Quantifying the Flexibility of Real-Time Systems

Rafik Henia, Alain Girault, Christophe Prévot, Sophie Quinton, Laurent Rioux

Quantifying the Flexibility of Real-Time Systems

Rafik Henia[1], Alain Girault[2], Christophe Prévot[1,2], Sophie Quinton[2], Laurent Rioux[1]
[1] Thales Research & Technology
[2] Inria Grenoble Rhône-Alpes

1. INTRODUCTION

The life cycle of many safety or mission critical real-time systems, such as avionic systems, satellites or software defined radios, is superior to 10 years whereas the supporting ICT technologies have a much faster evolution rate. This mismatch between system life cycle on one side and software life cycle on the other side is a challenge for performance engineers. It is therefore essential for commercial offers to allow technology evolutions and upgrades of the architecture and functions *after the initial system deployment*. Besides, this must be done while preserving the satisfaction of timing performance requirements over the entire system lifetime.

Our objective is to develop novel techniques to quantify the ability of a real-time system to cope with future software evolutions, while remaining schedulable. We propose to define the flexibility of a system as its ability to schedule a new software function and show how this concept can be used to anticipate at design time schedulability bottlenecks that may prove problematic, after deployment, for system evolution. In this paper we focus on an evolution scenario where a single task is added to an existing system running on a single-core processor. Our task model here is restricted to independent periodic tasks with implicit deadlines and the scheduling policy is static priority preemptive. Of course our goal is to generalize our approach to more complex systems.

The research area that is most closely related to our problem is sensitivity analysis [9, 2, 5, 7]. Sensitivity analysis is used (1) to provide guarantees on the schedulability of a system in case of uncertainty on the system parameters, or (2) given a non schedulable system, to find a set of changes that lead to a schedulable system.

Most related work on sensitivity analysis addresses changes of one type of parameter, e.g., changes of task worst-case execution times (WCETs) or changes of periods — but not both. In [2] for example, the authors analyze the sensitivity of a system to changes in the arrival frequency of all tasks, or in the WCET of all tasks. In [7], the authors are interested in sensitivity analysis of systems with shared resources, for which they study the impact of an increase of the WCETs on system schedulability.

Dealing with different parameters (i.e. periods and priorities) is the approach used in [8]. Here the authors define the sensitivity using evolutionary algorithms and a stochastic approach. It will be interesting to compare our results with this method.

In contrast with the above mentioned approaches, we do not allow parameters of the deployed system to be modified in according with industrial practice. But we need to handle at the same time changes in the period, priority and WCET of the task to be added. As a result, the set of possible system configurations that need to be checked is much smaller than in classical sensitivity analysis and we hope to be able to handle much more complex systems that what state-of-the-art sensitivity analysis can handle.

Other papers like [10, 1, 3] deal with flexibility related to task allocation, priority assignment or scenario based optimization. In [3], the authors use concepts similar to ours to define an optimal or a robust priority assignment. In [10], the authors deal with task allocation and priority assignment to maximize the extensibility of each task chain, where extensibility is the maximum execution time it is possible to increase the WCET before missing the deadline. In [1] the flexibility of a system is defined according to scenarios. To define the flexibility it is necessary to define possible scenarios of change.

Related work that deals with flexibility is used to define at design time what is the best task allocation or priority assignment. But most of the time in an industrial context, the designer cannot take into account only timing constraints and some functionalities have fixed parameters. Moreover we are interested in system evolution when most parameters of the system are already defined. It is at this point unclear if our problem (adding a task) can be precisely encoded into existing methods dealing with increased execution times.

As a summary, by restricting ourselves to a simple change scenario with industrial relevance, we hope to come up with an approach that can provide guidance for the evolution of complex systems, which cannot be covered by existing sensitivity analysis techniques. Finally, to the best of our knowledge, no existing work discusses the notion of limiting task, i.e. the task which will miss its deadline first. The final aim is to help the designer by giving him information to allow a system to evolve while respecting timing constraints.

2. SYSTEM MODEL

Unless otherwise specified all the parameters defined in the following have positive integer values. In particular, we assume a discrete time clock.

We consider a uni-processor real-time system S consisting of a finite set of n independent *tasks* scheduled according to the Static Priority Preemptive (SPP) scheduling policy. Each task τ_i, $i \in [1, n]$, is defined by a tuple (π_i, T_i, C_i) where π_i is the priority, T_i the period, and C_i an upper-bound on the worst case execution time (WCET). Each task τ_i has an *implicit deadline* $D_i = T_i$, meaning that a given activation of τ_i must finish before the next activation of τ_i.

We assume that different tasks have different priorities and use as convention that $\pi_i < \pi_j$ means that τ_i has a higher priority than τ_j. For the purpose of our evolution scenario, we also suppose that for any two tasks τ_i and τ_j with $\pi_i < \pi_j$, it is always possible to define a new task τ_{new} with $\pi_i < \pi_{new} < \pi_j$. This is done without loss of generality as priorities are only used to define a total order on tasks.

For each τ_i in S, $hp(i)$ is the set of tasks of S that have a higher priority than τ_i, while $lp(i)$ is the set of tasks of S that have a lower priority than τ_i.

The execution of a task τ_i is triggered periodically with period T_i and each activation of τ_i generates a new *instance*. A task instance is defined by its activation time, possible preemption delays, and finish time. Preemption delays are due to the task instance being blocked by higher priority task instances. Each instance of τ_i finishes at the latest after having being scheduled (i.e., not counting the preemption delays) for C_i units of time.

3. PROBLEM STATEMENT

We are interested in system evolutions that may happen after system delivery. We focus here on a simple evolution scenario where a new task τ_{new} defined by $(\pi_{new}, T_{new}, C_{new})$ is added to a schedulable system S, thus yielding the system $S_{new} = S \cup \{\tau_{new}\}$. From now on, τ_{new} will denote the task $(\pi_{new}, T_{new}, C_{new})$ and S_{new} will denote the system $S \cup \{\tau_{new}\}$.

We wish to answer the following question: Given a schedulable system S and a task τ_{new} with unknown parameters: how can we quantify the flexibility of S, that is, its ability to accommodate τ_{new} while remaining schedulable?

To address these issues, we start below by formalizing the notions of *evolution* and *flexibility* of a system.

Definition 1. Evolution: An *evolution* of a system S is a task τ_{new} where π_{new} belongs to $[\min_i \pi_i - 1, \max_i \pi_i + 1] \backslash \{\pi_i\}_i$ and (T_{new}, C_{new}) belongs to $\mathbb{N}^+ \times \mathbb{N}^+$, such that the CPU load after the evolution does not exceed 100%:

$$\frac{C_{new}}{T_{new}} + \sum_i \frac{C_i}{T_i} \leq 1$$

Definition 2. Valid evolution: S being a schedulable system, an evolution τ_{new} is valid for S if S_{new} is schedulable.

We denote by F_S the set of all valid evolutions for system S. Note that if τ_{new} is in F_S, then for any $0 < C'_{new} < C_{new}$, the task $(\pi_{new}, T_{new}, C'_{new})$ is also in F_S. Moreover for any $T'_{new} > T_{new}$, the task $(\pi_{new}, T'_{new}, C_{new})$ is also in F_S. The flexibility of a system quantifies its ability to schedule a new task. We focus in this paper on the flexibility w.r.t. C_{new}. The notion of flexibility w.r.t. T_{new} is left for future work.

Definition 3. Flexibility: Let S be a schedulable system. The *flexibility* of S is the partial function flex_S from $\mathbb{N}^+ \times \mathbb{N}^+$ into $\mathbb{N}^+$ such that $(\pi_{new}, T_{new}, \mathit{flex}_S(\pi_{new}, T_{new}))$ is in F_S and $\mathit{flex}_S(\pi_{new}, T_{new})$ is maximal.

The function *flex* returns, for all (π_{new}, T_{new}), the value of the maximum WCET such that $S \cup \{(\pi, T, \mathit{flex}_S(\pi, T))\}$ is schedulable. This function is partial as such an WCET may not exist. In the rest of the paper, we first focus on methods for computing *flex* and then we will show how we can use it for system design.

4. SLACK ANALYSIS

Let us first briefly recall the standard analysis used to establish the schedulability of systems of independent periodic tasks on a single processor under the SPP policy [6].

The *response time* of an instance of a task τ_i is the delay between its activation and its finish time. The *worst case response time* of τ_i (WCRT) is the maximum response time over all instances of τ_i; it is denoted r_i. A system is *schedulable* if $r_i \leq T_i$ for any task τ_i.

A *critical instant* of τ_i is an activation scenario that maximises the response time of τ_i. A task is said to be schedulable if and only if its WCRT is smaller than its (implicit) deadline: $r_i \leq T_i$. For the systems we consider, a critical instant of τ_i occurs whenever an activation of a task instance occurs simultaneously with the activation of all higher-priority tasks. The WCRT of τ_i can thus be obtained by computing:

$$r_i = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \tag{1}$$

To quantify the flexibility of a system, we start by introducing the *slack* of a task, which takes into account all the preemptions from higher priority tasks [4].

Definition 4. The slack of a task τ_i is the maximum value it is possible to increase C_i while keeping τ_i schedulable.

The slack Sl_i of a task τ_i can be computed using the algorithm presented in [4].

THEOREM 1. *Let S be a schedulable system and τ_{new} a task. S_{new} is schedulable if:*

$$\forall \tau_i \in lp(new), \left\lceil \frac{T_i}{T_{new}} \right\rceil C_{new} \leq Sl_i \tag{2}$$

and

$$r_{new} \leq T_{new} \tag{3}$$

PROOF SKETCH OF THEOREM 1. We have to prove that, for each $\tau_i \in S \cup \{\tau_{new}\}$, we have $r_i \leq T_i$. Since S is schedulable, this holds for each $\tau_h \in hp(new)$. For τ_{new} itself, this holds directly by Eq. (3). And for each $\tau_\ell \in lp(new)$, the preemptions of the tasks in $hp(\ell)$ are already taken into account in the slack Sl_ℓ, hence there only remains to take into account the preemptions of τ_{new} on τ_ℓ, which is precisely the goal of Eq. (2). □

This gives a sufficient but non necessary condition because the slack Sl_i is a lower bound on the slack available for τ_i. The reason the criterion is not necessary is that the preemption delays induced by τ_{new} may be overapproximated.

5. FLEXIBILITY ANALYSIS

Recall that the flexibility of a system is quantified through the function *flex*. We now show how to approximate *flex* using the above sufficient schedulability condition for S_{new}.

Definition 5. For a given period T_{new} and priority π_{new} of τ_{new}, denote

- C_S the largest WCET of τ_{new} such that S is schedulable after evolving,

- $C_{\tau_{new}}$ the largest WCET of τ_{new} such that τ_{new} is schedulable after evolving.

THEOREM 2. *S being a system and τ_{new} an evolution, let C_S^{max} be the largest WCET allowed by Eq. (2) i.e. a lower bound of C_S. Let $C_{\tau_{new}}^{max}$ be the largest WCET allowed by Eq. (3) i.e. a lower bound of $C_{\tau_{new}}$. Then*

$$C_{S_{new}}^{max} = \min\{C_S^{max}, C_{\tau_{new}}^{max}\}$$

is a lower bound of $flex_S(\pi_{new}, T_{new})$.

PROOF SKETCH OF THEOREM 2. Theorem 1 gives a guarantee on the schedulability of S_{new} depending on τ_{new} which is a sufficient non necessary condition. As a consequence, Theorem 2 based on Theorem 1 is also a sufficient and non necessary condition, i.e., a lower bound of the largest C_{new} that guarantees the schedulability of S_{new} for fixed π_{new} and T_{new} — which is given by $flex_S$. □

We can compute both C_S^{max} and $C_{\tau_{new}}^{max}$ using Theorem 3.

THEOREM 3. *Let S be a schedulable system and τ_{new} a task. Then we have C_S^{max} and $C_{\tau_{new}}^{max}$:*

$$C_S^{max} = \min_{\tau_i \in lp(new)} \left\lfloor \frac{Sl_i}{\left\lceil \frac{T_i}{T_{new}} \right\rceil} \right\rfloor \quad (4)$$

and

$$C_{\tau_{new}}^{max} = \left\lfloor T_{new} - \sum_{\tau_j \in hp(new)} \left\lceil \frac{T_{new}}{T_j} \right\rceil C_j \right\rfloor \quad (5)$$

PROOF. Eq. (2) implies that, for all task τ_i in $lp(new)$, we have $C_{new} \leq Sl_i / \lceil \frac{T_i}{T_{new}} \rceil$. Since C_{new} must be in $\mathbb{N}^+$, it follows that

$$C_{new} \leq \min_{\tau_i \in lp(new)} \left\lfloor Sl_i / \lceil \frac{T_i}{T_{new}} \rceil \right\rfloor$$

hence Eq. (4). We found the largest C_{new} by computing the largest Sl_{new}. Again, since C_{new} must be in $\mathbb{N}^+$, this leads to Eq. (5). □

Intuitively, if τ_{new} has a high priority, then it will be easy to schedule but it will have a stronger impact on S. And this impact will be greater if the period T_{new} is small. In contrast, a task τ_{new} with a low priority will have a smaller impact on the other tasks and it will be harder to schedule with a small period.

Note that the function $flex_S$ is partial. Both Conditions (4) and (5) on the flexibility of τ_{new} imply that some tuples (π_{new}, T_{new}) lead to non schedulable systems. These tuples imply that at least one task in S_{new} will have not enough time to be executed. This is the case in a system if we add τ_{new} with a high priority and a small period.

6. LIMITING TASK OF THE SYSTEM

We now focus on the fact that for a priority π_{new} and a period T_{new}, C_S^{max} is limited by one, or possibly several tasks in $lp(new)$. We call the lowest priority task among them the *limiting task*.

Definition 6. The *limiting task* of S w.r.t. a priority π_{new} and a period T_{new}, denoted τ_ℓ, is the lowest priority task τ_ℓ such that:

$$\ell = \arg\min_{\tau_i \in lp(new)} \left\lfloor \frac{Sl_i}{\left\lceil \frac{T_i}{T_{new}} \right\rceil} \right\rfloor \quad (6)$$

Note that our definition of limiting task is based on the sufficient condition in Theorem 1. As a result, it is a good indicator of where the bottlenecks for software evolution are, but it is not an exact criterion (yet).

We are particularly interested in the fact that some tasks never limit C_S^{max}, whatever the priority and period of τ_{new}.

THEOREM 4. *Consider the system S. The tasks that will never limit the system are all the tasks τ_i such that:*

$$\exists \tau_j \in S \text{ s.t. } \tau_j \in lp(i) \wedge T_j \leq T_i$$

PROOF SKETCH OF THEOREM 4. Tasks for which there exists another task of lower priority (hence with more preemptions) and lesser or equal period (hence with less or the same available time to finish) can never be the limiting task. □

A good strategy to increase the flexibility of the system is to decrease the interference of higher priority tasks on the limiting task τ_ℓ, i.e., to increase its slack. One source of complexity here is that the limiting task is not necessarily the same for different values of T_{new} and π_{new}.

We therefore identify the values of the period T_{new} upon which τ_ℓ may change. The set of these values is:

$$\mathscr{T} = \left\{ t \in \mathbb{N}^+ \mid \exists \tau_i \in S, \left\lceil \frac{T_i}{t} \right\rceil \neq \left\lceil \frac{T_i}{t-1} \right\rceil \right\} \quad (7)$$

This corresponds to the values of T_{new} for which the number of preemptions of τ_{new} on τ_i, with $i \in lp(new)$, changes. It follows that τ_ℓ remains constant in any interval $[T_{new}^i, T_{new}^j[$ where T_{new}^i and T_{new}^j are two consecutive periods in $\mathscr{T}$. It is thus sufficient to compute the limiting task of S w.r.t. each priority π_{new} and period of $T_{new} \in \mathscr{T}$ using Eq. (6).

7. CASE STUDY

Let us now show how the concepts presented in this paper are applied on an example. Consider a system:

$$S = \{(2, 10, 1), (4, 5, 1), (6, 15, 1), (8, 10, 2), (10, 30, 2)\}$$

Case 1: Priority and period of τ_{new} are known

Suppose that we want to add a task τ_{new} to S, with $\pi_{new} = 1$ and $T_{new} = 5$. By applying Theorem 3 and Definition 6 we determine that $C_S^{max} = 1$ and τ_5 is the limiting task of S. This means that τ_{new} should not have an execution time larger than 1 otherwise τ_5 may miss its deadline.

Case 2: C_S^{max} as a function of T_{new}

Let us now assume that we know nothing about τ_{new} and we want to quantify the flexibility of S.

Table 1 shows the values of C_S^{max} obtained for the possible values of π_{new} and τ_{new}. Whenever any WCET larger than 0 would make one task in S unschedulable we put $\perp$. Note that we group periods of τ_{new} for which the number of preemptions of tasks in S does not change, based on the computation of $\mathscr{T}$. In addition, if $\pi_{new} \geq 11$ then τ_{new} will have no impact on the system so we do not represent C_{new}^{max}.

Similarly, Table 2 shows $C_{\tau_{new}}^{max}$ for possible values of π_{new} and T_{new} (we leave out periods larger than 15). Finally, based on Table 1 and Table 2 we easily obtain Table 3 which shows the possible values of $C_{S_{new}}^{max}$.

The concepts introduced in this paper allow the system designer to: (1) understand when an evolution will be constrained by the need to preserve the schedulability of the system, and when it will be constrained by the need to schedule

T_{new} \ π_{new}	1	3	5	7	9
[2, 3[	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
[3, 4[	1	1	1	1	1
[4, 5[	1	1	1	1	1
[5, 6[	1	1	1	1	1
[6, 8[	2	2	2	2	2
[8, 10[	2	2	2	2	2
[10, 15[	3	3	3	3	3
[15, 30[	3	3	4	4	5
[30, $+\infty$[	3	3	4	4	11

Table 1: C_S^{max} for different values of π_{new} and T_{new}

T_{new} \ π_{new}	3	5	7	9	11
2	1	$\perp$	$\perp$	$\perp$	$\perp$
3	2	1	$\perp$	$\perp$	$\perp$
4	3	2	1	$\perp$	$\perp$
5	4	3	2	$\perp$	$\perp$
6	5	3	2	$\perp$	$\perp$
7	6	4	3	1	$\perp$
8	7	5	4	2	$\perp$
9	8	6	5	3	1
10	9	7	6	4	2
11	9	6	5	1	$\perp$
12	10	7	6	2	$\perp$
13	11	8	7	3	1
14	12	9	8	4	2
15	13	10	9	5	3

Table 2: $C_{\tau_{new}}^{max}$ for different values of π_{new} and T_{new}

the new task; (2) identify, through the concept of limiting task, which task in the system will "break" first in case of a software update.

8. CONCLUSION

In this paper we have defined the flexibility of a system as its capability to schedule a new task. We also presented an approach to quantify the flexibility of a system. More importantly, we show that it is possible under certain conditions to identify the task that will directly induce the limitations on a possible software update. If performed at design time, such a result can be used to adjust the system design by giving more slack to the limiting task. We illustrate how these results apply to a simple system.

We plan to extend our work in several main directions. First, we want to replace our sufficient conditions for by necessary and sufficient conditions so as to provide more relevant information to the system designer.

Second, we want to study further the notion of limiting task and propose a solution to compare the flexibility of different systems. Our objective is to come up with a set of guidelines for the designer to help him factor in flexibility at design time. Such guidelines must be simple to understand by someone with limited expertise in schedulability analysis, and they must take into account the fact that many design choices are fixed.

Finally, we intend to study more complex systems. We are in particular interested in systems with jitter, sporadic

T_{new} \ π_{new}	1	3	5	7	9	11
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
3	1	1	1	$\perp$	$\perp$	$\perp$
4	1	1	1	1	$\perp$	$\perp$
5	1	1	1	1	$\perp$	$\perp$
6	2	2	2	2	$\perp$	$\perp$
7	2	2	2	2	1	$\perp$
8	2	2	2	2	2	$\perp$
9	2	2	2	2	2	1
10	3	3	3	3	3	2
11	3	3	3	3	1	$\perp$
12	3	3	3	3	2	$\perp$
13	3	3	3	3	3	1
14	3	3	3	3	3	2
15	3	3	4	4	5	3

Table 3: $C_{S_{new}}^{max}$ for different values of π_{new} and T_{new}

bursts, task chains, multiple processing resources, etc. When compared to the general sensitivity analysis problem, the problem we consider has many fixed parameters. This is why we expect to be able to provide useful results even for such complex systems.

9. REFERENCES

[1] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *RTAS'06*, pages 221–230, San Jose (CA), USA, 2006. IEEE.

[2] E. Bini, M. Di Natale, and G. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *ECRTS'06*, pages 13–22. IEEE, 2006.

[3] R. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *RTSS'07*, pages 3–14. IEEE, 2007.

[4] R. I. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *RTSS'93*, pages 222–231, 1993.

[5] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Syst.*, 46(3):305–331, 2010.

[6] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[7] S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *ASIAN'97*, volume 1345 of *LNCS*, pages 72–82. Springer-Verlag, 1997.

[8] R. Racu, A. Hamann, and R. Ernst. Sensitivity analysis of complex embedded real-time systems. *Real-Time Syst.*, 39(1-3):31–72, 2008.

[9] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *RTAS'05*, pages 160–169. IEEE, 2005.

[10] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *RTAS'09*, pages 275–284. IEEE, 2009.