Modeling Real-Time and non Real-Time Interoperability

Dietmar Prisching [a]
dietmar.prisching@avl.com

Bernhard Rinner [b]
rinner@iti.tugraz.at

Abstract

Most real-world automation systems (AS) require both real-time (RT) as well as non real-time (NRT) functionality. Modern AS are more frequently realized as a single platform solution in order to reduce their development time. This trend is driven by the advances in processor technology. The interoperability between RT and NRT processing is an important parameter in a single platform solution.
This paper focuses on the modeling the performance metrics of RT-NRT interoperability based on response time analysis (RTA), which is extended by RT-NRT overhead. The performance metrics are (i) the response times of RT tasks, (ii) the CPU utilization of the RT load and (iii) NRT timing properties. First results demonstrate the feasibility of our approach.

Keywords: *automation system; real-time; hybrid computation; performance model;*

1. Introduction

Most real-world automation systems (AS) include also functions that do not require real-time guarantees, i.e., they are comprised of a *real-time (RT)* part and a *non real-time (NRT)* part. In a typical AS, the RT part is responsible for control, data acquisition, signal conditioning and monitoring. The NRT part is responsible for data post processing, visualization, data persistence, system parameter settings and, especially, for the graphical user interfaces (GUI). Modern AS are more frequently realized as *a single platform solution*, because the development time is smaller than the development time of the RT and NRT parts on different platforms. This trend is also driven by the advances in processor technology. Processing power has increased considerably over the last years, and a single processor may now offer sufficient power to run AS with RT and NRT parts.

An important parameter for the combination of NRT and RT processing is *interoperability,* which is defined as the ability to run the NRT part along with the RT part and vice-versa [6]. An important requirement for a RT-NRT combination is that the RT part is preferred over the NRT part, and the NRT part does not influence the RT part. This paper focuses on a model to calculate performance metrics of complex asynchronous and fixed priority systems that consist of a combination of RT and NRT processing. The performance metrics are (i) the response times of RT tasks, (ii) the CPU utilization of the RT load and (iii) NRT timing properties. The modeling is based on a *response time analysis (RTA)* derived from Burns [2], [3] and Bernat [1]. RTA is an effective, simple and flexible technique that allows the modeling of most aspects of fixed priority real-time systems. We have extended this approach to quantify the interoperability of RT and NRT processing. Therefore, we include the overhead of RT and NRT processing as well as the overhead of the RT processing itself into the RTA formulation. Furthermore, we show that RTA can be used to retrieve the NRT computation distribution in dependency on a RT processing load.

In Section 2 we briefly summarize RTA. Section 3 introduces our analytic model and Section 4 describes the implementation of our modeling approach and presents results. Finally, Section 5 concludes this paper with a summary and a discussion on further work.

2. Response Time Analysis (RTA)

Based on [2] and [1] is the computation of the finish time of the k^{th} invocation of a task τ_i $F_i(k)$ the smallest $\omega \geq 0$ such that:

$$\omega = kC_i + \delta_i(k) + I_i(\omega) \qquad \textbf{(2-1)}$$

- $I_i(\omega)$ is the interference of tasks of higher priority than τ_i during interval $[0,\omega)$ and is given by:

$$I_i(\omega) = \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j$$

C_i…Computation time of task τ_i

[a] AVL List GmbH, Graz AUSTRIA
[b] Institute for Technical Informatics, Graz University of Technology, AUSTRIA

T_j…Cycle time of task τ_j
$hp(\tau_j)$… is the set of tasks of higher priority than τ_i

- $\delta_i(\omega)$: The idle time at level i at ω is the amount of time the processor can be used by tasks of lower priority than τ_i during period of time [0,ω). The amount of idle time at the arrival of each task invocation (arrival time (A_i)) is of special interest. Thus, it can also be written as $\delta_i(k) = \delta_i(A_i(k))$.

The computation of δi(t) is more complex because it can not be computed directly[1]. To compute the amount of idle time at level i between [0,t), a virtual task τ_v is introduced that has a period and a *deadline* (D_v) equal to the time t: $\tau_v = (T_v = t, D_v = t, C_v)$. The maximum time the processor can be used by tasks of lower priority than τ_i is the maximum computation time, C_v, that makes task τ_v meet its deadline. The scheduling test is performed by solving Eq. (2-2) for ω and checking whether $\omega \le D_v$:

$$\omega = C_v + I_i^{\bullet}(\omega) \quad \textbf{(2-2)}$$

where the amount of interference at level i including task τ_i, denoted by $I_i^{\bullet}(\omega)$, is given by:

$$I_i^{\bullet}(\omega) = \sum_{\tau_j \in hep(\tau_{ii})} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j \quad \textbf{(2-3)}$$

$hep(\tau i)$… is the set of tasks of higher or equal priority than task τi

Context switch (CS) overhead is of special interest and related work can be found at e.g. [4] and [2]. A common approach to model CS is to introduce a fictitious task that belongs to the periodic task with the same period T but with computation time C_{CS}.

3. Modeling RT-NRT Interoperability

Performance metrics of interest in this work are (i) the response times (R_i) of RT tasks, (ii) the CPU utilization of a RT load and (iii) NRT timing properties. We start from a process model made up of n RT tasks and a single NRT task. When no RT task is ready to execute, the lowest priority task (NRT task) is scheduled.

Response times
First, the computing of the idle time distribution for a individual level i ($\delta_i(0,T)$) based on Eq. (2-2) [1] has to be carried out. C_v is incremented from 0 to T until the condition $\omega \le D_v$ fails (see Figure 3-1). The computed values of ω can be used to derive the idle time distribution. Each monotonic rise of ω with regard to C_v corresponds to an idle time, and each discontinuous step of ω corresponds to an interference at level i (see Figure 3-1). This computation is carried out for each level i.

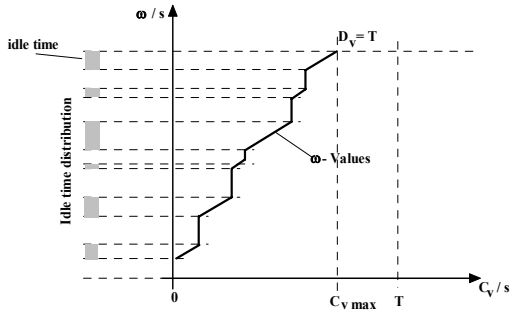


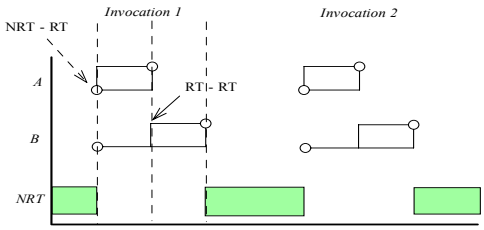
Figure 3-1: The distribution of the idle time at each level can be derived from the computed values of ω.

Based on the idle time distribution the computation of $F_i(k)$ is done with Eq. (2-1). In order to introduce context switch times in our approach, we also need the start times of the multiple task invocations. We can compute these start times with a simple modification of Eq. (2-1). We simply set the k^{th} computation time of task τ_i as the smallest possible time unit that is used in the model. Therefore, the computation of start time $S_i(k)$ is based on the fact that this time is equivalent to the worst case response time of a task with the same priority as task τ_i and a computation time C = 1(smallest time unit) + (k-1)C_i + $\delta_i(k)$.

$$\omega = 1 + (k-1)C_i + \delta_i(k) + I_i(\omega) \quad \textbf{(3-1)}$$

Next, we introduce context switch (CS) overhead into our model. So, for further clarification we denote a operating system CS as NRT – RT and a thread CS within the same process as RT-RT(SP) and a thread CS across different processes as RT-RT(DP) (thread, process see [8]). The time for a NRT-RT is defined as $C_{NRT\text{-}RT}$, for RT-RT(SP) as $C_{RT\text{-}RT(SP)}$ and for RT-RT(DP) as $C_{RT\text{-}RT(DP)}$. With the calculated start and finish times of the tasks it is possible the retrieve the kind of CS that occurs for a certain invocation of a task τ_i. This happens as follows. In Figure 3-2 we see the invocations of a task set that consist of two RT tasks (A, B). At an invocation of A an OS switch happens (NRT – RT). Furthermore, also a thread CS between the NRT task and A occurs. We define this as a RT-RT(DP). At the finish of A a thread CS to B happens. Dependent whether A or B belongs to the same or different processes, a RT-RT(SP) or RT-RT(DP) occurs.

[1] Due to the fact that invocations of tasks cannot always be completely counted at time any time t, the simple equation $\delta_i(t) = t - \sum_{\tau_j \in hep(\tau_i)} \left\lceil \frac{\omega}{T_j} \right\rceil \cdot C_j$ is not sufficient for computing the idle time.



Invocation k	A		
	$\Phi_A^{NRT-RT}(k)$	$\Phi_A^{RT-RT(SP)}(k)$	$\Phi_A^{RT-RT(DP)}(k)$
1	1	0	1
2	1	0	1
	B		
	$\Phi_B^{NRT-RT}(k)$	$\Phi_B^{RT-RT(SP)}(k)$	$\Phi_B^{RT-RT(DP)}(k)$
1	0	1	0
2	0	1	0

Figure 3-2: Schematic CS example

The information whether a CS occurs prior to a certain task, the invocation can be represented by step functions Φ ($\Phi^{\text{NRT-RT}}$, $\Phi^{\text{RT-RT(SP)}}$, $\Phi^{\text{RT-RT(DP)}}$) (Φ for A and B see bottom of Figure 3-2). The interference caused by the NRT-RT overhead at level i including CS of task i, denoted by $I_i^{NRT-RT}(\omega)$, is given by:

$$I_i^{NRT-RT}(\omega) = \sum_{\tau_j \in hep(\tau_{ij})} \sum_{k=0}^{\left\lceil \frac{\omega}{T_j} \right\rceil} \Phi_j^{NRT-RT}(k) \cdot C_{NRT-RT}$$

hep(τ_{ij})… is set of tasks of higher or equal priority than task τ_i also including task τ_i.

In the same way, the interference caused by the RT-RT(SP) and RT-RT(DP) overhead respectively at level i including CS of task i, is given by:

$$I_i^{RT-RT(SP)}(\omega) = \sum_{\tau_j \in hep(\tau_{ij})} \sum_{k=0}^{\left\lceil \frac{\omega}{T_j} \right\rceil} \Phi_j^{RT-RT(SP)}(k) \cdot C_{RT-RT(SP)}$$

and

$$I_i^{RT-RT(DP)}(\omega) = \sum_{\tau_j \in hep(\tau_{ij})} \sum_{k=0}^{\left\lceil \frac{\omega}{T_j} \right\rceil} \Phi_j^{RT-RT(DP)}(k) \cdot C_{RT-RT(DP)}$$

The interferences from CS between NRT and RT, RT-RT(SP) and RT-RT(DP) can be summarized and we write:

$$I_i^{CS} = I_i^{NRT-RT} + I_i^{RT-RT(SP)} + I_i^{RT-RT(DP)}$$

Therefore, NRT-RT interference is included into the response time calculation and Eq. (2–1) and Eq. (2-2) can be extended by $I_i^{CS}(\omega)$.

CPU Utilization

With the previous results the rate (frequency) of the appearance of certain CS is known. So we can denote $f_{\text{NRT-RT}}$ as the frequency of NRT-RT, with $f_{\text{RT-RT(SP)}}$ as the frequency of RT-RT(SP) and with $f_{\text{RT-RT(DP)}}$ as the frequency of RT-RT(SP). Based the formulation from [5] the CPU utilization for a specific RT-load is given by:

$$U_\Pi = \sum \frac{C_j}{T_j} + f_{NRT-RT} \cdot C_{NRT-RT} + f_{RT-RT(SP)} \cdot C_{RT-RT(SP)} + f_{RT-RT(DP)} \cdot C_{RT-RT(DP)} \quad \textbf{(3-2)}$$

NRT Properties

The following properties of the RT-NRT interoperability can be derived from the above calculations.

- The $\delta_i(0,T)$-distribution at the lowest priority level i is equal to the NRT computation distribution.
- The response time of the lowest priority task at the critical instant corresponds with the longest suspension of NRT computation.
- $f_{\text{NRT-RT}}$ is a crucial factor for NRT-RT interoperability overhead. NRT update-functionality (e.g. graphic) is disturbed by a higher $f_{\text{NRT-RT}}$. Thus the RT-NRT interoperability performance isn't 100% at high $f_{\text{NRT-RT}}$ (see [7]).

4. Implementation

Computing the timing properties

The main steps for computing the timing properties for a RT-NRT processing can be summarized as follows:

1. The timing metrics of a RT-processing that does not consider interoperability are calculated.
2. With the calculated start and finish times of the tasks the step functions ($\Phi^{\text{NRT-RT}}$, $\Phi^{\text{RT-RT(SP)}}$, $\Phi^{\text{RT-RT(DP)}}$) are determined.
3. With our extended equations start and finish times of the tasks are calculated that consider NRT-RT, RT-RT(SP) and RT-RT(DP). However, these results can change the step functions and so step 2 and step 3 have to be repeated until the step functions do not change between the iterations.
4. Finally, the CPU utilization from the RT load based on Eq. (3-2) is calculated as well as the NRT properties can be determined based on the timing metrics results (see chapter 3, NRT Properties).

Example Task Set

Up-to-date timing requirements are data acquisition, monitoring and control with 10 kHz. So in this example, we demonstrate the applicability of our model. We work

with three tasks (thread T1, T2 and T3) shown in Table 4-1 and each thread belong to an own process. We compute this example excluding interoperability (EI) and including interoperability (II). In Figure 4–1 the resulting multiple invocations for EI and II are depicted. Table 4-2 shows the calculated timing properties for both in detail.[2] Without the context switches T1, T2 and T3 keep its deadlines. However, when context switch times are included T3 misses its deadline (see Table 4-2).

Table 4-1: Example Task Set

Process	Task	T / µs	C / µs	D / µs	Priority
A	T1	100	20	100	1
B	T2	200	50	100	2
C	T3	400	20	100	3

Table 4-2: Timing properties of EI and II

			Excl. Interoperability (EI)			Incl. Interoperability (II)		
Task	**K**	**A / µs**	**S / µs**	**F / µs**	**R / µs**	**S / µs**	**F / µs**	**R / µs**
T1	1	0	0	20	20	6	26	26
	2	100	100	120	20	103	123	26
	3	200	200	220	20	206	226	26
T2	1	0	20	70	70	29	79	79
	2	200	220	270	70	229	279	79
	3	400	420	470	70	429	479	79
T3	1	0	70	90	90	82	128	**128**
	2	400	470	490	90	482	528	**128**

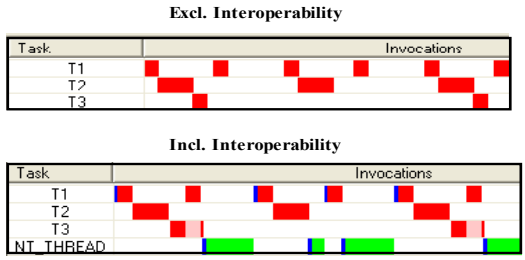


Figure 4-1: Calculated multiple invocations excluding and including interoperability.

The calculated CPU utilization for EI is U_{Π}= 50 %. At II $f_{NRT\text{-}RT}$ follows with 7450 Hz and $f_{RT\text{-}RT(DP)}$ with 19950 Hz. Therefore, the CPU utilization with Eq. (3-2) is U_{Π}= 60.48 %, which corresponds better to a real situation. The longest suspension from NRT (NT_THREAD) is 128 µs which is much longer than the recommended maximum suspension time of 7 ms for the NRT part. Thus, an $f_{NRT\text{-}RT}$ with 7450 Hz disturbs the NRT update-functionality, so that, e.g., the NRT graphical user interface can be significantly affected.

[2] For a PENTIUM II (434 MHz) $C_{NRT\text{-}RT}$ = 3 µs, $C_{RT\text{-}RT(DP)}$ = 3 µs and $C_{RT\text{-}RT(SP)}$ = 2 µs (Worst case context switch time)

5. Discussion

We have presented a process model for computing the timing properties of complex systems with real-time (RT) and non real-time (NRT) processing. Our modeling technique is based on the response time analysis (RTA). We have extended the RTA formulation to include RT and NRT interoperability overhead for the computation of response times (R_i) of task and CPU utilization of a RT load. Furthermore, we show that the RTA formulation can be used to obtain NRT timing properties that are important for the development and operation of most AS. For further work, we will optimize and speed-up the idle time computation $\delta_i(0,T)$, take offsets and sporadic tasks formulations into for our model and finally, we will include interrupts that are executed in the context of the pre-empted task.

References

[1] BERNAT. G., 2002 Response Time Analysis of Asynchronous Real-Time Systems, Uni. of York

[2] BURNS, A., 1994: Preemptive Priority-Based Scheduling: An Appropriate Engineering Approach, Advances in RealTime Systems, 1994 225-248.

[3] BURNS, A., WELLINGS. A., 2001: Real-Time systems and programming languages. Addision Wesly 3rd edition, 2001

[4] BUSQUETS, M., WELLINGS A., 1996: Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real Time Systems. Uni. of York

[5] BUTTAZZO, G. C., 1997: Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, London, Kluwer Academic Publishers

[6] http://www.dedicated-systems.com

[7] PRISCHING, D., RINNER, B., 2003: Thread-based analysis of embedded applications with real-time and non real-time processing on a single-processor platform, embedded world 2003 Congress, Nürnberg

[8] SILBERSCHATZ. A., GALVIN, P., GAGNE, G., 2000: Applied Operating System Concepts, Hohn New York, Wiley & Sons, Inc.