View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Universidade do Minho: RepositoriUM

A Compositional Monitoring Framework for Hard Real-Time Systems

André de Matos Pedro[1], David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto[2]

[1] CISTER/INESC TEC, ISEP, Polytechnic Institute of Porto, Portugal
{anmap,dmrpe,lmp}@isep.ipp.pt
[2] HASLab/INESC TEC & Universidade do Minho, Portugal
jsp@di.uminho.pt

Abstract. Runtime Monitoring of hard real-time embedded systems is a promising technique for ensuring that a running system respects timing constraints, possibly combined with faults originated by the software and/or hardware. This is particularly important when we have real-time embedded systems made of several components that must combine different levels of criticality, and different levels of correctness requirements. This paper introduces a compositional monitoring framework coupled with guarantees that include time isolation and the response time of a monitor for a predicted violation. The kind of monitors that we propose are automatically generated by synthesizing logic formulas of a timed temporal logic, and their correctness is ensured by construction.

1 Introduction

Real-time systems (RTSs) range from simple, isolated components to large, highly complex and inherently concurrent systems. They act upon a variety of environments which are frequently very dynamic and hard to capture during design time. Therefore, developing an RTS can easily become a very difficult task to complete. However, even in the presence of potentially complex requirements, the design and development processes for RTSs limit themselves to model-driven techniques and intensive testing and fault-injection, which are known to allow the existence of human introduced errors. At later stages of the development cycle such errors can become highly expensive and very hard to tackle, even with the number of static analysis tools available. A notable example is in the area of scheduling analysis, where schedules for task sets are obtained by a rigorously defined scheduling algorithm. In hard RTSs the scheduling guarantees for task sets are obtained prior to the execution of the system. It is also often the case that schedulability analysis has to be performed in a compositional framework, such as the one presented in [9,22], in order to determine a valid schedule for the system (*e.g.*, when the system is made of a set components, each of which with its own set of tasks and local scheduling policy).

On the more rigorous side of RTS development, formal methods have been introduced progressively in the development cycle, most of which are based on temporal logic. While standard temporal logics yield a natural and abstract framework for the analysis of safety and liveness properties [21], these logics fail to capture the specific timing properties of RTSs [13]. This limitation is tackled by a set of timed temporal logics [1], and many of these logics have already been used to develop model checking tools [5]. However, model checking has its own pitfalls, namely when the size of the state space of the model that captures the RTS under consideration is too large to be mechanically analyzed by a tool implementing a model checking algorithm. Moreover, it might be the case that the properties to be checked cannot be captured rigorously at the abstract level of the model of the system.

In order to address the cases where static analyses of an RTS fail, researchers have introduced the concept of *runtime verification* (RV). RV is a major complement to static methods because it can be used to check errors for which it is possible to conclude some property of interest based exclusively in knowledge that can be gathered only at execution time. Contrary to *ad hoc* instrumentation of runtime behavior, RV based approaches use formal specifications and synthesize them into *monitors*, that is, pieces of code that take partial traces of execution of the system and match them against the referred specifications and make a verdict. Moreover, monitors can be used both to verify and enforce the properties which are provided by components, even when the components assume the form of a black-box, as long as each component is coupled with a formal specification. A simple example of the power of RV is the case when the response to a property violation detection consists in shutting down a complex component and give control to a simpler, yet formally verified component. RV has been progressively adopted by the industry of real-time operating systems as described in [6].

In this paper we introduce a *compositional monitoring framework* (CMF) that allows us to make assumptions about the time isolation between components as well as the response times of the monitors. We apply this notion to components with different criticality assurances, and whose specific requirements shall be ensured statically and dynamically through schedulability analysis and runtime monitoring, respectively. To guarantee these frameworks' assumptions we use a fragment of the *metric temporal logic with durations* (MTL-$\int$) [14] to analyze the schedulability of the CMF, and to statically check the maximum response times of each of the generated monitors. To the best of our knowledge, this is the first approach that combines MTL-$\int$ with the generation of *monitors* with explicit durations, for RV of hard RTSs. The timing enforcers of the CMF are synthesized from MTL-$\int$ formulas.

The paper is organized as follows: in Section 2 we describe work that is related to the subject of this paper; in Section 3, we describe the model and architecture of the CMF; in Section 4 we introduce a version of MTL-$\int$, with a restricted syntax and augmented axiomatic system to handle the properties of our CMF; in Section 5 we describe the process that synthesizes MTL-$\int$ formulae into monitors; in Section 6 a set of guarantees provided by the CMF is introduced, including the response time bound guarantee of each monitor; in

Section 7 it is exemplified how to use monitors and a practical applicability of the proposed schedulability analysis is given; finally, Section 8 draws some conclusions and directions for further work.

2 Related Work

RV is being progressively introduced in RTS development in those corner cases for which static approaches are not strong enough. In the following we review theories and tools that are related to the ideas we are proposing in this paper, namely, monitor synthesis approaches based on timed temporal logics and their tools, as well as alternative techniques for schedulability analysis.

2.1 Monitor Based Approaches

Auguston and Takhtenbrot [3] describe a model-driven approach which dynamically enforces properties specified from statechart-based models via runtime monitoring. Monitors are automatically generated from formulas that specify the system's behavior, in a proposed assertion language, and their expressiveness always depends on the assertion language. Bauer *et al.* [4] propose an algorithm to efficiently generate monitors from TLTL formulae. Such monitors are able to specify real-time constraints from which verdicts can be made at any point of the execution. The three-valued notion of *timed linear-time temporal logic* $(\text{TLTL})_3$ is specially suitable for runtime monitoring since a complete set of traces is not available at runtime, and the monitors' specifications are increasingly evaluated. Nickovic *et al.* [16] describe a translation of MTL into deterministic timed automata. The full MTL language is considered, and no bounds are imposed in the future temporal connectives. The process first converts *metric temporal logic* (MTL) into non-deterministic timed automata, and then determinizes them. Another close research effort is the *runtime enforcement* of timed properties. In Pinisetty *et al.* [20], monitors are introduced to enforce properties with explicit time. This approach is useful to delay events (or messages) that arrive before the allowed time (e.g., when a buffer becomes full due to a premature arrival of an event).

Tool support for the monitorization of RTSs is scarce. Temporal Rover [8] is appropriate for monitoring of hard real-time systems due to the temporal constraints being specified by the MTL. However, the monitoring software is proprietary and many specifications are hidden from common users. Alves *et al.* [2] present the results of a formal computer-aided validation and verification of critical time-constrained requirements of the Brazilian Satellite Launcher flight software based on Temporal Rover. Pike *et al.* [17] introduce the Copilot tool which is able to monitor hard real-time systems. The tool is a compiler that supports a pre-defined streaming language in which properties shall be specified. The tool also generates a scheduler that guarantees the timing constraints of the system, and outcomes a constant-execution time and constant-space C program. However, no time specifications are allowed by the tool since they are statically ensured by a scheduler that is automatically generated. The correctness of the timed properties depends of this step. New features have later been added into the tool for distributed systems. A case study of a Byzantine fault-tolerant airspeed sensor system is described in [18]. More prominent experiments have been carried out recently as described by the report [19], where two case-studies using Copilot monitors are tested in a true avionic system. The authors also show the capability of their approach to cover such realistic settings.

2.2 Schedulability Analysis and Predictable Monitoring

Fersman et al. [11,12] introduces an interesting research effort that discards the classic schedulability analysis for uni-processor systems. The authors use *timed automata extended with real-time tasks* to specify the system behavior together with the scheduler behavior. Regarding these, the schedulability test remains a reachability analysis problem, which is normally solved by model checkers such as UPPAAL [5]. Recently, Fersman et al. [10] have showed that the schedulability for multi-processor systems is possible for non-preemptive and preemptive schedulers with constant execution time.

Work that addresses a predictable monitoring framework of temporal properties is proposed by Zhu et al. [24]. The authors take inspiration from classical schedulability analysis to find a response time bound for monitors using sporadic servers. However, no composability or duration of real time tasks were considered for runtime monitoring.

3 Proposed Framework

In this section we introduce our CMF, an abstract component-based framework that includes runtime monitors, thus supporting external observations at runtime. We begin by introducing the definitions of real-time task-sets and periodic resource models; event sequences; and lastly the framework.

We will assume *tasks sets* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, such that $n \in \mathbb{N}^+$ is the number of tasks $\tau_i = (p_i, e_i)$ where p_i and e_i are, respectively, the period and the worst-case execution time of τ_i. Each task $\tau_i \in \tau$ is periodic. A *periodic resource model* ω is a tuple (τ, π, θ, rm), where $\tau \subseteq \Gamma$, π is the *replenishment period*, θ is the *server budget*, and rm is the *rate monotonic* (RM) scheduling algorithm. The set of periodic resource models is denoted by Ω. The outputs of a resource model ω are *sequences of events*. Considering a pair (ω, τ_i) with $\omega \in \Omega$ and $\tau_i \in \tau$, each event can be of one of the following types: a *release-event* $\mathsf{erelease}(\omega, \tau_i)$; a *start-event* $\mathsf{estart}(\omega, \tau_i)$; a *sleep-event* $\mathsf{esleep}(\omega, \tau_i)$; a *resume-event* $\mathsf{eresume}(\omega, \tau_i)$; or a *stop-event* $\mathsf{estop}(\omega, \tau_i)$. In addition, we assume a parameterized event $\varepsilon(\omega_j, \tau_i, id)$ that denotes the critical events of a task, where id is the event identifier, and $\mathsf{erenewal}(\omega)$ denotes the budget release of a resource model. We denote sets of events by $\mathcal{E}$.

Event sequences are a formalism that allows us to describe the scheduler behavior, creating a generic event language that a system can produce. If a system produces unexpected event words, we shall consider it a faulty system. This abstraction also establishes an interface for temporal logic observations [14]. A sequence of events, also known as *execution trace*, is an infinite sequence

$$\rho = (e_1, t_1)(e_2, t_2)\cdots$$

of time-stamped events (e_i, t_i) with $e_i \in \mathcal{E}$ and $t_i \in \mathbb{R}^+$. The sequence satisfies monotonicity and progresses, *i.e.*, $t_i \leq t_{i+1}$ for all $i \in \mathbb{N}^+$, and for all $t \in \mathbb{R}^+$ there is some $i > 0$ such that $t_i > t$, respectively.

3.1 CMF model and architecture

The CMF model is composed of a set of elements of one of the following types:

- (Component) A *simple component* $C = (\Gamma, \omega, \vartheta, \phi)$ is a component, where ω is a resource model, ϑ is a scheduler, and ϕ is a set of properties to be verified at runtime. The scheduler ϑ behaves accordingly to a scheduling policy, such as a fixed-priority scheduler. The variable ϕ is a set of properties defined in a *program logic* to monitor the behavior of the task set Γ.
- (Hypervisor) A supervisor component manages several components allowing us to coordinate component executions with lower interference among them. Let $\mathrm{H} = (\Omega, \eta_p, \eta_m, \phi_h)$ denote a hypervisor, where Ω is a set of periodic resource models, η_p is a set of notational processors (these is for future work) that may be assigned to the required components, η_m is a set of notational memory blocks that each component is able to use, and ϕ_h is a set of properties that the hypervisor H shall employ.

The CMF architecture accommodates the previously defined components as depicted in Fig. 1. The monitor for each component is synthesized automatically by the set of monitor properties ϕ assigned to each component. ϕ is a logic formula corresponding to a specification, which can be seen as an assume/guarantee condition of a component. Our architecture manages the monitors by three levels of criticality, and joins similar monitors in similar resource models: M^h, M^m, and M^l, respectively. This management can be composed by n-levels. A monitor resource model is viewed as a resource model of a component. However, each component or hypervisor has a set of quasi omniscient monitors (resp. hypervisor monitor) that draws a verdict about the assumptions of the architecture that may be violated.

4 Metric Temporal Logic with Durations

In this section we introduce a fragment of the formal logic MTL-$\int$ [14], whose evaluation is carried out with respect to sequences of events produced by resource models. MTL-$\int$ enables the automatic generation of monitors and, at the same time, it allows us to statically ensure properties about our architecture (resp. the response time bound of monitors). Such monitors are able to observe and

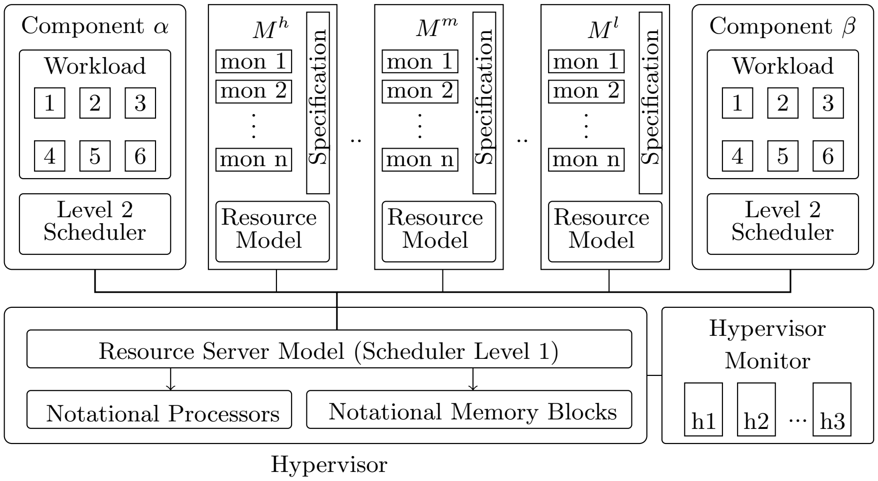


Fig. 1: Component-based Monitoring Architecture (CMA)

check timed constraints, as well as the *worst case execution time* (WCET) of the tasks of a given resource model. Nevertheless, computing the value of MTL-$\int$ formulae may not always be possible. In order to cope with this limitation, we consider a fragment of MTL-$\int$ that uses only the $\leq$, $<$, and $=$ relations over terms, and and we exclude occurrences of functions in terms. We also consider a strong form of the existential quantifier operator, which we denote by $\exists'$.

Definition 1 (MTL-$\int$). *Let $\mathcal{P}$ be a set of propositions and $\mathcal{V}$ a set of logical variables. Logical variables in $\mathcal{V}$ are mapped to $\mathbb{R}$. The syntax of MTL-$\int$ is defined inductively, as follows:*

$$\delta ::= \alpha \mid x \mid \textstyle\int^{\delta} \varphi$$

$$\varphi ::= p \mid \delta_1 {\sim} \delta_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1\, U_{\sim\gamma}\, \varphi_2 \mid \varphi_1\, S_{\sim\gamma}\, \varphi_2 \mid \exists x\, \varphi \mid \exists' x\, \varphi$$

where δ are terms, $\int^{\delta} \varphi$ is the duration of the subformula φ in the interval $[0, \delta]$, x is a continuous variable in $\mathcal{V}$, $p \in \mathcal{P}$ is an atomic proposition, $\gamma \in \mathbb{R}_{\geq 0}$, $\sim \in \{<, \leq, =\}$, and $\alpha \in \mathbb{R}$.

We are now able to define the semantic of the MTL-$\int$. The semantic of MTL-$\int$ is separated in two parts: *terms* and *formulas*. The semantic of terms is defined using the notation $\mathcal{T}[\![\tau]\!](\sigma, \vartheta)t$ in Table 1. All terms represent numerical values in $\mathbb{R}_0^+$. The term $\int^{\delta} \varphi$ is the integral over the Boolean function $B_{\phi(\sigma,\vartheta)}(t)$ (whose return value is 1 if $(\sigma, \vartheta, t) \models \phi$, and 0 otherwise). Since $B_{\phi(\sigma,\vartheta)}(t)$ behaves as a step function, it is always Riemann integrable. The same is not true in the full MTL-$\int$ logic. The semantic of the MTL-$\int$ formula is defined inductively in Table 1, where the satisfability of a formula ϕ in a model (σ, ϑ) at time t is defined by $(\sigma, \vartheta, t) \models \phi$. σ and ϑ are an observation function and a logic

Evaluation of the restricted MTL-$\int$ terms		
$\mathcal{T}[\![\alpha]\!](\sigma,\vartheta)t$	$=$	α
$\mathcal{T}[\![x]\!](\sigma,\vartheta)t$	$=$	$\vartheta(x)$
$\mathcal{T}[\![\int^{\delta}\phi]\!](\sigma,\vartheta)t$	$=$	$\begin{cases}\int_t^{t+\mathcal{T}[\![\delta]\!](\sigma,\vartheta)t} B_{\phi(\sigma,\vartheta)}(t_*)\ dt_* \text{ if } \mathcal{T}[\![\delta]\!](\sigma,\vartheta)t \geq 0\\ 0 \text{ otherwise}\end{cases}$
Evaluation of the restricted MTL-$\int$ formulas		
(σ,ϑ,t)	$\models$	p iff $\sigma(p)(t) = \mathit{true}$ and $t < \lvert\sigma\rvert$
(σ,ϑ,t)	$\models$	$\delta_1 \sim \delta_2$ iff $\mathcal{T}[\![\delta_1]\!](\sigma,\vartheta)t \sim \mathcal{T}[\![\delta_2]\!](\sigma,\vartheta)t$
(σ,ϑ,t)	$\models$	$\phi_1 \vee \phi_2$ iff $(\sigma,\vartheta,t) \models \phi_1$ or $(\sigma,\vartheta,t) \models \phi_2$
(σ,ϑ,t)	$\models$	$\neg\phi$ iff $(\sigma,\vartheta,t) \not\models \phi$
(σ,ϑ,t)	$\models$	$\phi_1\ U_{\sim\gamma}\ \phi_2$ iff $\exists t' \in \mathbb{R}_{\geq 0}$ such that
		$t \leq t' \sim t + \gamma \wedge (\sigma,\vartheta,t') \models \phi_2$, and
		$\forall t'' \in \mathbb{R}_{\geq 0},\ t \leq t'' < t', (\sigma,\vartheta,t'') \models \phi_1$
(σ,ϑ,t)	$\models$	$\phi_1\ S_{\sim\gamma}\ \phi_2$ iff $\exists t' \in \mathbb{R}_{\geq 0}$ such that
		$t - \gamma \sim t' \leq t \wedge (\sigma,\vartheta,j) \models \phi_2$, and
		$\forall t'' \in \mathbb{R}_{\geq 0},\ t' < t'' \leq t, (\sigma,\vartheta,t'') \models \phi_1$
(σ,ϑ,t)	$\models$	$\exists x\, \varphi$ iff $\exists v \in \mathbb{R}$ st. $(\sigma,\vartheta_x^v,t) \models \phi$
(σ,ϑ,t)	$\models$	$\exists' x\, \varphi$ iff there exists a value $v \in \nu$ such that $(\sigma,\vartheta_x^v,t) \models \phi$

Table 1: Semantic of the restricted MTL-$\int$

Operator	Abbreviation	Equivalent Formula
Eventually	$\Diamond_{\sim\gamma}\phi$	$\mathit{true}\ U_{\sim\gamma}\ \phi$
Always	$\Box_{\sim\gamma}\phi$	$\neg(\Diamond_{\sim\gamma}\neg\phi)$
Next	$\bigcirc_{\phi_1}\phi_2$	$\phi_1\ U_{\sim\infty}\ \phi_2$
Implies Next	$\phi_1 \overset{\bigcirc}{\Longrightarrow} \phi_2$	$\neg\phi_1 \vee \bigcirc_{\phi_1}\phi_2$

Table 2: Syntactic abbreviations for our MTL-$\int$ fragment

environment defined as usual [14]. The set ν contains the time stamps for the differential between the observed truth values given by the σ function (allowing us to formulate some axioms to turn our logic evaluation in a computable function). We will use the abbreviations *eventually* ($\Diamond$) and *always* ($\Box$) as usual.

In the remaining of the paper we will frequently refer to the abbreviations presented in Table 2 in order to ease the presentation of properties that describes the monitor behavior. For illustrative purposes, we now introduce a practical example of the expressive power of MTL-$\int$'s language.

Example 1. To ensure that a monitor task responds in a bounded response time, the formula $\psi_1 \implies \Diamond_{\leq\gamma}\,\psi_2$ is sufficient. The proposition ψ_1 describes a set of events that may violate the system, the proposition ψ_2 describes the task invocation, and γ is the maximum expected response time bound. Informally, the formula means that if a fault event occurs, then the task executes within γ time units.

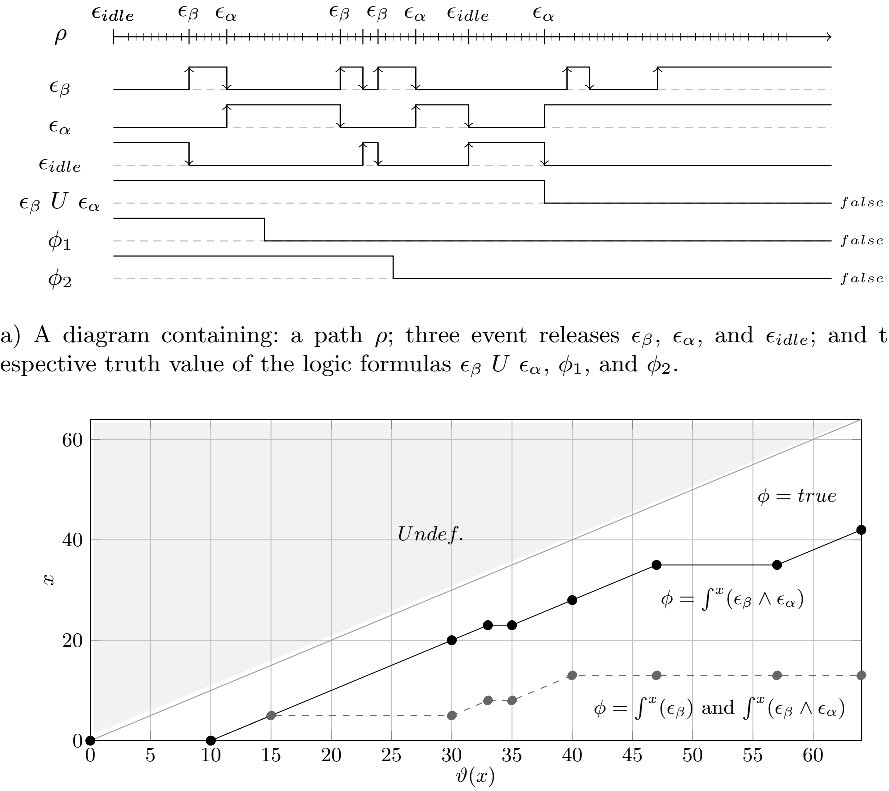
4.1 MTL-$\int$ Axiomatization

Having restricted the original MTL-$\int$ described in [14], we are able to fix new axioms for durations. Most interesting is that such axioms will allow us to turn our MTL-$\int$ fragment computable. As the meaning of the duration term $\int^r \phi$ is defined as an integral, and the relation $\leq$ as a term operator, we have axioms that capture properties of integrals over the $\leq$ operator. They are, as follows:

A1. $\exists x\ \alpha \leq \int^{\alpha} \phi \equiv \alpha \leq \int^{\alpha} \phi$;
A1′. $\exists x\ \int^{\alpha} \phi \leq \alpha \equiv \int^{\alpha} \phi \leq \alpha$;
A2. $\exists x\ x \leq \int^{\alpha} \phi \equiv \min(x) \leq \int^{\alpha} \phi$;
A2′. $\exists x\ \int^{\alpha} \phi \leq x \equiv \int^{\alpha} \phi \leq \max(x)$;
A3. $\exists x\ \alpha \leq \int^{x} \phi \equiv \alpha \leq \int^{\max x} \phi$;
A3′. $\exists x\ \int^{x} \phi \leq \alpha \equiv \int^{\min x} \phi \leq \alpha$;
A4. $\exists x\ \alpha \leq \int^{\int^{\int^{x} \phi_n} \phi_1} \phi \equiv \alpha \leq \int^{\int^{\int^{\max x} \phi_n} \phi_1} \phi$;
A4′. $\exists x\ \int^{\int^{\int^{x} \phi_n} \phi_1} \phi \leq \alpha \equiv \int^{\int^{\int^{\min x} \phi_n} \phi_1} \phi \leq \alpha$;
A5. $\exists x\ \exists y\ \int^{x} \phi_1 \leq \int^{y} \phi_2 \equiv \int^{\min(x)} \phi_1 \leq \int^{\max(y)} \phi_2$;
A6. $\exists x\ \int^{x} \phi_1 \leq \int^{x} \phi_2 \equiv \exists' p\ \int^{p} \phi_1 \leq \int^{p} \phi_2$.

Axioms A1 and A1′ remove the existential operator in the evaluation of a constant inequality over a constant interval. Axioms A2 and A2′ substitute the existential quantification with a minimum and maximum that a variable x can take according to the constraints applied to x. If x is unbounded the minimum is zero, and the maximum is infinity. Axioms A3 and A3′ reduce the existential operator over a duration into a minimum and maximum inequality. Axioms A4 and A4′ deal with nested durations. For the remaining axioms we have variables and duration primitives of MTL-$\int$ in both sides of the operator $\leq$. Once we have an infinite observation over a path composed by finite pieces we have a procedure to compute the truth value of this type of formula. Axiom A5 establishes that the order relation between two durations specified by different variables is the same as if their minimum and maximum allowed values are considered. This can be seen as a desynchronization of durations axiom. Axiom A6 means that the existential quantification of duration terms over the $\leq$ operator can be reduced by substituting the existential quantification in points along the path where observation of MTL-$\int$ formulas is made. This can be seen as a synchronization of durations axiom.

Example 2. Consider an application of Axiom A6 with the two MTL- formulas $\phi_1 = \int^{|\rho|}(\epsilon_\beta \wedge \epsilon_\alpha) \leq 10$ and $\phi_2 = \int^{|\rho|}(\epsilon_\beta) \leq 10$. To easily understand their use we will show two figures. Figure 2a depicts the sequence of events ρ, their respective activation times as well as the evaluation of three formulas $\epsilon_\beta\ U\ \epsilon_\alpha$, ϕ_1, and ϕ_2 over the duration of the sequence ρ. Figure 2b depict an evaluation of the formula $\exists\, x\ \int^{x}(\epsilon_\beta) \leq \int^{x}(\epsilon_\beta \wedge \epsilon_\alpha)$, where the undefined values are shown in gray. Note that the duration of any MTL-$\int$ formula cannot be greater than the duration of a *true* formula as depicted by the figure. In practice, we shall conclude that the points to compare are the time stamps of events of each path ρ since these points form a set of monotonic increasing piecewise linear segments.



(a) A diagram containing: a path ρ; three event releases ϵ_β, ϵ_α, and ϵ_{idle}; and the respective truth value of the logic formulas $\epsilon_\beta \; U \; \epsilon_\alpha$, ϕ_1, and ϕ_2.

(b) The graph depicts the formula $\int^x(\epsilon_\beta)$ and $\int^x(\epsilon_\beta \wedge \epsilon_\alpha)$ which allows us to visually check the formula $\forall x \;\; \int^x(\epsilon_\beta) \leq \int^x(\epsilon_\beta \wedge \epsilon_\alpha)$ in the finite interval $[0, 64)$.

Fig. 2: Diagram of a path (a) and respective duration computation (b)

5 Evaluation of MTL-$\int$ formulas

In order to synthesize monitors for our CMF we have defined a fragment of MTL-$\int$ that is able to describe durations for RTS, an algorithm to evaluate logic and their WCET estimation, and the time complexity analysis of the algorithm. We have implemented the semantics of our MTL-$\int$ fragment and implemented it in OCaml language [23].

5.1 The evaluation algorithm

The semantics of MTL-$\int$ introduced in [14] may not be fully computable due to the real numbered existential and universal quantifications. In addition, the axioms established previously allow us to compute the non negative real number existential and universal quantifications, and to enable the WCET estimation.

However, the validity of the argument is also shown by the evaluation function of Algorithm 1 where the MTL-$\int$ semantics is codified exclusively using functions applied to lists. This algorithm evaluates MTL-$\int$ formulas and produces Boolean verdicts.

Some notations need to be introduced before the description of the algorithm, `obs` is a function that corresponds to an observation, `env` is a logical environment, and `mt` is a term function which evaluates terms such as α, x, and $\int^{\alpha} \phi$. These functions are defined according to the semantics of [14]. Note that these terms are always computable, *i.e.*, terms are non negative real valued numbers.

The functions $\Phi_{\int}$, Φ_x, and Φ are special functions that rewrite formulas by applying the axioms described previously to compute new values. Next, we present an example to clarify the role of these functions.

Example 3. Suppose that we have an existential quantification of x over the formula $\alpha \leq \int^{\Phi(\sigma,\vartheta)tr_1} \phi$. This formula will be rewritten to $\alpha \leq \int^{\max(mt\,(\sigma,\vartheta)\,t\,r_1)} \phi$ which means that the maximum allowed value is enough to know if there exists an x sufficiently large to validate the inequality condition. The other functions follow the same principle, but considering $x \leq \int$, $\int \leq x$, and $\int \leq \int$.

5.2 The time complexity of our evaluation algorithm

In order to provide an analysis of the CMF, the WCET of monitors should be supplied. We address a pessimistic bound for our evaluation function EVAL based on time complexity, $T(m, n) = n \times m$, where n is the length of the formula and m the length of the trace to be consumed by the monitor. This means that in the worst case we have $n \times m$ comparisons between list elements, and the WCET can be computed by multiplying the constant cost that each list element takes. Note that our algorithm is based on functions applied to lists, $forall$, $exists$, and $fold\_left$, and a more accurate estimation of the WCET defining a recursive cost function could easily be employed.

5.3 Runtime monitoring as the evaluation of an MTL-$\int$ formula

After establishing a computable logic, a monitor may be seen as a procedure to evaluate a formula. Thus, we introduce the notion of monitor generated from an MTL-$\int$ formula.

Definition 2. *A monitor is a process that evaluates one specific bounded formula in the MTL-$\int$ fragment.*

Monitors belong to resources models (or components), and are represented by tasks (one formula produces one task). Our algorithm is adequate to estimate the WCET of a monitor even with a pessimistic bound, and also to be employed in practice (as will be seen later). Note that WCET parameters of monitors are required in order to make a prior schedulability analysis which ensures a certain responsiveness for the monitor components. Note also that this process is an alternative to the synthesis approaches using automata theory [16].

```
in  : An execution trace $\rho$ of length $|\rho|$, and a logic formula $\phi$.
out: A Boolean evaluation of the logic formula $\phi$ over the trace $\rho$.
let EVAL $\rho$ t $\phi$ = let $\sigma$ = obs $\rho$ in let $\vartheta$ = env in MODELS $(\sigma, \vartheta, t)$ $\phi$

let MODELS $(\sigma, \vartheta, t)$ $\phi$ = match $\phi$ with
  | p $\longrightarrow$ $\sigma$.evaluate p t and $\sigma$.interval t
  | $\neg$ $(\phi_1)$ $\longrightarrow$ not (MODELS $(\sigma, \vartheta, t)$ $\phi_1$)
  | $\vee$ $(\phi_1, \phi_2)$ $\longrightarrow$ MODELS ($\sigma$, $\vartheta$, t) $\phi_1$ or MODELS ($\sigma$, $\vartheta$, t) $\phi_2$
  | $U_{<\gamma}$ $(\phi_1, \phi_2)$ $\longrightarrow$ let (b,t') = exists (fun a $\longrightarrow$ MODELS $(\sigma, \vartheta, a)$ $\phi_2$) ($\sigma$.intrv t (t $+\gamma - \epsilon$)) in b and forall (fun $t''$ $\longrightarrow$ MODELS $(\sigma, \vartheta, t'')$ $\phi_1$) ($\sigma$.intrv t $(t' - \epsilon)$)
  | $U_{=\gamma}(\phi_1, \phi_2)$ $\longrightarrow$ MODELS $(\sigma, \vartheta, \gamma)$ $\phi_2$ and forall (fun $t''$ $\rightarrow$ MODELS $(\sigma, \vartheta, t'')$ $\phi_1$) ($\sigma$.intrv t $(t' - \epsilon)$)
  | $\exists$ $(var, \phi_1)$ $\rightarrow$ exists (fun n $\longrightarrow$ let () = $\vartheta$.add $var$ n in MODELS $(\sigma, \vartheta, t)$ $\phi_1$) ($\sigma$.intrv t ($\vartheta$.bound var))
  | $\sim$ ($\int^{r_1} \phi_1$,$\alpha$) $\rightarrow$ MT $(\sigma, \vartheta)$ t ($\int^{\Phi\,(\sigma,\vartheta)\,t\,r_1} \phi_1$) $\sim$ MT $(\sigma, \vartheta)$ $t$ $\alpha$
  | $\sim$ ($\alpha$, $\int^{r_1} \phi_1$) $\rightarrow$ MT $(\sigma, \vartheta)$ $t$ $\alpha$ $\sim$ MT $(\sigma, \vartheta)$ t ($\int^{\Phi\,(\sigma,\vartheta)\,t\,r_1} \phi_1$)
  | $\sim$ ($\int^{r_1} \phi_1$,$x$) $\rightarrow$ MT $(\sigma, \vartheta)$ t ($\int^{\Phi_x\,(\sigma,\vartheta)\,t\,r_1} \phi_1$) $\sim$ MT $(\sigma, \vartheta)$ $t$ $x$
  | $\sim$ ($x$, $\int^{r_1} \phi_1$) $\rightarrow$ MT $(\sigma, \vartheta)$ $t$ $x$ $\sim$ MT $(\sigma, \vartheta)$ t ($\int^{\Phi_x\,(\sigma,\vartheta)\,t\,r_1} \phi_1$)
  | $\sim$ ($\int^{r_1} \phi_1$,$\int^{r_2} \phi_2$) $\rightarrow$ MT $(\sigma, \vartheta)$ t ($\int^{\Phi_{\int}\,(\sigma,\vartheta)\,t\,r_1} \phi_1$) $\sim$ MT $(\sigma, \vartheta)$ t ($\int^{\Phi_{\int}\,(\sigma,\vartheta)\,t\,r_1} \phi_1$)

let MT $(\sigma, \vartheta)$ $t$ $r$ = match $r$ with
  | $\alpha \rightarrow \alpha$
  | $x \rightarrow \vartheta(x)$
  | $\int^{r} \phi$ $\rightarrow$ if MT $(\sigma, \vartheta)$ $t$ $r \geq 0$ and ($\sigma$.intrv $t$ $(t +$ MT$(\sigma, \vartheta)\, t\, r)) \geq 2$ then INT $t$ $(t +$ MT$(\sigma, \vartheta)\, t\, r)$ (ONE $\phi\,(\sigma, \vartheta)\, t_1$) else 0

let ONE $\phi\,(\sigma, \vartheta)\, t_1$ = if MODELS $(\sigma, \vartheta, t_1)$ $\phi$ then 1 else 0

let INT $t_b$ $t_e$ $f$ = let v,_ = fold_left (fun $(a, t_l), t \rightarrow (a + f(t) \cdot (t - t_l), t)$) (0, hd $\sigma$.intrv $t_b$ $t_e$) (tl $\sigma$.intrv $t_b$ $t_e$) in v
```

Algorithm 1: MTL-$\int$ Evaluation Algorithm, with $\sim \in \{<, \leq, =\}$.

Our approach is particularly suited to handle the reorganization of monitors that belong to different resource models. Once the monitor synthesis process provides one task per logical formula, the performance is affected. This is due to the increasing number of tasks that may reduce substantially the systems' utilization (a known problem of the fixed-priorities schedulers). To relax this problem, a clustering of monitor tasks based on the execution time, deadline, and response time bound is a possible solution.

After generating series of monitors the clusters are classified within n-level components, *i.e.*, each cluster is assigned to one resource model independently of the system under monitoring. This guarantees non-interference of time between monitors and the system's schedulability.

6 Guaranteeing Real-Time Constraints using MTL-$\int$ for CMF

Constraints for our CMF shall be statically ensured using our logic fragment as basis [7]. WCET violations of one or more tasks may interfere with other non monitoring tasks resulting in an undesirable environment. This can be tackled by using higher priority tasks for monitor processes or by assigning monitors to independent resource models. However, to guarantee non interference between resource models, we shall ensure the correct behavior of such models by specifying their allowed budgets and periods. Other formulas are required to establish a complete formalization as described in [7].

Assuming a correct release of events, namely the $\mathsf{erenewal}(\omega)$, the budget supply is specified by the formula $\phi(\omega)$ equivalent to

$$\Box_{\leq\infty}\left(\mathsf{erenewal}(\omega) \overset{\bigcirc}{\Longrightarrow} \left(\Diamond_{=\pi}\ \mathsf{erenewal}(\omega)\right) \wedge \int^{\pi} \bigvee_{\tau_i\in\tau} evs^{+}(\omega,\tau_i) \leq \theta\right),$$

where ω is one resource model; π and θ their renewal period and budget, respectively; $\mathsf{erenewal}(\omega)$ is the budget renewal event, and $evs^{+}(\omega_j,\tau_i) \overset{def}{=} \mathsf{estart}(\omega_j,\tau_i) \vee \mathsf{eresume}(\omega_j,\tau_i) \vee \mathsf{erenewal}(\omega) \vee \mathsf{estop}(\omega_j,\tau_i) \vee \varepsilon(\omega_j,\tau_i,\cdot) \vee \mathsf{erelease}(\omega_j,\tau_i)$. This formula states that for each occurrence of the event $\mathsf{erenewal}(\omega)$ in the resource model ω, the duration of the other events until π time units does not overpasses the budget θ per period π. In this formula we assume the correct specification of the periodic event releases ($\mathsf{erenewal}(\omega,\tau_i)$), and the schedulability of the workload according to a fixed priority policy. Note that this assumption can be discarded using schedulability analysis based on task automata or following the instruction provided in [7].

In addition, the predictability of our framework with respect to event sequences can be established by identifying the relevant or critical events, and preserving the partial order of events arrival for monitor processes. We denote the critical events by the subset $\mathcal{E}^{\phi}_{cr} \subseteq \mathcal{E}$, and the prefix-tree which preserves the partial order of events for all possible executions by pt. Given these predictable traces pt we are able to ensure the response time of the monitor `mon_id` for each trace $\rho \in pt$ by the formula

$$\textstyle\bigwedge_{e\in\mathcal{E}^{\phi}_{cr}} e \Longrightarrow \Diamond_{\leq\gamma}\,\mathsf{estop}(\omega,\tau_1), \tag{1}$$

where $\mathsf{estop}(\omega,\tau_1)$ is the triggered event that monitors generate at the end of their complete execution, and $\mathcal{E}^{\phi}_{cr}$ is a set of the events used by formula ϕ.

Example 4. Assuming two resource models RS-A($\pi = 10,\ \theta = 8$) and RS-C($\pi = 5,\ \theta = 1$) described in Figure 3 containing three tasks (ts1 with period of 14 and WCET of 3; ts2 with period of 20 and WCET of 5; and ts3 with period of 27 and WCET of 7), and one task (ts1 with period of 33 and WCET of 4), respectively. We could see that to guarantee the maximum detection delay of the monitor task ts1 in RS-C, the trace depicted in the Figure 3 need to be

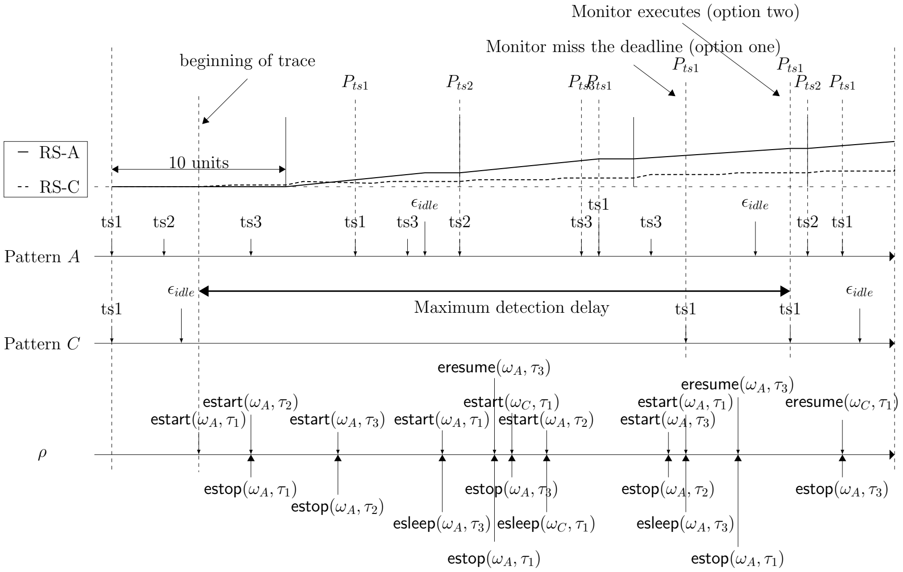

Monitor executes (option two)
Monitor miss the deadline (option one)
beginning of trace
P_{ts1}
P_{ts2}
$P_{ts2}P_{ts1}$
P_{ts1}
P_{ts1}
P_{ts2}
P_{ts1}
RS-A
RS-C
10 units
ϵ_{idle}
ϵ_{idle}
ts1
ts2
ts3
ts1
ts3
ts2
ts3
ts1
ts3
ts2
ts1
Pattern A
ϵ_{idle}
Maximum detection delay
ϵ_{idle}
ts1
ts1
ts1
Pattern C
ρ
eresume(ω_A, τ_3)
eresume(ω_A, τ_3)
estart(ω_A, τ_2)
estart(ω_C, τ_1)
estart(ω_A, τ_1)
eresume(ω_C, τ_1)
estart(ω_A, τ_1)
estart(ω_A, τ_3)
estart(ω_A, τ_1)
estart(ω_A, τ_2)
estart(ω_A, τ_3)
estop(ω_A, τ_1)
estop(ω_A, τ_3)
estop(ω_A, τ_2)
estop(ω_A, τ_3)
estop(ω_A, τ_2)
esleep(ω_A, τ_3)
esleep(ω_C, τ_1)
esleep(ω_A, τ_3)
estop(ω_A, τ_1)
estop(ω_A, τ_1)

Fig. 3: Example of patterns and the global trace generated by the composition of resource models defined in the Example 4

generated. This trace assumes the critical instant theorem [15] (to find the worst execution trace) as well as the hyper-period of the resource model (to define the length of the trace). Replacing the event $\mathsf{estop}(\omega, \tau_1)$ with $\mathsf{estop}(RS-C, ts1)$ in Equation 1 we are able to obtain the maximum detection delay of our trace, which corresponds to value γ equal to 34 time units. We also known that the deadline of 27 time units for monitor period is not enough for the established resource models settings, RS-A and RS-C, respectively. However, if we increase the period of the monitor task to a value greater than 39 time units we obtain a schedulable taskset.

7 Performance Evaluation of our MTL-$\int$ Approach

To estimate the performance of our evaluation algorithm we define some classical properties in MTL-$\int$ and monitor them using our evaluation algorithm, such as: $true\ U_{\leq t}\ \phi$ (eventually); $\phi \rightarrow \Box_{\leq t}\, \psi$ (bounded-invariance); $\phi \rightarrow \Diamond_{\leq t}\, \psi$ (bounded-response); $\Box_{\leq t} \int^t \phi \leq \beta$ (limited-duration); and $\phi \rightarrow \int^t \psi \leq \beta$ (bounded-duration). Results of the performance analysis are depicted in Table 3. The values are presented in milliseconds. Average values are computed over multiple runs provided by a stochastic model. The length of the input trace is denoted by $|\rho|$. The entry $t_{average}$ is the execution time that a set of monitors takes, on average, to the evaluation algorithm. The throughput shows how many events can be processed by the monitor as the trace increases, i.e., $\frac{|\rho|}{t_{monitor}}$.

Monitors	$|\rho|$				throughput
	10	100	1000	10000	
$true\ U_{\leq t}\ \phi$	0.051	1.717	171.376	26366.48	196.1, 0.379
$\phi \to \Box_{\leq t}\,\psi$	0.065	1.834	172.683	26159.36	153.8, 0.382
$\phi \to \Diamond_{\leq t}\,\psi$	0.055	1.765	174.594	26944.16	181.8, 0.371
$\Box_{\leq t} \int^t \phi \leq \beta$	0.309	65.950	76682.652	$>$ 10min	indef
$\phi \to \int^t \psi \leq \beta$	0.011	0.195	14.033	1993.12	909.1, 5.01
$t_{average}$	0.098	14.192	15443.068	indef	no value

Table 3: Performance analysis of enforcement monitors (milliseconds)

The experiments were performed on an Intel Core i3-3110M at 2.40GHz CPU, and 8 GB RAM running on Fedora 18 X64. Note that all the monitors are time bounded in t, indicating that only one trace that has this duration should assign a truth value for the formula (a verdict). Our algorithm executes in polynomial time as the experiments also show.

8 Conclusion and Further Work

In this paper we have introduced a novel approach to runtime monitoring. Compared with currently available methods our approach extends runtime monitoring for component-based approaches; introduces the monitor synthesis for duration formulas; establishes guarantees such as time interference of monitors, predictability, and avoidance of catastrophic scenarios due to WCET violations for our compositional framework; and supplies a platform for design of hard real-time embedded system where knowledge provided at execution time is required. In terms of current and future research goals, we are presently working on formally establishing the correctness of the algorithm as well as several properties of our MTL-$\int$ fragment. Our preliminary empirical findings strongly suggest that our fragment is well suited in terms of expressiveness for schedulability analysis of uni-processor and multi-processor systems.

Acknowledgments. The authors would like to thank the anonymous reviewers for their detailed and helpful comments. This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects Ref. FCOMP-01-0124-FEDER-022701 (CISTER), FCOMP-01-0124-FEDER-015006 (VIPCORE) and FCOMP-01-0124-FEDER-020486 (AVIACC).

References

1. R. Alur and T. A. Henzinger. Logics and Models of Real Time: A Survey. In *Real-Time: Theory in Practice, REX Workshop*, pages 74–106, 1992.

2. M. C. B. Alves, D. Drusinsky, J. B. Michael, and M. Shing. Formal validation and verification of space flight software using statechart-assertions and runtime execution monitoring. SOSE'11, pages 155 –160, 2011.
3. M. Auguston and M. Trakhtenbrot. Synthesis of Monitors for Real-Time Analysis of Reactive Systems. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 72–86. 2008.
4. A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.
5. G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks. UPPAAL 4.0. QEST '06, pages 125–126, 2006.
6. S. Cotard, S. Faucou, J. Bechennec, A. Queudet, and Y. Trinquet. A Data Flow Monitoring Service Based on Runtime Verification for AUTOSAR. HPCC '12, pages 1508–1515, 2012.
7. André de Matos Pedro, David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto. Logic-based Schedulability Analysis for Compositional Hard Real-Time Embedded Systems. In *Proceedings of the 6th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, CRTS '13, 2013.
8. D. Drusinsky. The Temporal Rover and the ATG Rover. SPIN Workshop, pages 323–330, 2000.
9. A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A Compositional Scheduling Framework for Digital Avionics Systems. RTCSA '09, pages 371–380, 2009.
10. E. Fersman, P. Krcal, P. Pettersson, and W. Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.
11. E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.
12. E. Fersman, P. Pettersson, and W. Yi. Timed Automata with Asynchronous Processes: Schedulability and Decidability. TACAS '02, pages 67–82, 2002.
13. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
14. Y. Lakhneche and J. Hooman. Metric temporal logic with durations. *Theor. Comput. Sci.*, 138(1):169–199, 1995.
15. C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.
16. D. Ničković and N. Piterman. From MTL to deterministic timed automata. FORMATS'10, pages 152–167, 2010.
17. L. Pike, A. Goodloe, R. Morisset, and S. Niller. Copilot: a hard real-time runtime monitor. RV'10, pages 345–359, 2010.
18. L. Pike, S. Niller, and N. Wegmann. Runtime verification for ultra-critical systems. RV'11, pages 310–324, 2012.
19. L. Pike, N. Wegmann, S. Niller, and A. Goodloe. Copilot: Monitoring Embedded Systems. *Innovations in Systems and Software Engineering*, pages 1–21, 2013.
20. S. Pinisetty, Y. Falcone, T. Jéron, H. Marchand, A. Rollet, and O. Nguena Timo. Runtime enforcement of timed properties. RV'13, pages 229–244. 2013.
21. A. Pnueli. The temporal logic of programs. SFCS '77, pages 46–57, 1977.
22. I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, 2008.
23. The OCaml Development Team. Ocaml programming language, 2013.
24. H. Zhu, M. B. Dwyer, and S. Goddard. Predictable Runtime Monitoring. ECRTS '09, pages 173 –183, 2009.