Accurate Strategy for Mixed Criticality Scheduling

Yasmina Abdeddaïm

Accurate Strategy for Mixed Criticality Scheduling

Yasmina Abdeddaïm

LIGM, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454 Marne-la-Vallée
yasmina.abdeddaim@esiee.fr

Abstract. In mixed criticality systems, functionalities with different criticalities share the same execution platform. The goal of a mixed criticality scheduling algorithm is to ensure a safe execution of the highest criticality tasks while using efficiently the execution platform. Classical real-time scheduling algorithms for mixed criticality systems propose to stop the execution of low criticality tasks to ensure that the highest criticality tasks meet their deadlines. In this work, we propose the Accurate Fault Mode (AFM) strategy for the scheduling problem of mixed criticality real-time systems. The advantages of this strategy are that firstly it can reduce the number of stopped low criticality tasks and secondly, that it allows the designer to define the low criticality tasks to be stopped depending on the criticality configuration of the system. Using model checking for timed game automata, we propose an exact feasibility test and exact schedulability tests for fixed priority and earliest deadline first scheduling algorithms for AFM strategy.

Keywords: Mixed criticality, Real-time scheduling, Timed Game Automata

1 Introduction

A mixed criticality system is a system that incorporates the functionalities of different criticalities on the same platform. The highest criticality functionalities are usually related to safety critical applications and need to fulfill strict certification requirements, while lower criticality functionalities are non-safety critical functionalities with less stringent certification requirements. To avoid interference between applications of different criticalities, traditional methods use temporal or spatial isolation between applications of different criticalities [17], however, this isolation may lead to an inefficient use of the execution platform.

With the increasing complexity of embedded systems in the industry of critical systems, standards in automotive and avionic, are now mentioning that "mixed criticality" must be supported on their platform [10].

The first work dealing with real-time scheduling for mixed criticality systems is that of Vestal [19]. In Vestal's model, a real-time task has several worst-case execution time (WCET) estimates, one per possible "mode of execution" of the system, called the criticality mode of the system. For example, a WCET in the case of a "regular mode" and a WCET in the case of a "fault mode" where the "fault mode" is more critical than a "regular mode" and the higher the criticality mode, the larger the WCET estimate. The intuition is that larger execution time

values are less likely to occur, but if they occur, they may be indicative of an erroneous event and the system moves in this case to a high criticality mode.

A system designed to be executed in a high criticality mode is safe but does not use the platform efficiently as high criticality WCETs are pessimistic, however, considering lowest criticality WCETs is unsafe even if the probability of occurrence of high criticality WCETs is small. The challenging issue is to ensure that the system is safe while using the execution platform efficiently.

Since Vestal's model, a standard approach for the scheduling problem of mixed criticality real-time systems has emerged. In this approach, the system starts its execution in the lowest criticality mode and moves to the highest criticality mode if the execution time of a job exceeds the WCET of the low criticality mode. In the lowest mode, all the tasks have to respect their timing constraints and when the criticality mode of the system increases, lowest criticality tasks are no more activated so that higher criticality tasks fulfill their requirements.

As stated in the frequently updated review paper [10], criticisms of the standard approach are addressed concerning the following hypothesis:

1. The criticality mode of the system can only increase: it should be possible for the system to recover and move from a high to a low criticality mode.
2. When the system is in a high criticality mode, the execution time of all the high tasks is supposed to be equal to the WCET of the high criticality mode: this hypothesis is too abusive, a high criticality mode, could be the consequence of the occurrence of an error in only one critical task.
3. When the criticality mode of the system increases, less critical tasks are definitely no more activated: even if lower criticality functionalities are non-safety critical functionalities they are relevant for the good functioning of the system [9]. Abandoning all low criticality tasks may have an impact on the execution of the system and degrade the quality of service.

In this paper, we present a dual criticality model where:

1. The criticality mode of the system increases if an erroneous behavior occurs (a job exceeds its low criticality WCET estimates), however, if the erroneous behavior disappears the criticality of the system decreases.
2. We introduce a more accurate measure of the criticality mode of the system called the criticality configuration of the system. The criticality configuration gives the set of tasks that are exhibiting an erroneous behavior.
3. The designer can specify the subset of lowest criticality tasks to stop when the system is in high criticality mode depending on the subset of high criticality tasks exceeding their low criticality WCET.

For this model, we propose an exact feasibility test and exact schedulability tests for fixed priority and earliest deadline first scheduling algorithms. These tests are derived from CTL model checking for timed game automata.

Section 2 reviews related research. Section 3 introduces the model and the mixed criticality scheduling problem. Section 4 presents the Accurate Fault Mode (AFM) strategy. Section 5 presents the feasibility and schedulability tests for

the AFM strategy. Section 6 is dedicated to an example used to illustrate the advantages of our approach. We conclude and give future research directions in Section 7.

2 Related work

For a complete review of mixed criticality real-time scheduling see [10]. We focus in this section on uniprocessor scheduling addressing one of the three criticisms cited in the introduction. Since Vestal's model, many scheduling policies have been proposed to handle the problem of stopping all low criticality tasks when the criticality of the system is high. In [12], the AMC Weakly Hard (AMC-WH) policy is proposed, in this policy, only a number of consecutive jobs of low critically tasks is allowed to be stopped when the system is in high criticality mode, the authors proposed a sufficient schedulability test for this policy. In [11], the authors introduce the notion of importance. When the system is in high criticality mode, low criticality tasks are stopped in the inverse order of their importance. In these two works, the set of tasks that are no more activated do not depend on the configuration of the system as proposed in our work.

In [13], the authors introduce the notion of interference constraint graph, where the designer can specify the subset of tasks to be stopped when a particular high criticality task exceeds a certain budget of time. This approach has some similarities with our work, however, in this approach, when a task is stopped, it is no more activated, and the strategy is defined for a task and not for a particular configuration.

Concerning the possibility of a system to recover and move from a high to a low criticality mode, a simple protocol presented in [18] is to switch to a low criticality mode at the first instant where no job is active. A more complex protocol, called the bailout protocol, is presented in [6, 7]. In this protocol, when a high criticality task exceeds its WCET at low criticality level, it is assigned a bailout fund, this bailout is funded by the execution times of low tasks or by the execution times of tasks that terminate before their estimated WCET.

None of the mentioned work proposes a model to handle the three criticisms cited in Section 1 at the same time and does provide exact feasibility and schedulability tests.

3 The problem statement

We model the mixed criticality real-time system as a set of n sporadic real-time tasks $\Gamma = \{\tau_i : i = 1 \dots n\}$ scheduled on a single processor. We restrict this work to dual criticality systems, i.e. systems with only two possible criticalities LO and HI, LO is the lowest criticality and HI is the highest criticality. Every task is assigned a criticality level defined by the system designer, nLO is the number of LO criticality tasks, and nHI is the number of HI criticality tasks. Without loss of generality, we suppose that the set Γ is sorted in a decreasing order of criticality. We denote Γ^{LO} the set of low criticality tasks, and Γ^{HI} the set of HI criticality tasks. Each task $\tau_i \in \Gamma$ generates an infinite number of jobs $\tau_{i,k}$.

A task $\tau_i \in \Gamma$ is defined as a tuple $(L_i, pr_i, T_i, D_i, C_i)$ where:

- $L_i \in \{LO, HI\}$ is the criticality of the task.
- $pr_i \in \{1 \dots n\}$ is the priority of the task with 1 as the highest priority and n the lowest priority. We use $lp(i)$ to be the indexes of tasks of lower priority than τ_i.
- T_i is the minimum time separation between jobs releases. Two tasks can have the same priority.
- $D_i \leq T_i$ is its relative deadline, i.e. the maximal time length between the arrival time and the completion time of a job.
- C_i is a tuple $(C_i(LO), C_i(HI))$, where $C_i(l) \in \mathbb{N}$ is the WCET budget of task τ_i at criticality level $l \in \{LO, HI\}$ s.t. if $L_i = LO$, $C_i(HI) = C_i(LO)$ and if $L_i = HI$, $C_i(LO) \leq C_i(HI)$.

For every task τ_i, the $C_i(l)$ WCET budget is a constant interval of time allocated to the execution of every job of the task when the criticality of the system is $l \in \{LO, HI\}$. We use the notion of run-time monitoring of execution times introduced in [5]. If a job of a task does not signal its completion after the execution of its allocated budget at its own criticality it is stopped, as a consequence, a LO criticality task cannot exceed its LO WCET budget.

A job is *active* at instant t if it is triggered by a task at time $t' \leq t$ and the job has not yet notified its completion at instant t. We note S_t the set of active jobs at time t. The response time $R_{i,k}$ of a job $\tau_{i,k}$ is the duration between the activation date of the job and its completion date. The current response time of an active job is the duration between the activation date of the job and the current time. A job $\tau_{i,k}$ respects its deadline if and only if $R_{i,k} \leq D_i$ and a task τ_i respects its deadline if and only if all the jobs activated by the task respect their deadlines.

An active job of a task τ_i is *critical*, if and only if τ_i is a high criticality task and the job does not notify its completion after the execution of $C_i(LO)$ time unit. An execution scenario of a task set in an interval of time $[t_1, t_2]$, gives for every instant t, $t_1 \leq t \leq t_2$ $Exec(t)$ the sets of active jobs, preempted and executed jobs at time t and for every active task its current response time and the duration since its activation. We note $\mathbb{E}$ the set of possible execution scenarios.

The notion of criticality mode of the system is used to characterize a particular execution scenario of a task set. The criticality mode of the system is HI at instant t if there exists an active critical job, otherwise the criticality mode of the system is LO. A formal definition is given in Definition 1.

Definition 1 *Criticality mode at time t. The criticality mode of the system at time t is a function $Cr(t) : \mathbb{R}^+ \to \{LO, HI\}$ with*

$$Cr(t) = \begin{cases} LO \text{ if for all } \tau_{i,k} \in S_t,\ \tau_{i,k} \text{ is not a critical job} \\ HI \text{ if there exists } \tau_{i,k} \in S_t \text{ a critical job at time } t \end{cases}$$

Given a task set, a scheduling algorithm gives at each instant the job to be executed among the set of active jobs for every execution scenario.

Definition 2 *Scheduling algorithm. A scheduling algorithm is a function Sched* : $(\mathbb{E}, \mathbb{R}^+) \rightarrow \{\tau_{i,k} \mid i = 1 \ldots n, k = 1 \ldots \infty\} \bigcup \{\perp\}$ *with*

- *if* $Sched(Exec(t), t) = \tau_{i,k}$ *then execute the active job* $\tau_{i,k} \in S_t$ *at time* t
- *if* $Sched(Exec(t), t) = \perp$ *then the processor is idle at time* t.

A scheduling algorithm is preemptive, if the execution of a job of a task can be preempted by a job of an other task. A scheduling algorithm is a job level fixed priority algorithm if a fixed priority is assigned to every active job of a task and at each time the job with the highest priority among the set of active jobs is executed. A scheduling algorithm is a fixed priority algorithm if priorities are assigned to tasks, i.e. all the jobs of a task have the same priority.

In definition 3, we define the mixed criticality scheduling problem. In this problem, when the criticality of the system is LO, no job is allowed to miss its deadline, but when the criticality of the system is HI, only jobs of HI criticality must respect their deadlines. The idea is to relax the timing constraints of low criticality functionalities of the system to focus on the good functioning of the high criticality functionalities.

Definition 3 *Mixed Criticality Schedulability. A task set* Γ *is mixed criticality schedulable according to a scheduling algorithm iff (1) all the tasks respect their deadlines when the criticality mode of the system is LO and (2) all the HI criticality tasks respect their deadlines when the criticality mode of the system is HI.*

A task set Γ is *mixed criticality feasible* if and only if there exists a scheduling algorithm such that Γ is schedulable according to this scheduling algorithm.

Note that the schedulability problem of Definition 3 is equivalent to the standard scheduling problem of mixed criticality real-time systems. The difference is in the way that we compute the criticality mode of the system, see Definition 1, as we take into account the fact that a system can recover and the criticality mode of the system can return to low if no job exceeds its WCET at LO criticality.

4 Accurate Fault Mode Strategy

Classical scheduling strategies for mixed criticality scheduling propose to stop the activation of low criticality tasks when the criticality mode of the system is HI as low criticality tasks are not constraint to respect their deadlines in a high criticality mode, see Definition 3. As we mentioned in the introduction, stopping the activation of all low criticality tasks has been criticized.

In this section, we introduce the "Accurate Fault Mode" (AFM) strategy. In this strategy, the designer can specify, by setting up a "fault mode policy", the set of LO criticality tasks that are no more activated when the system is in HI criticality mode. The idea is that even if they are not critical, some LO criticality tasks may have an impact on the quality of service of the system or have to be executed during the HI criticality period to ensure the degraded

mode. This subset of LO criticality tasks can change depending on the execution configuration of the task set. For example, if only one job is critical, the designer may decide to ensure a certain quality of service, otherwise, if more jobs are critical the quality of service is only ensured at its minimum.

To evaluate the criticality mode of the system in a more accurate way, we introduce the notion of criticality configuration. If the criticality mode of the system is HI, the criticality configuration returns the set of tasks that are exhibiting an erroneous behavior.

Definition 4 *Criticality configuration. The criticality configuration of the system* $\overrightarrow{HI} = (HI(1), \ldots, HI(nHI))$ *at time* t *is a boolean vector of size* nHI *with*

$$HI(i) = \begin{cases} 1 \text{ if there exits } \tau_{i,k} \text{ an active critical job of } \tau_i \text{ at } t \\ 0 \text{ if there is no active critical job of } \tau_i \text{ at } t \end{cases}$$

We note $\mathbb{H}$ *the set of possible criticality configurations.*

To define the policy for dealing with LO criticality tasks, the designer gives for every criticality configuration, the set of LO criticality tasks for which the activation can be stopped. The fault mode policy is formalized in Definition 5.

Definition 5 *Fault mode policy. Given a task set* Γ*, a fault mode policy of a criticality configuration* $\overrightarrow{HI} \in \mathbb{H}$ *is a function* $policy^{\overrightarrow{HI}}$ *from* Γ^{LO} *to* $\{0, 1\}$ *with,*

$$policy^{\overrightarrow{HI}}(\tau_i) = \begin{cases} 1 \text{ denotes that jobs of } \tau_i \text{ are not activated when} \\ \quad \text{the criticality configuration is } \overrightarrow{HI} \\ 0 \text{ denotes that jobs of } \tau_i \text{ are activated when the} \\ \quad \text{criticality configuration is } \overrightarrow{HI} \end{cases}$$

A fault mode policy $policy_i$ *is defined by* $policy_i = \bigcup_{\overrightarrow{HI} \in \mathbb{H}} policy^{\overrightarrow{HI}}$.

In Definition 6, we define the Accurate Fault Mode (AFM) strategy for mixed criticality scheduling.

Definition 6 *Accurate Fault Mode (AFM) strategy (w.r.t. a fault mode policy). Given a task set* Γ *and a fault mode policy* $policy_i$*, a scheduling algorithm Sched respects an Accurate Fault Mode (AFM) strategy if and only if when the criticality of the system is* HI *the schedule computed using Sched respects the fault mode policy* $policy_i$*.*

In Definition 7, we define the AFM schedulability problem for mixed criticality scheduling. In this problem, a subset of LO criticality tasks may have to respect their deadlines even in a HI criticality mode of the system. This subset of tasks is defined using the fault mode policy given by the system designer.

Definition 7 *AFM Schedulability. A task set* Γ *is AFM schedulable according to the scheduling algorithm Sched if and only if (1) all the tasks respect their*

deadlines when the criticality mode of the system is LO and (2) all the active jobs respect their deadlines when the criticality mode of the system is HI and the fault mode policy of the AFM strategy is applied.

A task set Γ is AFM feasible if and only if there exists an AFM strategy and a scheduling algorithm such that Γ is AFM schedulable.

5 Feasibility and schedulability analysis

In this section, we present exact feasibility and schedulability tests for the AFM strategy. These tests are based on CTL model checking for timed game automata.

5.1 A timed game model

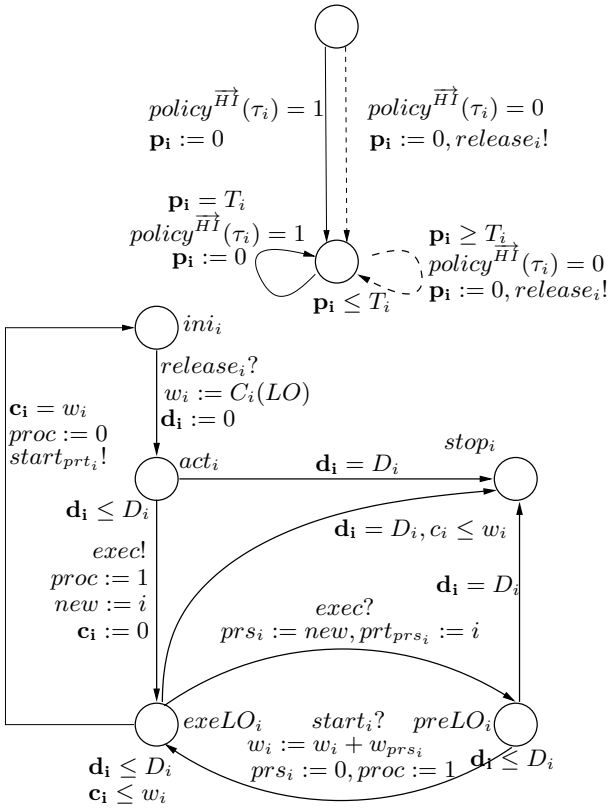

Fig. 1. $TaskLO_i$ and $PeriodLO_i$ of a LO criticality task τ_i

We model a task set Γ as a network of timed game automata T_Γ as presented in this section. A complete model using the tool Uppaal-Tiga [8] is available in [1].

A Timed automaton (TA) [3] is a model extending the automaton model with a set of real variables, called clocks, evolving synchronously with time. Transitions of a time automaton can be labelled by a clock constraint, clocks can be reset in transitions and states can be constrained by a staying condition.

A network of timed automata is the parallel composition of a set of timed automata, the parallel composition uses an interleaving semantic. Synchronous communication between timed automata is done using input actions denoted $a?$ and output actions denoted $a!$, this notation is used in the tool Uppaal [15]. A configuration of a network of n timed automata T_i is a pair $(q, \mathbf{v})$ where $q = (q_1, \ldots, q_n)$ is a vector of states, with q_i is a state of T_i, and $\mathbf{v}$ a vector of clock valuations.

A Timed game automaton (TGA) [16] is an extension of the time automaton model where the set of transitions is split into controllable (Δ_c) and uncontrollable (Δ_u) transitions. In Figures 1 and 2, dashed lines represent uncontrollable transitions. This model defines the rules of a game between a controller (controllable transitions) and the environment (uncontrollable transitions).

Given a timed game automaton T and a logic formula ϕ, if T satisfies ϕ then there exists a strategy f, defining for every possible configuration, the controllable transition to execute, s.t. T supervised by f always satisfies ϕ whatever are the uncontrollable transitions chosen by the environment. A strategy is formally a partial mapping from the set of runs of the TGA to the set $\Delta_c \cup \{\lambda\}$ s.t. for a finite run ξ:

- if $f(\xi) = e \in \Delta_c$ then execute transition e from the last configuration of ξ,
- if $f(\xi) = \lambda$ then wait in the last configuration of ξ.

We model a LO criticality task $\tau_i \in \Gamma^{LO}$ using a timed game automaton $TaskLO_i$ with a set $\mathcal{Q} = \{ini_i, act_i, exeLO_i, preLO_i, stop_i\}$ of states and two clocks c_i and d_i. This automaton is synchronized with a timed game automaton $PeriodLO_i$ using the action $release_i$. If the fault mode policy defined by the designer states that the activation of task τ_i is not stopped for the current criticality configuration ($\overrightarrow{policy^{HI}}(\tau_i) = 0$), the action $release_i$ is launched by the automaton $PeriodLO_i$. In this case, the automaton $TaskLO_i$ is synchronized with the uncontrollable transition of automaton $PeriodLO_i$. This transition is an uncontrollable transition as the task set Γ is sporadic i.e. a job is triggered at the earliest every T_i time unit, but we don't control the time by which the job is triggered. Otherwise, if the fault mode policy states that jobs of task τ_i are no more activated ($\overrightarrow{policy^{HI}}(\tau_i) = 1$), the action $release_i$ is not launched.

The automaton $TaskLO_i$ starts its execution at state ini_i, when an action $release_i$ is captured, the automaton moves to state act_i and the clock d_i is reset to zero, when the clock d_i reaches the deadline D_i the automaton moves to state $stop_i$.When the task starts its execution, the automaton moves to state $exeLO_i$, the global variable $proc$ is reset to one indicating that the processor is not idle and the clock c_i is reset to zero. The clock c_i is used to measure w_i the response time of task τ_i. The response time w_i of a task is set initially to $C_i(LO)$.

To be able to handle preemption using timed automata, we restrict ourselves to job level fixed priority scheduling algorithms. In a job level fixed priority

scheduling algorithm, the priority does not change during time between two jobs $\tau_{i,k}$ and $\tau_{j,r}$ of two different tasks.This restriction does not limit much the generality of our work as most of the commonly known scheduling algorithms, as fixed priority (FP) and earliest deadline first (EDF), respect this restriction. As a consequence, in our model, a task can be preempted only if a new job is executed, this is done by using a global action *exec*? synchronizing every preemption with the execution of a new job. When a task is preempted, the automaton moves to state $preLO_i$, the variable prs_i records the identifier of the preempting task and prt_i the identifier of the task preempted by τ_i. When the preempting task τ_{prs_i} completes, the response time w_i of the task τ_i is augmented by the response time of the preempting task τ_{prs_i}. This method has been used in [2] to model preemptions.

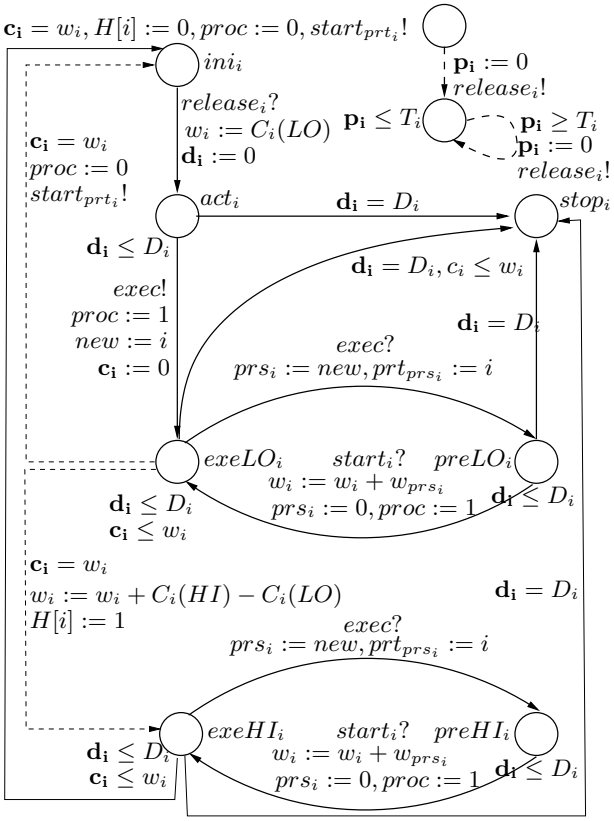

Fig. 2. $TaskHI_i$ and $PeriodHI_i$ of a HI criticality task τ_i

We model a HI criticality task $\tau_i \in \Gamma^{HI}$ using a timed game automaton $TaskHI_i$, this automaton is synchronized with a timed game automaton $PeriodHI_i$. In addition to states of automaton $TaskLO_i$, automaton $TaskHI_i$

has two specifics states $exeHI_i$ and $preHI_i$ used to model the behavior of a critical job. The transitions from $exeLO_i$ to ini_i and from $exeLO_i$ to $exeHI_i$ are uncontrollable as we don't know in advance if the job will terminate after the execution of $C_i(LO)$ time unit or exhibits an erroneous behavior.

If the job is critical, the variable w_i is augmented by $C_i(HI) - C_i(LO)$. In the transition from $exeLO_i$ to $exeH_i$, the criticality configuration $\overrightarrow{HI}$ is updated by adding the task τ_i to the set of tasks with a critical job ($HI[i] = 1$). When the critical job terminates its execution, transition from $exeHI_i$ to ini_i, the criticality configuration $\overrightarrow{HI}$ is updated by removing the task τ_i from the set of tasks with a critical job ($HI[i] = 0$).

Given a task set Γ, the network of TGA T_Γ is the parallel composition of $TaskLO_1, \ldots, TaskLO_{nLO}$ and $TaskHI_1, \ldots, TaskHI_{nHI}$ and $PeriodLO_i \ldots PeriodLO_{nLO}$ and $PeriodHI_i \ldots PeriodLO_{nHI}$. The TGA T_Γ is augmented with a clock t, this clock is never reset and is used to measure the total elapsed time.

We say that a configuration $(q, \mathbf{v}, t)$ is equivalent to an execution scenario $Exec(t)$ iff if a task is active, or preempted, or executed in q then it is also active or preempted, or executed in $Exec(t)$ and the current response times and duration since the activation of every active task are identical in $Exec(t)$ and $(q, \mathbf{v}, t)$.

5.2 Exact Feasibility and schedulability tests

We use CTL [14] model checking for time game automata to build feasibility and schedulability tests for the AFM strategy. The task set Γ is modeled using a network of timed game automata as presented in Section 5.1.

Theorem 1 (Exact AFM Feasibility test) *The task set Γ is AFM feasible (w.r.t. a fault mode policy) according to a job level fixed priority algorithm iff the network of timed game automata T_Γ modeling Γ satisfies the CTL Formula 1.*

$$AG\neg(\bigvee_{\tau_i \in \Gamma} stop_i) \tag{1}$$

Proof. An automaton T satisfies the formula $AG\phi$ iff there exists a wining strategy s.t. whatever the execution of uncontrollable transitions, there exists an execution where all the states satisfy ϕ.

Suppose that Formula 1 is satisfied, then, whatever is the execution of uncontrollable transitions in T_Γ, there exists an execution where $stop_i$ is not reached, meaning that no active job misses its deadline. This execution respects an AFM strategy as in every possible criticality configuration, a LO criticality job is active, iff $policy^{\overrightarrow{HI}}(\tau_i) = 0$, i.e. iff the fault mode policy states that the activation of jobs of task τ_i are not stopped for the current criticality configuration. A scheduling algorithm where no active job misses its deadline and respecting an AFM strategy can be computed using the wining strategy where $\forall t \in \mathbb{R}^+$

$Sched(Exec(t), t) = \tau_{i,k}$,with k the kth active job of τ_i, if there exists ξ in T_Γ with a last configuration $(q, \mathbf{v}, t)$ equivalent to $Exec(t)$ and $f(\xi) = e$ with e is a transition from act_i to $exeLO_i$.

Now, suppose that the task set Γ is AFM feasible according to a job level fixed priority algorithm and an AFM strategy. Thus, there exists a job level fixed priority scheduling algorithm *Sched* respecting the AFM strategy where no active job misses its deadline. Using the scheduling algorithm, we can compute a wining strategy f for the timed game defined by T_Γ and Formula 1 with

$f(\xi)$ = take transition from act_i to $exeLO_i$ if the last configuration $(q, \mathbf{v}, t)$ of ξ is equivalent to $Exec(t)$, $act_i \in q$ and $Sched(Exec(t), t) = \tau_{i,k}$.

Theorem 2 (Exact FP AFM schedulability test) *The task set Γ is fixed priority (FP) AFM schedulable (w.r.t. a fault mode policy) iff the network of timed game automata modeling Γ satisfies the CTL Formula 2.*

$$AG\neg(\bigvee_{\tau_i\in\Gamma} stop_i)\bigwedge\neg(\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in lp(i)}(act_i\wedge exeLO_j)\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in lp(i)}(act_i\wedge exeHI_j)\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in lp(i)}(preLO_i\wedge exeLO_j)\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in lp(i)}(preLO_i\wedge exeHI_j)\bigvee_{\tau_i\in\Gamma^{HI}}\bigvee_{\tau_j\in lp(i)}(preHI_i\wedge exeLO_j)\bigvee_{\tau_i\in\Gamma^{HI}}\bigvee_{\tau_j\in lp(i)}(preHI_i\wedge exeHI_j)) \tag{2}$$

Proof. Formula 2 is satisfied iff there exists an execution where no job misses its deadline (state $stop_i$ not reached) and if a job of a task τ_i cannot be active (state act_i) or preempted (state $preLO_i$ or $preHI_i$) if a job of a task τ_j of lower priority is executed (state $exeLO_j$ or $exeHI_j$). Thus the execution of jobs is done according to a fixed priority scheduling algorithm.

Theorem 3 (Exact EDF AFM schedulability test) *The task set Γ is earliest deadline first (EDF) AFM schedulable (w.r.t. a fault mode policy) iff the network of timed game automata modeling Γ satisfies the CTL Formula 3.*

$$AG\neg(\bigvee_{\tau_i\in\Gamma} stop_i)\bigwedge\neg(\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in\Gamma}(act_i\wedge exeLO_j\wedge p_{ij})\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in\Gamma^{HI}}(act_i\wedge exeHI_j\wedge p_{ij})\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in\Gamma}(preLO_i\wedge exeLO_j\wedge p_{ij})\bigvee_{\tau_i\in\Gamma}\bigvee_{\tau_j\in\Gamma^{HI}}(preLO_i\wedge exeHI_j\wedge p_{ij})\bigvee_{\tau_i\in\Gamma^{HI}}\bigvee_{\tau_j\in\Gamma}(preHI_i\wedge exeLO_j\wedge p_{ij})\bigvee_{\tau_i\in\Gamma^{HI}}\bigvee_{\tau_j\in\Gamma^{HI}}(preHI_i\wedge exeHI_j\wedge p_{ij})) \tag{3}$$

where p_{ij} is a state of an observer automaton reachable when $d_i - d_j > D_i - D_j$.

Proof. In earliest deadline first (EDF) schedulability, priorities are assigned to jobs dynamically according to their absolute deadlines. Formula 3 is satisfied iff there exists an execution where no job misses its deadline (state $stop_i$ not reached) and a job of a task τ_i cannot be active (state act_i) or preempted (state

$preLO_i$ or $preHI_i$) if a job of a task τ_j with an absolute deadline less close to its deadline ($D_j - d_j > D_i - d_i$) is executed (state $exeLO_j$ or $exeHI_j$). Thus the execution of jobs is done according to EDF algorithm.

6 Illustrative Example

We illustrate our approach using a task set Γ_1 with $\tau_1 = (HI, 3, 10, 10, (1, 2))$, $\tau_2 = (HI, 2, 8, 8, (2, 4))$ and $\tau_3 = (LO, 1, 4, 4, (2, 2))$. The CPU utilization of Γ_1 in the case where no LO criticality job is stopped and all the jobs of HI criticality tasks are critical is equal to 1.2, i.e. the task set is not feasible if we don't scarifice some LO criticality jobs when the criticality of the system is HI.We present in sections 6.1 and 6.2 two benefits of using the AFM strategy.

6.1 Decrease the number of sacrificed LO criticality jobs

The first benefit of our approach is that the designer can define a fault mode policy with the aim of reducing the number of jobs that are sacrificed, i.e. that are no more activated when the criticality of the system is HI.

We compare our method with the classical adaptive mixed criticality strategy (AMC) [4]. In AMC, when the criticality of the system is HI, LO criticality tasks are no more activated, and all jobs of HI criticality tasks are supposed to have an execution time equal to their HI WCET. We consider, also for AMC, that the system returns to LO mode at the first instant where no active job is critical.

Using the sufficient schedulability test, AMC_{rtb}, of AMC presented in [4], we cannot conclude that Γ_1 is schedulable using AMC, however we use the AFM strategy with a policy, $policy_{AMC}$, defined by $\overrightarrow{policy_{AMC}^{HI}}(\tau_3) = 1$ for all the criticality configurations where τ_1 or τ_2 have a critical job. Using the exact schedulability test of Formula 2, we prove that the task set Γ_1 is fixed priority AFM schedulable according to $policy_{AMC}$.

We compare the AMC strategy to an AFM strategy using the fault mode policy, $policy_1$, where $\overrightarrow{policy_1^{HI}}(\tau_3) = 1$ for all the criticality configurations where τ_2 has a critical job and $\overrightarrow{policy_1^{HI}}(\tau_3) = 0$ otherwise. In other words, LO criticality jobs are sacrified only when a job of task τ_2 is critical. Using the exact schedulability test of Formula 2, we prove that the task set Γ_1 is fixed priority AFM schedulable according to $policy_1$. We conclude that there is no need to scarify jobs of task τ_3 when only jobs of tasks τ_1 are critical, this will reduce the number of LO criticality jobs to stop to ensure the schedulability of the system.

To illustrate this, we consider an execution scenario in a time window from 0 to 44. In this scenario, the two last jobs of τ_1 are critical and the three last jobs of τ_2 are critical. We remind that a job is critical if the job does notify its completion after the execution of its WCET at the LO criticality level. Critical jobs are presented in black in the figures.

Figure 3 represents the schedule using AMC. At time 12, the system moves to HI criticality mode until time 17. As a consequence, the forth job of task τ_3 is not activated. Note that even if the second job of task τ_2 is not critical, its WCET is equal to 4 as all jobs of HI criticality tasks are supposed to have an execution time equal to their HI WCET when the criticality of the system is HI

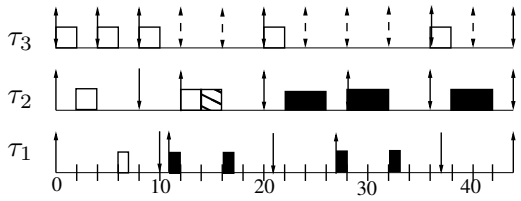

Fig. 3. FP Scheduling of Γ_1 using AMC

in AMC. The criticality of the system is again HI in the time interval $[24, 26]$, $[28, 33]$ and $[40, 42]$. Using AMC, we can see that 6 jobs of τ_3 are not activated.

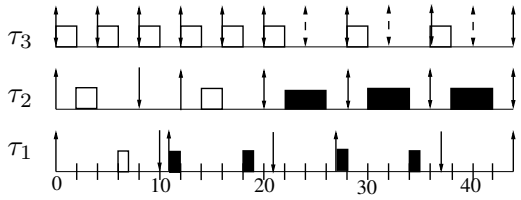

Fig. 4. FP Scheduling of Γ_1 using AFM and $policy_1$

In Figure 4, we can see that for the same execution scenario, using AFM strategy with FP scheduling and $policy_1$, only 3 jobs of τ_3 are not activated. In fact, the others job of task τ_3 are activated as even if the criticality mode of the system is HI, only the task τ_1 has a critical job.

6.2 LO criticality jobs are necessary in some HI criticality configurations

Even if some tasks are classified as LO criticality tasks by the designer, they may be necessary for the good functioning of the system. For example, in the case where task τ_3 is a non critical task performing some image analysis that are necessary when both jobs of τ_1 and τ_2 are critical to help the system to recover.

In this case, even if the task set is schedulable using $policy_1$, the system is unsafe because, as we can see in Figure 4, at instant 32, jobs of both τ_1 and τ_2 are critical, and the job of τ_3 is not executed, while it is necessary to ensure the safety of the system.

To test if the task set is schedulable if LO criticality tasks are not stopped when jobs of both τ_1 and τ_2 are critical, we use the fault mode policy $policy_2$. This policy is defined with, $policy_2^{\overrightarrow{HI}}(\tau_3) = 1$ for all the criticality configurations where "only" τ_2 has a critical job and $policy_2^{\overrightarrow{HI}}(\tau_3) = 0$ otherwise.

Using the exact schedulability test of Formula 2, we prove that the task set is not FP AFM schedulable according to $policy_2$. As we can see in Figure 5, at time 32 the two tasks τ_1 and τ_2 have a critical job, in this case, according to

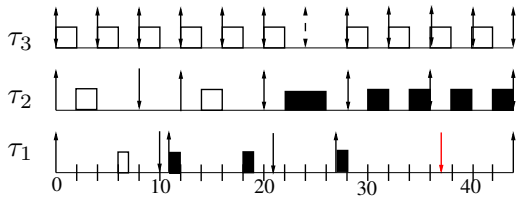

Fig. 5. FP Scheduling of Γ_1 using AFM and $policy_2$

the policy, the job of task τ_3 is activated. Using this policy, the third job of τ_1 misses its deadline.

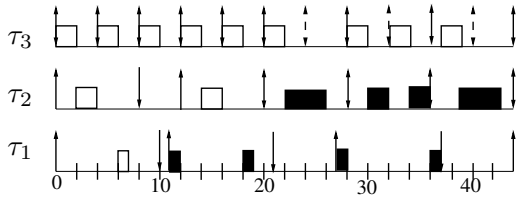

Fig. 6. EDF Scheduling of Γ_1 using AFM and $policy_2$

However, using the exact schedulability test of Formula 3, we prove that the task set is EDF AFM schedulable according to $policy_2$. As we can see in Figure 6, at time 32 the two tasks τ_1 and τ_2 have a critical job, in this case, according to the policy, the job of task τ_3 is activated.

7 Conclusion

In this paper, we introduce a scheduling strategy for the mixed criticality real-time scheduling problem where the designer can define his own policy to deal with low criticality tasks when the criticality of the system is high. For this model, we propose exact feasibility and schedulability tests for job level fixed priority algorithms based on CTL model checking for timed game automata. Using an example, we illustrate the benefits of the proposed strategy. We are aware that our exact tests face the state explosion problem, since the upper bound complexity of model checking on time game automata is EXPTIME, however, as future work, we plan to propose a more specific game model taking into account the characteristics of our real-time scheduling problem, our intuition is that only a subset of configuration is needed to prove the feasibility of the problem. An other direction, is to be able to generate the fault mode policy.

References

1. Abdeddaïm, Y.: Link to upload uppaal-tiga files. `https://perso.esiee.fr/~abdedday/VECOS2020.tar.gz`

2. Abdeddaïm, Y., Masson, D.: Real-time scheduling of energy harvesting embedded systems with timed automata. In: 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA. pp. 31–40 (2012)
3. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Automata, Languages and Programming, 17th International Colloquium, ICALP. pp. 322–335 (1990)
4. Baruah, S.K., Burns, A., Davis, R.I.: Response-time analysis for mixed criticality systems. In: Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS. pp. 34–43 (2011)
5. Baruah, S.K., Vestal, S.: Schedulability analysis of sporadic tasks with multiple criticality specifications. In: 20th Euromicro Conference on Real-Time Systems, ECRTS. pp. 147–155 (2008)
6. Bate, I., Burns, A., Davis, R.I.: A bailout protocol for mixed criticality systems. In: 27th Euromicro Conference on Real-Time Systems, ECRTS. pp. 259–268 (2015)
7. Bate, I., Burns, A., Davis, R.I.: An enhanced bailout protocol for mixed criticality embedded software. IEEE Trans. Software Eng. **43**(4), 298–320 (2017)
8. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: Computer Aided Verification, 19th International Conference, CAV. pp. 121–125 (2007)
9. Burn, A.: Mixed criticality - a personal view. vol. 5. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015)
10. Burns, A., Davis, R.I.: Mixed criticality systems - a review. `https://www-□users.cs.york.ac.uk/~burns/review.pdf`
11. Fleming, T., Burns, A.: Incorporating the notion of importance into mixed criticality systems. In: 3rd workshop on Mixed Criticality Systems, WMC (2013)
12. Gettings, O., Quinton, S., Davis, R.I.: Mixed criticality systems with weakly-hard constraints. In: Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS. pp. 237–246 (2015)
13. Huang, P., Kumar, P., Stoimenov, N., Thiele, L.: Interference constraint graph - A new specification for mixed-criticality systems. In: Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA. pp. 1–8 (2013)
14. Kozen, D. (ed.): Logics of Programs, Workshop, Lecture Notes in Computer Science, vol. 131. Springer (1982)
15. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT **1**(1-2), 134–152 (1997)
16. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: STACS, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, Proceedings. pp. 229–242 (1995)
17. Tamas-Selicean, D., Pop, P.: Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In: Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS. pp. 24–33 (2011)
18. Tindell, K., Alonso, A.: A very simple protocol for mode change. In: Technical report, Universidad Politecnica de Madrid (1996)
19. Vestal, S.: Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proceedings of the 28th IEEE Real-Time Systems Symposium RTSS. pp. 239–243 (2007)