Using semantic properties for real time scheduling

Christian Fotsing, Annie Geniet
LISI, ENSMA
1 Av. Clement Ader BP 40109-86961 Futuroscope Chasseneuil-France
fotsingc@ensma.fr, annie.geniet@univ-poitiers.fr

Guy Vidal-Naquet
SUPELEC - 3 rue Joliot-Curie
Plateau de Moulon. BP 91192 Gif-Sur-Yvette Cedex-France
guy.Vidal-Naquet@supelec.fr

Abstract

We consider interacting tasks with conditional statements. The classical temporal model associates a single WCET to each task, and considers the union of the real-time primitives that occur in the different branches of conditional statements. Thus the model considers only a non realistic worst case, which can lead to erroneous conclusions. Furthermore, semantic is never considered. We extend the task model with conditional statements, and the notion of schedule is replaced by the notion of scheduling tree. We then present some examples which illustrate the benefit of taking semantic into account. This lays the basis for a complete model based scheduling analysis which explicitely takes conditional statements and semantic into account.

1 Introduction

We consider real time applications dedicated to process control. They are generally modeled as a set of periodic, possibly interacting tasks. For safety reasons, tasks have to respect firm deadlines. One of the main challenges for system designer is to ensure that all deadlines are met. This is the concern of scheduling. Scheduling theory relies on the temporal modeling of tasks. Classically, a task $\tau_i <r_i, C_i, D_i, P_i>$ is modeled by four temporal parameters [LL73]: its first release time r_i, its worst case execution time (WCET) C_i, its relative deadline D_i which is the maximum acceptable delay between release and completion of any instances of the task, and its period P_i. A task consists in an infinite set of instances (or jobs) released at times $r_i + k \times P_i$, where k is a natural integer. Tasks may also communicate or share critical resources. They may thus use real-time primitives (send or receive messages, lock and unlock resources). In classical approaches, one single WCET is associated to each task. If a task contains conditional statements, its WCET corresponds to the longest execution path of the task. It can be obtained either by simulation or by static syntaxical analysis of the task code [CPRS03]. All real-time primitives are also considered to occur, even if they are in different branches of conditional statements. Thus scheduling analysis is performed from a non realistic model. This can lead to erroneous conclusions. Our aim is to illustrate that the classical modeling of tasks must be refined. Moreover, we illustrate through some examples that semantic must also be considered, otherwise, again erroneous conclusions could be drawn.

The paper is organised as follows. First we present our assumptions. Then, we present our task model and the scheduling model: since we consider conditional statements, the notion of schedule is no longer suited, and must be replaced by the notion of scheduling tree which describes the different possible effective behaviors of the application. We present the limitations of the classical approach through some examples and we conclude and present our future investigations.

2 Context and classical temporal model

2.1 General context

In this paper, we consider applications composed of interacting periodic tasks. They run on preemptive uniprocessor systems. We consider pre-runtime (off-line) scheduling. This means that we want to obtain, from the modeling of the tasks, a global execution model of the application. This execution model (a schedule in classical scheduling analysis) can then be implemented within the system.

We assume that tasks consist in [CGGC96, GCG00]

- blocks composed of imperative statements. b_i(j) denotes the i^{th} block of the tasks, with execution time equal to j,
- conditional statements: IF condition THEN ... ELSE .... End if,

- Lock(R) and Unlock(R) where R is a resource of the system,

- Send(M) and Receive(M) where M is a message linked to a mailbox.

We assume that real-time primitive processing times are included in the processing times of neighbour blocks. Furthermore, execution times of condition evaluation are included in the execution times of the statements of the branches THEN and ELSE. E.g., for task T_1 on Figure 1, the computing time required by the evaluation of the condition is included in the execution times of blocks b_2 and b_3. Then we model tasks by **execution trees** (Figure 1).

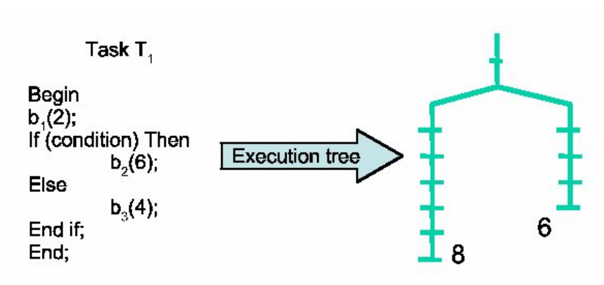

Figure 1. Modeling of a task with conditional statement by an execution tree

2.2 From execution trees to classical temporal model of tasks

The classical model, called here **sequential model**, is a sequence of blocks and of real-time primitives.The WCET of each block is computed, and real-time primitives are temporally located. A real-time primitive occurs at time t if there exists a branch of the task in which this primitive occurs at time t [Gro99, Bab96, Nie91]. We present succintly the computation of this sequential model. WCET are computed using classical methods that can be found in the literature [CPRS03, Pua05].
If we were to follow the usual model, the task of Figure 1 would be represented by (see Figure 2):

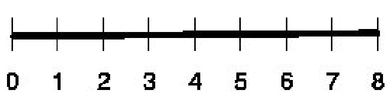

Figure 2. Usual model for the task of Figure 1

2.2.1 Task without conditional statements

The WCET is an upper bound of the execution time required by the task when it runs alone on the processor. The sequential model is here the effective description of the behavior of the task.

2.2.2 Task with conditional statements

Each branch of the execution tree is associated to its own WCET. The global WCET is then the maximal value of these WCET. Figure 3 illustrates the WCET computation.

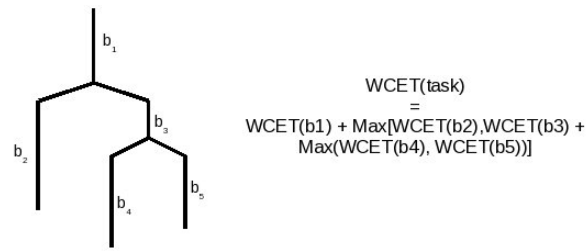

Figure 3. Evaluation of WCET for a task with conditional statements

2.2.3 Task with conditional statements and resource utilization

We consider a task with a conditional statement such that:

- The THEN branch has a WCET equal to C_1, and uses resource R_1 between execution times t_1 and t_2

- The ELSE branch has a WCET equal to C_2, and uses resource R_2 between execution times t'_1 and t'_2

The sequential model has a WCET equal to max(C_1, C_2), and the task is considered to use R_1 between t_1 and t_2 and R_2 between t'_1 and t'_2. Figure 4 illustrates this case.

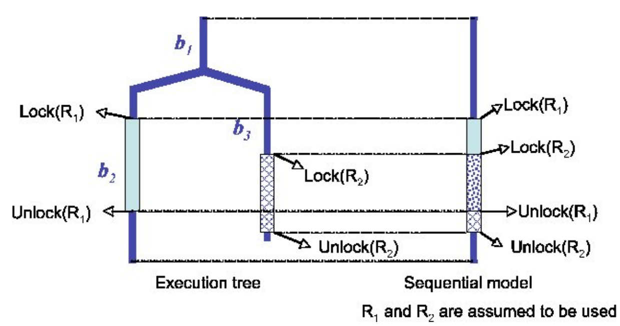

Figure 4. Conditional statement and utilization of resources

2.2.4 Task with conditional statement and communication primitives

We consider a task with a conditional statement such that:

- The THEN branch has a WCET equal to C_1, and sends a message M_1 at execution time t

- The ELSE branch has a WCET equal to C_2, and receives a message M_2 at execution time t'

The sequential model has a WCET equal to max(C_1, C_2), and the task is considered to send message M_1 at time t and to receive message M_2 at time t'. Figure 5 illustrates this case.

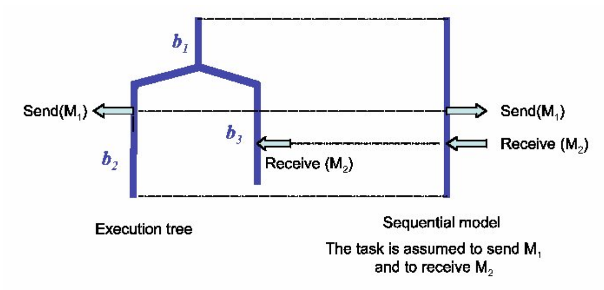

Figure 5. Conditional statement and communication

3 Scheduling

Classical scheduling approaches compute schedules from the WCET. Schedules here completely hide conditional behaviours of tasks. Now, if one wants to depicte the actual behaviour of the system, he has to consider the tree modeling of tasks, and he cannot thus produce one single schedule. He must describe every possible paths followed by the application. Thus, we must introduce a new execution model: **the scheduling tree**, which lets explicitely appear each conditional node. Consider the application of Figure 6, composed of two synchronous tasks. The behavior of the application must be modeled on the time interval [0, 16] where 16 is the hyperperiod [1], and it is then iterated. A scheduling tree is given in Figure 6. The application processes T_2 for one time unit, then the conditional statement of T_2 is scheduled. There are thus two subtrees, which correspond to the further behaviour of the application if the "then" respectively "else" branch of T_2 is chosen. The application can here have 8 different behaviours on [0, 16]. A scheduling tree is then **valid** if all deadlines are met whatever the different choices made in conditional statements. The scheduling tree of Figure 6 is valid since the first occurrence of T_2 completes either at time 3 or at time 4, and the second occurrence at times 11 or 12, whatever the followed branch. And task T_1 completes between times 12 and 16. So all deadlines are met. A real-time system is then said to be **globally (or strongly) feasible** if there exists at least one valid scheduling tree. We are now interested in the feasibility analysis. We consider pre run-time (off-line) analysis for uniprocessor systems. When sequential models are used, we have a necessary condition for a system to be feasible:

[1]the lcm of all periods

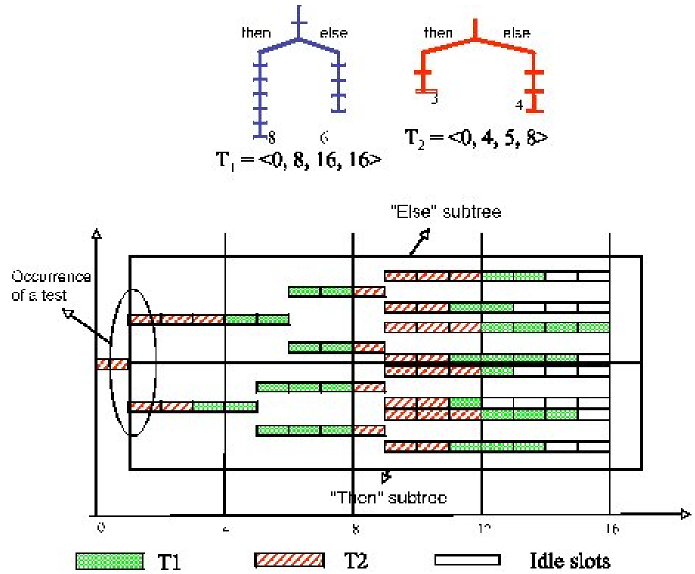

Figure 6. Scheduling tree

Property 1 *[But97] If a system of n sequential tasks (τ_1,τ_2,..., τ_n) is feasible then its utilization factor [2] is at most equal to 1*

Our aim is now to present some examples which show that considering the sequential models of tasks can lead to erroneous conclusions.

4 Classical analysis versus our analysis

4.1 Utilization of several resources

We consider a system S_1 composed of two tasks T_1 <0, 16, 32, 32> and T_2 <0, 2, 4, 4>. They share two resources R_1 and R_2.

- T_1 uses R_1 between execution time units 3 and 6
- T_1 uses R_2 between execution time units 4 and 8
- T_2 uses R_1 during its first execution time unit and then R_2 during the second.

We first use here the sequential model, and conclude that the system is not feasible, because task T_2 always miss its third deadline. Indeed, between times 0 and 8, 2 instances of T_2 are processed, for 4 time units. T_1 is thus processed for 4 time units within the 8 first time units. Thus at time 8, T_2 is released, but R_1 is locked by T_1, and it cannot be unlocked before time 10. But then R_2 will be locked for still 2 more time units, thus won't be available before time 12. Thus T_2 will miss its deadline (see Figure 7). We now refine our analysis, and consider the pseudo-code of tasks. We assume that the sequential model of T_1 has been deduced from the code given in Figure 8. We can observe that resources R_1 and R_2 are never both used by an instance of T_1. The former analysis thus doen't hold.

[2]the utilization factor U is defined as $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$

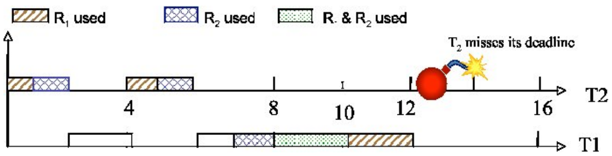

Figure 7. System S_1 is not feasible

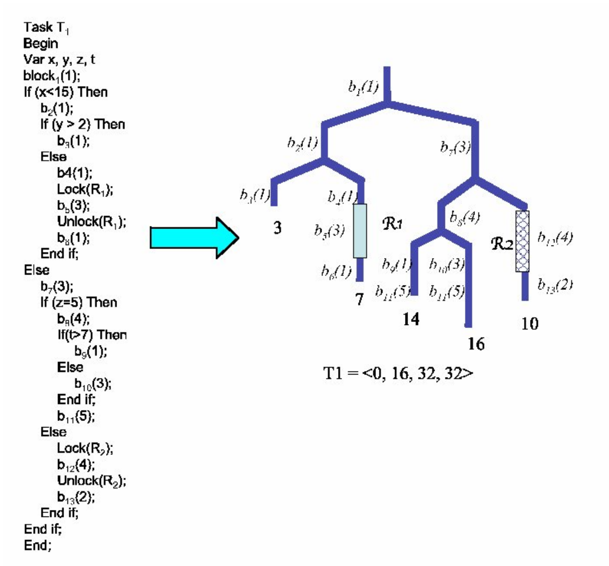

Figure 8. Code and execution tree of task T_1

In fact, either T_1 uses R_1 and then T_2 can be processed at time 10 even if R_2 is still locked, or T_1 uses R_2, then T_2 can be processed at time 8 before T_1 locks R_2. We present in Figure 9 a valid scheduling tree of the application. The system is thus feasible, despite the negative conclusion when using the classical sequential model.

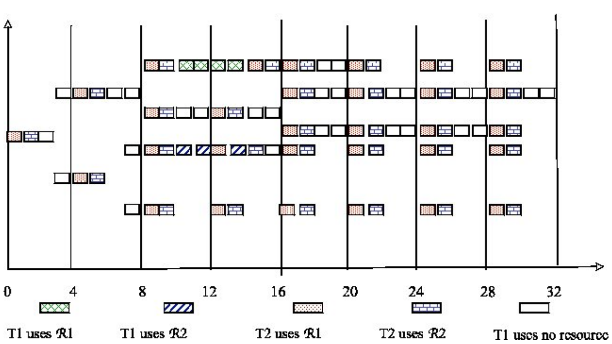

Figure 9. A scheduling tree of S_1

4.2 Considering semantic

We consider a system S_2 composed of two communicating tasks T_1 <0, 5, 9, 9> and T_2 <0, 5, 9, 9>. We assume that:

- T_1 receives a message M to T_2 after its 1^{st} execution time unit
- T_1 sends a message M' from T_2 after its 4^{th} execution time unit
- T_2 receives a message M' from T_1 after its 2^{nd} execution time unit
- T_2 sends a message M to T_1 after its 3^{rd} execution time unit

This system is not feasible for two reasons:

- it doesn't respect the necessary condition (property 1): $U = \frac{5}{9} + \frac{5}{9} = \frac{10}{9} > 1$,
- deadlock cannot be avoided.

Now, let us assume that this sequential model comes from the pseudo code described in Figure 10. Furthermore,

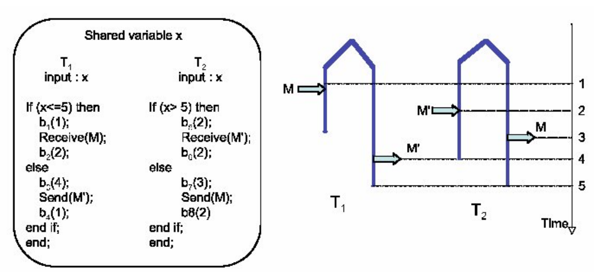

Figure 10. System S_2

since x is an input parameter of both tasks, and, since they are always released at the same time, they always consider the same value for x. We can notice that, according to the semantic of tests, either T_2 executes its THEN branch and T_2 its ELSE branch, or conversely. Thus either message M is send and received or message M'. The deadlock thus doesn't take place. Moreover, the case corresponding to U greater than 1 is impossible, because of the semantic. And the system is feasible in practise. Figure 11 presents a valid scheduling tree. Here again, only the semantically correct branches have been kept.

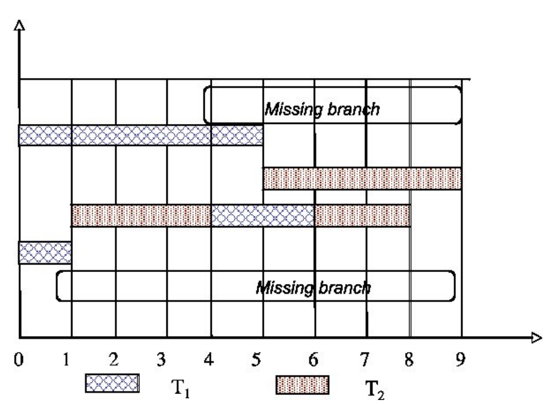

Figure 11. A scheduling tree of S_2 - Missing branches are semantically inconsistent

5 Conclusion and future work

We have laid the basis for future works. We intend to develop a complete model based methodology for scheduling analysis. For that purpose, we will extend the Petri nets based methodology proposed in [CGGC96, GCG00]. The former examples prove that the classical sequential model of tasks as well as the notion of schedule must be refined. They also prove that it is mandatory to take semantic into account in the feasibility analysis. The general frame of our future investigations is given in Figure 12.

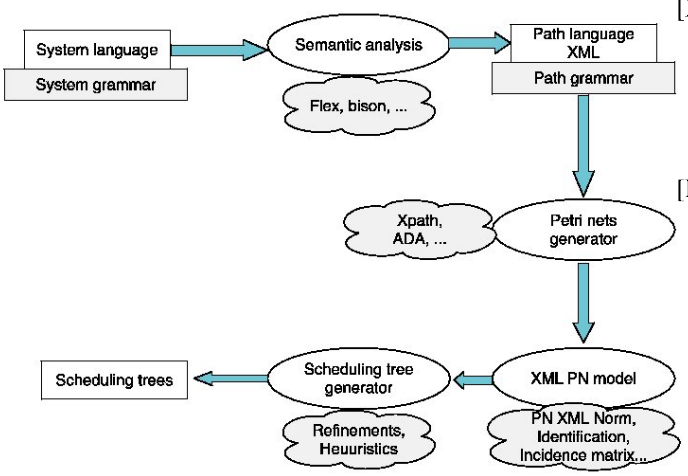

Figure 12. Our methodology

References

[Bab96] J.P. Babau. *Etude du comportement temporel des applications temps réel à contraintes strictes basée sur une analyse d'ordonnançabilité*. PhD thesis, University of Poitiers, France, 1996.

[But97] G.C. Buttazo. *Hard real Time computing systems: predictable scheduling algorithms and applications*. Kluwer Academic, 1997.

[CGGC96] A. Choquet-Geniet, D. Geniet, and F. Cottet. Exhaustive computation of the scheduled task execution sequences of a real-time application. In *Proc. of FTRTFT'96*, October 1996.

[CPRS03] A. Colin, I. Puaut, C. Rochange, and P. Sainrat. Calcul de majorants des pires temps d'exécution : état de l'art. *Techniques et Sciences Informatiques (TSI)*, 22(5):651–677, 2003.

[GCG00] E. Grolleau and A. Choquet-Geniet. Off line computation of real time schedules by means of petri nets. In R. Boel and G. Stremersch, editors, *Discrete events systems*, pages 309–316. Kluwer Academic Publishers, 2000.

[Gro99] E. Grolleau. *Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur*. PhD thesis, ENSMA: Ecole Nationale Supérieure de Mécanique et d'Aérotechnique Ecole Doctorale Sciences Pour l'Ingénieur, Novembre 1999.

[LL73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in real-time environnement. *Journal of the ACM*, 20(1), 1973. pages 46, 61.

[Nie91] E. Niehaus. Program representation and translation for predictable real-time systems. In *Proceedings og the 12th IEEE Real-Time Systems Symposium*, pages 53–63, San Antonio, Texas, 1991.

[Pua05] I. Puaut. Obtention de pires temps d'exécution (wcet, worst case execution times). Technical report, Université de RENNES 1, IRISA, Septembre 2005.