meet their deadlines.

[2]This can be observed by the higher priority task in each node having $J_i = 0$.

cpu 1				cpu 2				cpu 3			
task	*Prio*	*J*	*WCRT*	*task*	*Prio*	*J*	*WCRT*	*task*	*Prio*	*J*	*WCRT*
RS1	0	150	300	RS1	0	150	300	RS1	0	150	300
RS2	1	756	1056	RS2	1	985	1285	RS2	1	1061	1361
RC1	2	756	1206	RC2	2	985	1435	RC3	2	1061	1511
RR12	3	985	1585	RR11	3	756	1356	RR11	3	756	1356
RR13	4	1061	1811	RR13	4	1061	1811	RR12	4	985	1735
RR22	5	2046	2946	RR21	5	1588	2488	RR21	5	1588	2488
RR23	6	2048	3248	RR23	6	2048	3098	RR22	6	1896	2946
msg	*Prio*	*C*	*WCRT*	*msg*	*Prio*	*C*	*WCRT*	*msg*	*Prio*	*C*	*WCRT*
Data.req	0	153	306	Data.req	2	153	535	Data.req	4	153	611
Rtr.req	1	76	382	Rtr.req	3	76	611	Rtr.req	5	76	687

Table 2: RELCAN task set results using the extended holistic theory with *jitter*.

Because in this model, a task cannot receive more than one message within the same period we must define additional recipient tasks, on all nodes:

- (RR1x) Refers to RR1 handling a message (DATA.IND), sent by node x.
- (RR2x) Refers to RR2 handling a message (RTR.IND), sent by node x.

Task RR12 in cpu 1, handles the DATA.IND sent by cpu 2, task RR23 in cpu 2 handles the RTR.IND sent by cpu 3, and so on.

The results presented in Table 2 refer to the same example by considering that tasks suffer from release jitter, due to non-preemptive session handlers, as described in Section 3. Note that each higher priority task now suffers from a initial jitter of $J_i = 150$, which corresponds to the session handler of low priority events, with the highest WCET (each lower priority task has $J_i = 0$). It can be seen that with the effect of jitter some tasks miss their deadlines.

Table 3 refers to the same example computed with a tool implementing the timing offset model (results generated using the method of [10] and presented here for convenience of the comparison). A transaction is composed by a set of tasks that execute with given offsets in relation to its initial value (in this model $end = offset + WCRT$). Note that since the transactions are not synchronized between them, each one can start at any instant during the execution of the others. Each transaction corresponds to the chain of events generated by a user request to the RELCAN layer. All the tasks (and messages) in transaction 1 have higher priority than the ones in transaction 2, which in turn has higher priority than the ones in transaction 3.

By comparing the results in Table 1 and 3, it is possible to see that the latest task misses its deadline in the offset model. On the other hand, task response times in transaction 1 are lower than the times obtained for tasks in Table 1, but this does not occur with the other transactions.

transaction 1				transaction 2				transaction 3			
task	*cpu*	*offset*	*end*	*task*	*cpu*	*offset*	*end*	*task*	*cpu*	*offset*	*end*
RS1	1	0	150	RS1	2	0	300	RS1	3	0	600
RS2	1	456	606	RS2	2	835	1135	RS2	3	1135	1735
RC	1	456	756	RC	2	835	1435	RC	3	1135	2035
RR12	2	456	606	RR11	1	835	1285	RR11	1	1135	2035
RR13	3	456	606	RR13	3	835	1135	RR12	2	1135	1885
RR22	2	835	985	RR21	1	1517	1967	RR21	1	2117	3017
RR23	3	835	985	RR23	3	1517	1817	RR22	2	2117	2867
msg		*offset*	*end*	*msg*		*offset*	*end*	*msg*		*offset*	*end*
DATA		150	456	DATA		300	835	DATA		600	1135
CONFIRM		606	835	CONFIRM		1135	1517	CONFIRM		1735	2117

Table 3: RELCAN task set results using the extended timing offset model.

cpu 1				cpu 2				cpu 3			
task	*Prio*	*J*	*WCRT*	*task*	*Prio*	*J*	*WCRT*	*task*	*Prio*	*J*	*WCRT*
RS1	0	0	150	RS1	0	0	150	RS1	0	0	150
RS2	1	456	756	RS2	1	685	909	RS2	1	761	985
RC1	2	456	906	RC2	2	686	1059	RC3	2	761	1135
RR12	3	685	1209	RR11	3	456	1056	RR11	3	456	1056
RR13	4	761	1435	RR13	4	761	1435	RR12	4	685	1359
RR22	5	1596	2496	RR21	5	1138	2267	RR21	5	1138	2267
RR23	6	1748	2722	RR23	6	1748	2722	RR22	6	1596	2646
msg	*Prio*	*C*	*WCRT*	*msg*	*Prio*	*C*	*WCRT*	*msg*	*Prio*	*C*	*WCRT*
Data.req	0	153	306	Data.req	1	153	459	Data.req	2	153	535
Rtr.req	3	76	611	Rtr.req	3	76	687	Rtr.req	5	76	687

Table 4: RELCAN task set results with changed message priorities.

This is because in the timing offset module, it can happen that the largest task of a higher priority transaction is scheduled at any time during the execution of the current transaction. The pessimism that is introduced by the holistic model, was verified by validating these results manually (for a best case). We also experimented with giving higher priority to the DATA.REQ messages, instead of giving higher priorities to messages in a given cpu. As can be seen from Table 4, slightly better results were obtained. Now the response time for RR1 tasks has decreased which means that all nodes receive the RELCAN DATA message earlier.

From the results presented in Table 2, we can conclude that the effect of jitter is an important aspect that must be taken in consideration when designing the event scheduler that supports this class of event-based protocol composition frameworks. We are currently studying methods to minimize the jitter in *RT-Appia*.

6 Conclusion

In this paper we have presented a method to analyze the timing behavior of real-time protocol stacks implemented using event-based composition frameworks. The method proposed here, which is based on the holistic approach [7], has been compared with a related method based on the offset timing analysis [8]. In our experiments, despite the pessimism inherent to the approach, the holistic model can achieve better results. On the other hand, for a small number of transactions, the offset timing analysis is more optimistic. Unfortunately, this optimism is lost as the number of transactions increases. This is due to the fact that interference caused by higher priority transactions in the lower ones introduces to much pessimism.

We verified this having computed our case-study with both methods for several number of nodes (and the correspondent number of transactions). With two nodes the offset timing analysis gave better results. However as the number of transactions increases, the worst case response times of tasks approaches the ones computed with the holistic model and the later gave little better results.

We think that the pessimism introduced by both methods is due to the fact that the computed response times follows a exponential form, with the increasing number of nodes and transactions.

We believe that the offset timing analysis is more appropriate for systems that support asynchronous events and when its possible to have a small number of transactions. For more complex systems in which all events are synchronized, the holistic method may be preferable. We intend to support both methods and use the most appropriate according to the application scenario.

References

[1] Matti A. Hiltunen, Xiaonan Han, and Richard D. Schlichting. Real-time issues in Cactus. Technical Report AZ85721, Department of Computer Science, University of Arizona, Tucson, 1996.

[2] Matti A. Hiltunen, Richard D. Schlichting, Xiaonan Han, Melvin Cardozo, and Rajsekhar Das. Real-time dependable channels: Customizing qos attributes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):600–612, June 1999.

[3] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 707–710, Phoenix, Arizona, April 2001. IEEE.

[4] J. Rodrigues, H. Miranda, J. Ventura, and L. Rodrigues. The design of RTAppia. In *Proceedings of the Sixth IEEE International Workshop on Object-oriented Real-Time Dependable Systems*, pages 275–282, Rome, Italy, January 2001. IEEE.

[5] José Rufino, Paulo Veríssimo, Guilherme Arroz, Carlos Almeida, and Luís Rodrigues. Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th IEEE International Symposium on Fault-Tolerant Computing*, pages 150–159, Munich, Germany, June 1998. IEEE.

[6] John A. Stankovic and Krithi Ramamritham. Editorial: What is predictability for real-time systems? *The Journal of Real-Time Systems*, 2:247–254, 1990.

[7] Ken W. Tindell. Holistic schedulability analysis for distributed hard real-time systems. Technical Report YCS197, Department of Computer Science, University of York, April 1993.

[8] Kenneth William Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS221, Department of Computer Science, University of York, January 1994.

[9] Franco Travostino, Ed Menze, and Franklin Reynolds. Paths: Programming with system resources in support of real-time distributed applications. In *Proceedings of the 2nd IEEE Workshop on Object-Oriented Real-Time Dependable Systems*, Laguna Beach, CA, February 1996.

[10] J. Ventura, J. Rodrigues, and L. Rodrigues. Response time analysis of composable micro-protocols. In *Proceedings of the 4rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, pages 335–342, Magdeburg, Germany, May 2001. IEEE.