Verifying end-to-end real-time constraints on multi-periodic models

Julien Forget, Frédéric Boniol, Claire Pagetti

Verifying end-to-end real-time constraints on multi-periodic models

Julien Forget[1], Frédéric Boniol[2] and Claire Pagetti[2]
[1] Univ. Lille, France, julien.forget@univ-lille1.fr
[2] ONERA-Toulouse, firstname.lastname@onera.fr

Abstract

Control-command systems must usually satisfy a set of high-level end-to-end timing constraints to ensure their correctness. We propose a formal approach to verify these properties directly at the model level. First, we introduce a small language for specifying arbitrary end-to-end constraints. Then, we show how to verify any constraint of this language for a system represented with a multi-periodic synchronous model, a model that retains the main concepts of data-flow oriented programming languages (such as Matlab/Simulink, synchronous languages or AADL). One advantage of this approach is that it is simpler to verify end-to-end constraints at the model level, early in the development process, rather than at the implementation level.

1 Introduction

Control-command applications, such as avionics embedded flight control systems for instance, are subject to real-time constraints: not only do they need to compute the correct values, they also need to compute these values at the correct time. Such a system is usually designed as a set of periodic tasks; over the last four decades, the real-time community has developed many scheduling algorithms and analysis techniques to ensure that, at system execution, each instance of a task completes before its deadline. In this paper, we focus on a different type of real-time analysis, *end-to-end analysis*, which analyses the time required for performing a chain of computations, from input acquisition to output production.

Control-command systems development typically starts by specifying a high-level model of the system, such as a Matlab/Simulink model for instance. The model is then translated into lower-level code, typically C, either manually or using code generation tools. In our approach, we perform the end-to-end analysis at the model level, which simplifies the analysis by abstracting from implementation details. Furthermore, we can detect an end-to-end constraint violation during the model conception and modify the model accordingly, which is far less time-consuming than performing corrections on the final implementation.

Our contribution is two-folds. First, we define a language to specify formally end-to-end constraints. It considers an abstract representation of the system execution, the *temporal behaviour* of the system, which only provides the estimated best and worst-case completion date of each task instance and the data-dependencies between task instances. This representation completely abstracts from the actual scheduler. Using the constraints language, we provide formal definitions of common end-to-end constraints, such as best and worst-case latencies for instance. This language enables us to devise a general verification technique (see below) capable of checking any constraint of the language, instead of devising a new verification technique for each new type of constraint.

Second, we propose a technique for verifying constraints of this language for a system modelled using a *synchronous multi-periodic model*. In this model, a system consists of a set of tasks instantiated periodically and related by causal data-dependencies (i.e. the consumer of some data cannot start be-

fore the producer completes). The verification technique abstracts from actual execution dates, by reasoning on the abstract temporal behaviour derived from the system model. We only assume that the system is schedulable, i.e. that execution will respect task deadlines and task dependencies. Schedulability analysis is out of the scope of this paper, see for instance [1]. The model was designed so that it should be simple to extract such a representation from systems programmed with data-flow oriented programming languages such as SIMULINK, AADL [2] or the multi-rate synchronous language PRELUDE [3]. The complexity of verifying a constraint of is linear in the number of tasks of the system multiplied by the least common multiple of the periods of the tasks.

The paper is organized as follows. Related works are detailed in Section 2. In order to better understand the key notions on which our analyses relies, we present an informal study of end-to-end constraints in Section 3. The temporal behaviour model and the end-to-end constraints language are introduced in Section 4. The multi-periodic synchronous model is defined in Section 5 and the verification of end-to-end constraints for this model is detailed in Section 6.

2 Related works

First, we clarify the difference between end-to-end analysis and other real-time analyses. Worst-case Execution Time (WCET) analysis [4] calculates an upper bound to the execution time of each task of a system. This is required to perform a schedulability analysis and guarantee that every task will meet its deadline [1, 5]. Worst-case Response-Time (WCRT) analysis [1, 6] computes an upper bound to the time required for a task to complete its execution, when executed concurrently with the rest of the tasks of the system. It is a popular technique for checking system schedulability, Demand Bound Function analysis [1] being another one. In these analyses, real-time properties usually concern a single task (even though their analysis requires to consider the whole system). By contrast, end-to-end constraints concern several tasks involved in a chain of computation. In [7, 8], authors propose WCRT analyses for groups of tasks, called *transactions*, however all tasks in a transaction must have the same period. Worst-Case Traversal Time (WCTT) analysis [9] calculates an upper bound to the time required for some input data to reach its destination in a network of components. Real-time calculus [10] derives schedulability analysis techniques from network calculus [9].

Several models have been proposed to accurately specify the timing behaviour and the timing requirements of a real-time system at the model level. This includes the Timing Architecture Description Language (TADL) [11], adopted by AUTOSAR and EAST-ADL, and the Clock Constraint Specification Language (CCSL) [12], designed for UML-MARTE. General verification techniques have been proposed for CCSL [12] (based on model checking) and for TADL [13] (based on timed automata), but the complexity is quite high.

End-to-end properties analysis at the model level has been studied in [14–16]. These methods, targeted for the automotive context, assume *register buffer communication*, where the producer task writes in the buffer when it completes and the consumer task reads from it when it starts, while we consider causal communications. Furthermore, properties analysis is tightly related to the underlying scheduling policy. In [17], authors proposed an end-to-end properties analysis that is performed at the model level, which abstracts from scheduling details, but still with register buffer communication. End-to-end latency analysis for systems with causal communications has been studied in [18–21]. Our work provides a more general analysis method for arbitrary end-to-end properties (not only latency).

Real-time parameter synthesis, has a different objective, which is to compute parameters that will enforce the satisfaction of some imposed end-to-end constraints. Task period synthesis is presented in [22, 23], task WCET synthesis in [24] and job-level dependencies synthesis in [20].

3 Informal study

We illustrate the notion of end-to-end constraints on a subset of the longitudinal flight control system (Fig-

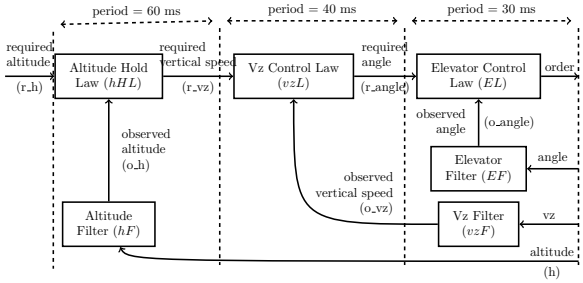


Figure 1: Vertical speed control from ROSACE

ure 1) extracted from the ROSACE case-study [25]. This system controls the angle of the control surface (*order*) based on: the current surface angle (*angle*), the current aircraft vertical speed (vz), the current aircraft altitude (*altitude*) and the altitude required by the pilot (*required altitude*). The software architecture consists of six tasks that make up three computation chains. Tasks operate at three different rates (30ms, 40ms, 60ms) and some tasks of different rates communicate. The system must satisfy end-to-end real-time constraints detailed below.

Latency constraint Let us consider the navigation computation chain (r_h, hHL, vzL, EL, *order*). To ensure a smooth reaction of the aircraft to pilot commands, the time elapsed between a pilot order (*required altitude*) and the modification of the control surface angle (*order*) is required to be less than 1s. Assuming that the control surface requires 400ms to reach an ordered angle, the end-to-end latency of the computation chain must thus be less than 600ms.

A possible temporal behaviour of this chain is detailed in Figure 2. Arrows stand for backwards data-dependencies, for instance the fifth surface angle order $order^5$ depends on the first pilot order r_h^1. The worst-case latency of a pair of dependent task instances ($order^p$,r_h^q) is when r_h^q is executed at the beginning of its period and $order^p$ is executed at the end of its period. Then, we consider the worst-case latency for each input paired with the first output that depends on it (e.g. we ignore the latency of ($order^6$,r_h^1)). The behaviour described in this diagram satisfies the 600ms latency

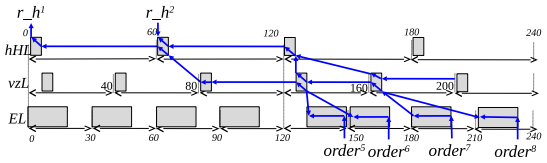


Figure 2: Chain from r_h to *order*

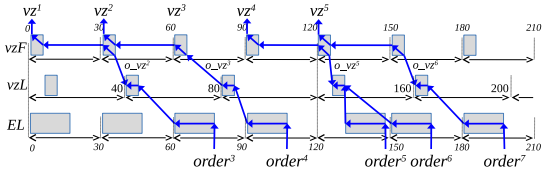


Figure 3: Chain from vz to *order*

constraint: the worst-case latency is 150ms, achieved for ($order^5$,r_h^1) and ($order^7$,r_h^2).

Reactivity Let us consider the chain (vz, vzF, vzL, EL, *order*). A gust of wind may suddenly increase the vertical speed of the aircraft (vz). To preserve the aircraft structure integrity, and for passengers comfort, the control system must react to any variation of vz that lasts longer than 120ms, by reorienting the control surfaces accordingly (by computing a new value for *order*).

Let us assume that tasks EL and vzL incur no delay, other than their execution time, but that vzF incurs a logical delay of one cycle, meaning that the vzF input at its repetition n only impacts its output at its repetition $n+1$. A possible temporal behaviour for this chain is described in Figure 3. In the worst-case scenario, vz^3 is received at date 60, it impacts the output of vzF^4 produced after date 90 (due to the internal delay of vzF), thus after the execution of vzL^3. Finally, the output of vzF^5 overwrites the output of vzF^4 at date 120. As a result, vz^3 impacts no instance of the output *order*. This means that a variation of vz that lasts for up to 60ms may have no impact on *order*. Thus the reactivity of the chain is no less than 60ms.

4 End-to-end properties specification

In this section we propose a small language dedicated to the specification of end-to-end real-time constraints and use this language to define some classic end-to-end constraints. Constraints are formulated based on the system *temporal behaviour*, an abstract representation of the system execution that abstracts from the underlying real-time task model and scheduler.

4.1 Temporal behaviour

The informal study presented in the previous section emphasizes the information we need to perform end-to-end analysis: 1) the completion date of each instance of each task; 2) the data-dependencies relating task instances. We capture this information in the *temporal behaviour* of a system. In the rest of the paper we denote $\mathbb{N}^*$ the set of strictly positive integer (i.e. zero excluded). A temporal behaviour is defined as a pair $\{\mathcal{J}, \mathcal{D}\}$, where $\mathcal{J}$ is the set of jobs and $\mathcal{D}$ is the set of job dependencies. Each job τ_i^p corresponds to the instantiation of some task τ_i (with $i, p \in \mathbb{N}^*$). Since the exact duration of a job execution is usually not known, but can only be bounded with best-case execution times (BCET) and worst-case execution times (WCET), our end-to-end analysis relies on bounds on completion times instead of exact values. For each job τ_i^p, the temporal behaviour of a system provides its earliest completion time, denoted $etime(\tau_i^p)$, and its latest completion time, denoted $ltime(\tau_i^p)$. We assume that we always have $ltime(\tau_i^p) \leq etime(\tau_i^{p+1})$. The computation of $etime()$ and $ltime()$ not only depends on jobs BCET/WCET but also on the system schedule. However, the temporal behaviour model merely describes a system execution, but does not consider how it was produced. In particular, it makes no assumptions on how jobs are generated (periods, release dates), and on whether jobs respect some deadlines or not. Completion dates computation will be detailed in Sections 5 and 6.

We denote $\tau_i^p \leftarrow \tau_j^q$ when $(\tau_i^p, \tau_j^q) \in \mathcal{D}$, which represents the fact that τ_j^q produces data required for computing τ_i^p. By extension, we write $\tau_i \leftarrow \tau_j$ when there exists some p, q, such that $\tau_i^p \leftarrow \tau_j^q$. Each job has a set of predecessors $in(\tau_i^p) = \{\tau_j^q \in \mathcal{J} | \tau_i^p \leftarrow \tau_j^q\}$ and a set of successors $out(\tau_i^p) = \{\tau_j^q \in \mathcal{J} | \tau_j^q \leftarrow \tau_i^p\}$. A job that has no predecessors is an input job or *sensor job*. A job that has no successor is an output job or *actuator job*. We denote $\leftarrow^*$ the transitive closure of the dependence relation. A task τ_i may introduce internal logical delays, meaning that a job of this task produces its output based on the input received by an older job of the task. This is specified using function del_i. The function definition is out of the scope of this paper, but it can be obtained by a static analysis of the task internal data-dependencies. The semantics is as follows:

Definition 1. *If* $\tau_j^q \leftarrow \tau_i^p$ *and* $\tau_k^r \leftarrow \tau_j^{q+del_j(\tau_i,\tau_k)}$, *then* $\tau_k^r \leftarrow^* \tau_i^p$.

By extension, we write $\tau_i \leftarrow^* \tau_j$ when there exists some p, q, such that $\tau_i^p \leftarrow^* \tau_j^q$. End-to-end properties are imposed on a chain of computation or *functional chain*, which relates a sensor to an actuator through a series of dependencies.

Definition 2. *A functional chain is a sequence of tasks* $(\tau_1, \ldots, \tau_n)$, *where* τ_1 *is a sensor,* τ_n *is an actuator, and for all* $1 < i \leq n$, $\tau_i \leftarrow \tau_{i-1}$.

In the example of Figure 3, $order^5$ is a job of $order$. We have $order^5 \leftarrow EL^5$, $EL^5 \leftarrow vzL^4$, $vzL^4 \leftarrow vzF^5$ and $vZF^4 \leftarrow vz^4$. Since $del_{vzF}(vz, vzL) = 1$, we have $vZL^4 \leftarrow^* vz^4$. By transitivity, we also have $order^5 \leftarrow^* vz^4$ and $(vz, vzF, vzL, EL, order)$ is a functional chain.

4.2 End-to-end constraints language

Now, we define a language for specifying the end-to-end constraints we want to verify on some temporal behaviour. The language focuses on: relevant inputs ($rlv(x_S)$), i.e. inputs that are used by some output, the first and last successors of a relevant input ($first(idx)$, $last(idx)$), the earliest/latest completion date of a job ($date$), the duration that separates input and output completion dates ($duration$) and the

constraint on a duration ($formula$). Term $task^{idx}$ corresponds to some element of the job set. We denote by $\mathcal{E}$ the set of end-to-end properties, i.e. the set of terms corresponding to rule $formula$.

$$
\begin{array}{lll}
x_S & ::= & x \mid x-1 \mid x+1 \mid k \in \mathbb{N} \\
idx & ::= & rlv(x_S) \mid idx-1 \mid idx+1 \\
 & & \mid last(idx) \mid first(idx) \\
date & ::= & etime(task^{idx}) \mid ltime(task^{idx}) \\
duration & ::= & date - date \mid \min_{x\in\mathbb{N}}(date-date) \\
 & & \mid \max_{x\in\mathbb{N}}(date-date) \\
formula & ::= & duration <= k \in \mathbb{N}
\end{array}
$$

Definition 3. *Let $\tau' \leftarrow^* \tau$ and $q \in \mathbb{N}^*$:*

- *$\tau^{rlv(q)}$ is the q-th* relevant value *of τ (with respect to τ') iff: $\exists, p \in \mathbb{N}^* | \tau'^p \leftarrow^* \tau^{rlv(q)}$;*
- *$rlv(q) < rlv(q+1)$ (rlv is a strictly increasing function);*
- *$last(rlv(q)) = max\{p | \tau'^p \leftarrow^* \tau^{rlv(q)}\}$;*
- *$first(rlv(q)) = min\{p | \tau'^p \leftarrow^* \tau^{rlv(q)}\}$.*

For instance, in the example of Figure 3, for the dependence $vzL \leftarrow vzF$, we have $rlv(1) = 2$, meaning that vzF^1 is unused (irrelevant). Then $rlv(2) = 3$ and $rlv(3) = 5$, meaning that vzF^4 is unused. For the dependence $EL \leftarrow vzL$, $first(4) = 5$ and $last(4) = 6$, because vzL^4 is first used by EL^5 and last used by EL^6. Let us now consider the latency constraint for the first input of chain $(r\_h, ..., order)$ in the example of Figure 2. The first relevant value of r_h is produced by $r\_h^{rlv(1)}$ (i.e. r_h^1) and the instances that consume it are instances $order^p$ with $p \in [first(rlv(1)), last(rlv(1))]$ (i.e. $order^5$, $order^6$). Latency is computed considering the first consumer, thus we consider the time elapsed for dependency $order^{first(rlv(1))} \leftarrow^* r\_h^{rlv(1)}$. The worst-case corresponds to a situation where $r\_h^{rlv(1)}$ is sampled as early as possible (date 0), while the first consumer $order^{first(rlv(1))}$ is produced as late as possible (date 150). Using our language, the latency constraint is expressed as: $ltime(order^{first(rlv(1))}) - etime(r\_h^{rlv(1)}) \leq 600$.

When $rlv(1) > 1$, some instances of the consumer task depend on a constant initial value, that is not computed by the producer task. This is for instance the case in Figure 3 for EL^1 in dependency $EL \leftarrow vzL$. We complete the previous definitions as follows to handle the initial value.

Definition 4. *We define:*

$$rlv(0) = 0 \quad first(0) = 1 \quad last(0) = first(rlv(1)) - 1$$
$$etime(\tau_i^0) = ltime(\tau_i^0) = 0$$

4.3 Classic end-to-end properties

We can now formulate several classic end-to-end properties, such as those of [14], using the specification language we just introduced.

Worst-case latency The latency of a functional chain can be defined informally as the time needed for an input value to propagate to an output value (also called *first to first* delay in [14]). To define the worst-case latency (WCL) of a chain, we need to consider all of its pairs of dependent values. However, we must also take into account input values that are consumed by no output: in the example of Figure 3, if the value of vz changes during its third sampling period (i.e. for vz^3) and provided it remains unchanged until the fourth sampling period (i.e. for vz^4), then *order* will be impacted only at its fifth period (i.e. for $order^5$). This leads us to the definition below. Notice that an input value is unused only in case $rlv(x-1)+1 \neq rlv(x)$.

Definition 5. *Let $\tau_n \leftarrow^* \tau_1$. $WCL(\tau_1, \tau_n) =$*

$$\max_{x\in\mathbb{N}^*}(ltime(\tau_n^{first(rlv(x))}) - etime(\tau_1^{rlv(x-1)+1}))$$

Best-case latency Consider the chain $(r\_h, ..., order)$, the first relevant input value is r_h^1 and the first occurrence of *order* that depends on it is $order^5$ (i.e., $rlv(1) = 1$ and $first(rlv(1))) = 5$). The best-case latency of this pair is achieved when r_h^1 is read at the latest, i.e. just before 60, and $order^5$ is produced at the earliest, i.e. just after 120. More formally: $BCL(r\_h^1, order^5) = etime(order^{first(rlv(1))}) - ltime(r\_h^{rlv(1)})$. Thus:

Definition 6. *Let* $\tau_n \leftarrow^* \tau_1$. $BCL(\tau_1, \tau_n) =$

$$max(0, \min_{x \in \mathbb{N}}(etime(\tau_n^{first(rlv(x))}) - ltime(\tau_1^{rlv(x)})))$$

Best-case and worst-case freshness Let $\tau_n \leftarrow^* \tau_1$. At any time t, the freshness is the difference $t - t_1$, where t_1 is the reading date of the value of τ_1 used to compute the current value of τ_n (also called *last to last* delay in [14]). Consequently, a freshness constraint $Freshness(\tau_1, \tau_n) \leq \Delta_f$ requires that, at every time t, the value of τ_1 used to compute the current value of τ_n has been read no earlier than $t - \Delta_f$. Let us consider the chain $(vz, ..., order)$ depicted Figure 3. We have $order^3 \leftarrow vz^1$. Considering that vz^1 is acquired at $t \in [0, 30)$ and that $order^3$ is produced at $t' \in [60, 90)$, then the freshness of $order^3$, that is $t' - t$, is bounded as follows: $60 - 30 < t' - t < 90 - 0$.

The best-case freshness (BCF) of the chain is obtained for $order^5$, which depends on vz^4. The best-case freshness of $order^5$ is $120 - 120 = 0$, which is achieved when vz^4 is read at the latest (just before 120), and $order^5$ is produced at the earliest (just after 120). For the worst-case freshness (WCF), we note that $order^6$ depends on the same input value as $order^5$. This means that the input is not refreshed for $order^6$. The worst-case freshness of *order* with respect to vz^4 is thus $180 - 90 = 90$, i.e. the latest completion time of the last occurrence of *order* that depends on vz^4 minus the earliest completion time of vz^4. So, we have: $bcf(order, vz^4) = etime(order^5) - ltime(vz^4)$ and $wcf(order, vz^4) = ltime(order^6) - etime(vz^4)$.

Formal definitions are provided below. Let us emphasize that, even though freshness and latency are different notions, the definitions of BCL and BCF coincide. However, WCL and WCF do not.

Definition 7. *Let* $\tau_n \leftarrow^* \tau_1$. *The worst-case (resp. best-case) freshness between* τ_n *and* τ_1 *is defined as:*

$$WCF(\tau_1, \tau_n) = \max_{x \in \mathbb{N}}(ltime(\tau_n^{last(rlv(x))}) - etime(\tau_1^{rlv(x)}))$$

$$BCF(\tau_1, \tau_n) = max(0, \min_{x \in \mathbb{N}}(etime(\tau_n^{first(rlv(x))}) - ltime(\tau_1^{rlv(x)})))$$

Worst-case reactivity The reactivity of a chain between two tasks τ_1 and τ_n characterizes the minimal duration of any value change on τ_1 such that this change is eventually propagated to τ_n (this property has no equivalent in [14]). For instance, let us consider the chain $(vz, ..., order)$. Figure 3, shows that vz^3 does not impact any occurrence of *order*. Let us consider the following scenario: (1) vz^2 is read at the earliest, i.e. at 30; (2) the value of vz changes just after 30 but then gets back to its previous value just before 120; (3) vz^4 is read at the latest, i.e. at 120. In this scenario, the variation on the value of vz goes "unnoticed" by the system, because vz^3 is not a relevant occurrence of vz.

Definition 8. *Let* $\tau_n \leftarrow^* \tau_1$. $WCR(\tau_n, \tau_1) =$

$$\max_{x \in \mathbb{N}}(ltime((\tau_1^{rlv(x+1)})) - etime(\tau_1^{rlv(x)}))$$

5 Multi-periodic synchronous model

In this section, we define a *multi-periodic synchronous model* to represent the system whose temporal behaviour we want to analyze. We also detail how to obtain the temporal behaviour corresponding to a system modelled that way.

5.1 Definition

A multi-periodic synchronous model is a pair $(\mathcal{T}, \mathcal{M})$. Each $\tau_i \in \mathcal{T}$ (with $1 \leq i \leq |\mathcal{T}|$) is a task instantiated periodically with period $T_i \in \mathbb{N}^*$ (see Section 5.2 for the impact of periods on temporal behaviours). τ_i^n denotes the n^{th} job of τ_i (with $n \in \mathbb{N}^*$). Each $M_{i,j} \in \mathcal{M}$ (with $1 \leq i \leq |\mathcal{T}|$, $1 \leq j \leq |\mathcal{T}|$) is a *dependence matrix*, which provides a specification of all the pairs (p, q) such that $\tau_i^p \leftarrow \tau_j^q$. Dependence matrices, based on [26], represent dependencies that follow patterns repeated periodically. Most industrial applications and design tools that support multi-rate systems, such as Simulink, AADL or Prelude, use such periodic patterns. The reader is referred to [27] for a comparison with other multi-rate data-dependencies models. Let $\mathcal{I}_n$ denote the set of nat-

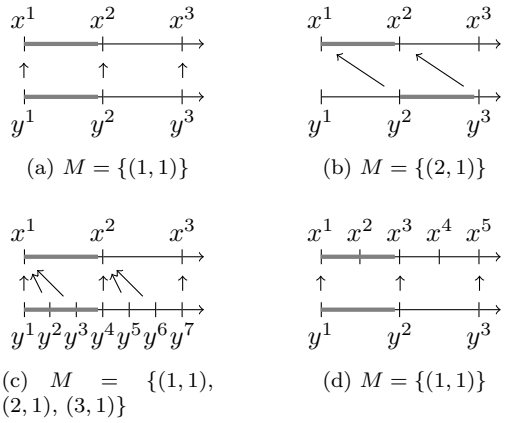
ural integers strictly smaller than n. Let $lcm(n, n')$ denote the least common multiple of n and n'.

Definition 9. *Let τ_i and τ_j be two tasks. Let $H = lcm(T_i, T_j)$. A dependence matrix M for τ_i, τ_j, is such that $M \subseteq \mathcal{I}_{H/T_i} \times \mathcal{I}_{H/T_j}$, where for all $(p,q),(p',q') \in M^2$, $p = p' \Rightarrow q = q'$ and $p' > p \Rightarrow q' \geq q$. The term $\tau_i \overset{M}{\leftarrow} \tau_j$ denotes a dependence relation defined as follows:*

$$\forall (p,q) \in M^\omega, \tau_i^p \leftarrow \tau_j^q$$

$$\text{with } M^\omega \equiv \{(n,n') | \begin{array}{l} \exists k \in \mathbb{N}, (m,m') \in M, \\ (n,n') = (m,m') + (k\frac{H}{T_i}, k\frac{H}{T_j}) \end{array} \}$$

This definition is illustrated in Figure 4. Intuitively, M lists all the pairs of dependent values during the time interval corresponding to the first *hyperperiod* (H in previous definition). This interval is depicted with a thick gray line in the figure. The pattern is then repeated indefinitely at each hyperperiod. A simple precedence (Figure 4a) corresponds to the case where $T_y = T_x$ and $M = \{(1,1)\}$. The pattern consists only of $y^1 \leftarrow x^1$, which is first repeated as $y^2 \leftarrow x^2$, then as $y^3 \leftarrow x^3$, ... So we have $y^p \leftarrow x^p$ for all $p \in \mathbb{N}$. Figure 4b corresponds to a delayed communication: the pattern $y^2 \leftarrow x^1$ is repeated as $y^3 \leftarrow x^2$, then as $y^4 \leftarrow x^3$, ... Here y^1 depends on no value of x, instead it depends on some constant initial value. Notice also that in this example there is an offset between the intervals covered by M respectively for x ($[0, T_x)$) and for y ($[T_x, 2T_x)$). Figure 4c corresponds to an over-sampling, where each value of x is used three times. The pattern is $y^1 \leftarrow x^1$, $y^2 \leftarrow x^1$, $y^3 \leftarrow x^1$, it is repeated as: $y^{1+1*3=4} \leftarrow x^{1+1*1=2}$, $y^{2+1*3=5} \leftarrow x^{1+1*1=2}$, $y^{3+1*3=6} \leftarrow x^{1+1*1=2}$, ... Figure 4d corresponds to an under-sampling, where only one out of two successive values of x are used. The pattern is $y^1 \leftarrow x^1$, it is repeated as: $y^{1+1*1=2} \leftarrow x^{1+1*2=3}$, $y^{1+2*1=3} \leftarrow x^{1+2*2=5}$, ...



(a) $M = \{(1,1)\}$ (b) $M = \{(2,1)\}$

(c) $M = \{(1,1), (2,1), (3,1)\}$ (d) $M = \{(1,1)\}$

Figure 4: Some classic communication patterns ($y \overset{M}{\leftarrow} x$)

5.2 Completion dates

Task periods and task dependencies imply timing constraints. For a given system execution, let $start(\tau_i^p)$ denote the date at which τ_i^p starts and $end(\tau_i^p)$ denote the date at which it completes. Following the synchronous model, we make the following assumptions, which must be respected by the final system implementation:

- *Relaxed synchronous hypothesis* (or *implicit deadlines model*): as in the relaxed synchronous approach studied in [3,28] and in the classic real-time task model [1], we require each instance of a task to complete before the next activation of this task: $start(\tau_i^p) \geq T_i * (p-1)$ and $end(\tau_i^p) < T_i * p$;
- *Causality*: a task instance cannot start executing before all its predecessors complete their execution: $\forall \tau_i^p \leftarrow \tau_j^q, start(\tau_i^p) \geq end(\tau_j^q)$.

In the following, we assume that the final system implementation respects these timing constraints, which is typically ensured by the system compiler (when automated code-generation is used) and by the system scheduling policy. We abstract from actual execution dates and from execution times, and only consider best and worst-cases, we let $etime(\tau_i^p) = T_i * (p-1)$ and $ltime(\tau_i^p) = T_i * p$. This leads to a safe over-approximation of end-to-end properties. In a way, this is similar to how synchronous languages [29] abstract from real-time to simplify programming. Furthermore, as shown in [14, 30], the contribution of task periods to end-to-end delays is far more important than that of WCET.

5.3 Functional chains

To compute end-to-end properties (in Section 6), we will need to consider the set of task instances dependencies of the functional chain, that is, for a chain $(\tau, ..., \tau')$, the pairs $(p, q) \in \mathbb{N}^2$ such that $\tau'^p \leftarrow^* \tau^q$. To this intent, we first define a composition operator on precedence matrices, which is reminiscent of the unfolding technique of [27].

Definition 10. *Let M_1, M_2, be dependence matrices. The composition of M_1 with M_2 is denoted $M_1 \circ M_2$. The term $\tau_i \overset{M_1 \circ M_2}{\leftarrow^*} \tau_j$ denotes a dependence relation defined as follows:*

$\forall p, q \in \mathbb{N}^2, \tau_i^p \leftarrow^* \tau_j^q$ *iff* $\exists t \in \mathbb{N} | (p, t) \in M_1^\omega, (t, q) \in M_2^\omega$

Let $|M|$ denote the cardinality of M, i.e. the number of pairs of values in M. Algorithm 1 describes a procedure to compute the composition of two dependence matrices. We use the following auxiliary functions:

- $unfold(M, n, T_o, T_i)$ unfolds precedence matrix M over duration n. For instance, in Figure 4c, $unfold(M, 60, 10, 30) = \{(1,1), (2,1), (3,1), (4,2), (5,2), (6,2)\}$;
- $pat\_size(M, T_o, T_i)$ returns the duration covered by one pattern of M. It is equal to $|M| * T_o$. For instance, in Figure 2, for $(r\_h, ..., order)$, we have $pat\_size(M, 30, 60) = 4 * 30 = 120$ (see Section 5.4 to understand why $|M| = 4$);
- $closure(M_1, M_2)$ takes two precedence matrices M_1,M_2 and returns the precedence matrix consisting of the pairs (p, q) such that there exists t with $(p, t) \in M_1$ and $(t, q) \in M_2$.

In the following, we consider that the composition is computed with Algorithm 1. To compute the dependence matrix of a whole functional chain, we also need to account for internal task delays.

Property 1. *Let $\tau_a \overset{M_1}{\leftarrow^*} \tau_b$ and $\tau_b \overset{M_2}{\leftarrow} \tau_c$. We have $\tau_a \overset{M}{\leftarrow^*} \tau_c$, with:*

$$M = M_1' \circ M_2$$
$$M_1' = \{(p, q) | \exists p', p = p' + del_b(c, a), (p', q) \in M_1\}$$

Algorithm 1 Procedure $\mathcal{C}$ for composing M_1 with M_2

Require: $\tau_i \overset{M_1 \circ M_2}{\leftarrow} \tau_j$
  $hp \leftarrow lcm(pat\_size(M_1, T_i, T_j), pat\_size(M_2, T_i, T_j))$
  $M_1' \leftarrow unfold(M_1, hp, T_i, T_j)$
  $M_2' \leftarrow unfold(M_2, hp, T_i, T_j)$
  return $closure(M_1', M_2')$

Proof. By adding $del_b(c, a)$ to each output index of M_1 while keeping input indices unchanged, we delay each input (the "time" needed for an input to impact an output) by $del_b(c, a)$. □

We can now bound the size of dependence matrices.

Lemma 1. *Let $C = (\tau_1, \ldots, \tau_n)$ a functional chain. We have $\tau_n \overset{M}{\leftarrow^*} \tau_1$, with $|M| \leq lcm_{i \leq n}(T_i)/T_n$.*

Proof. By induction.

Induction base. For a single precedence $\tau_2 \overset{M_1}{\leftarrow} \tau_1$, due to the definition of a dependence matrix, we have $|M_1| \leq lcm(T_1, T_2)/T_2$.

Induction step. Assume $\tau_n \overset{M}{\leftarrow^*} \tau_1$ and $|M| \leq lcm(T_1, ..., T_n)/T_n$. Let $\tau_{n+1} \overset{M_{n+1}}{\leftarrow^*} \tau_n$. Then, we have $\tau_{n+1} \overset{M'}{\leftarrow^*} \tau_1$, with $M' = M_{n+1} \circ \{1 + (del_b(c, a), 1)\} \circ M$. Let $M'' = \{1 + (del_b(c, a), 1)\} \circ M$. We have $|M''| \leq lcm(T_1, ..., T_n)/T_n$. Finally, according to Algorithm 1, $|M'| \leq lcm(T_1, ..., T_n, T_{n+1})/T_{n+1}$. □

5.4 Example

Let us apply the previous notions on the flight control system of section 3.

System model Real-time characteristics are as follows:

$\mathcal{T}$	T	del
hHL	60	$del(r\_h, vzL) = 1$
hF	60	0
vzL	40	$del(hHL, EL) = 1$
		$del(vzF, EL) = 0$
EL	30	0
EF	30	0
vzF	30	$del(vz, vzL) = 1$
r_h	60	0
$order$	30	0

$\mathcal{M}$	Precedence matrix
$hHL \overset{M_1}{\leftarrow} r\_h$	$M_1 = \{(1,1)\}$
$vzL \overset{M_2}{\leftarrow} hHL$	$M_2 = \{(3,2),(4,3),(5,3)\}$
$EL \overset{M_3}{\leftarrow} vzL$	$M_3 = \{(5,4),(6,4),(7,5),(8,5)\}$
$order \overset{M_4}{\leftarrow} EL$	$M_4 = \{(1,1)\}$
$vz \overset{M_6}{\leftarrow} vzF$	$M_6 = \{(1,1)\}$
$vzL \overset{M_7}{\leftarrow} vzF$	$M_7 = \{(2,2),(3,3),(4,5)\}$
$EL \overset{M_8}{\leftarrow} vzL$	$M_8 = \{(3,2),(4,3),(5,4),(6,4)\}$
$order \overset{M_9}{\leftarrow} EL$	$M_9 = \{(1,1)\}$

Precedence matrices are more complex than those shown before. For instance, for $M_2 = \{(3,2),(4,3),(5,3)\}$, for the first pattern, we have $vzL^3 \leftarrow hHL^2$, $vzL^4 \leftarrow hHL^3$, $vzL^5 \leftarrow hHL^3$, then we repeat the pattern with $vzL^6 \leftarrow hHL^4$, $vzL^7 \leftarrow hHL^5$, ...

This example also illustrates the possibility to have an offset between the time intervals covered by one pattern of the precedence matrix for the input and output tasks. For instance, let us consider dependence $vzL \overset{M_2}{\leftarrow} hHL$. The time interval covered by one pattern of M_2 has length 120, but for hHL it corresponds to $[0, 120)$ while for vzL it corresponds to $[80, 200)$.

Functional chain r_h **to** *order* The behaviour of this chain is shown in Figure 2. Even though the hyperperiod of r_h and *order* is 60, the precedence matrix obtained for describing their transitive dependence relation covers an interval of duration 120. This is coherent with Lemma 1 and is due to the period of vzL being equal to 40. For the complete chain, we obtain: $order \overset{M}{\leftarrow}^* r_h$ *with* $M = \{(5,1),(6,1),(7,2),(8,2)\}$

Functional chain vz **to** *order* The behaviour of this chain is shown in Figure 3. Since $del_{vzL}(vzF, EL) = 0$, vzL induces no delay on this computation chain, even though it did for the previous chain. Compared to the previous example, here some values are unused. For instance, $M_7 = \{(2,2),(3,3),(4,5)\}$, so the sequence of values of vzF consumed by vzL is 2,3,5,... Thus vzF^4 is unused. However, since vzF introduces a delay, when we consider the whole chain the input that is unused is actually vz^3. For the complete chain, we obtain: $order \overset{M}{\leftarrow}^* vz$ *with* $M = \{(3,1),(4,2),(5,4),(6,4)\}$

End-to-end properties. The properties that must be checked are specified as follows, using the definitions provided in Section 4.3: $WCL(r\_h, order) \leq 600$, $WCR(vz, order) \leq 120$.

6 Verification

In this section, we detail how to verify end-to-end constraints expressed with the constraints language of Section 4.2, for a system specified with the synchronous model of Section 5, based on its temporal behaviour.

6.1 Computing term date

We first detail how to compute terms of the *date* rule of the constraints language. To this intent, we define an equivalence relation on elements of a dependence matrix that will simplify the computation. For any set S, let $|S|$ denote its cardinality.

Definition 11. *Let M be a dependence matrix. We define an equivalence relation $\sim$ on M and a partitioning of M into subsets $[M]_i$ as follows:*

- $\forall (p,q),(p',q') \in M^2, (p,q) \sim (p',q') \equiv q = q'$
- *$M/\sim$ is the quotient of M by $\sim$ (its set of equivalence classes);*
- $\forall i, 1 \leq i < |(M/\sim)|, \forall (p,q),(p',q') \in M^2$*:*

$$[M]_i \in (M/\sim)$$
$$(p,q) \in [M]_i \wedge (p',q') \in [M]_{i+1} \Rightarrow q < q'$$

Additionally:

- *Let $in([M]_i)$ be such that for all $(p,q) \in [M]_i$, $q = in([M]_i)$;*
- *Let $outs([M]_i)$ denote the first projection of $[M]_i$.*

For instance, for the precedence matrix $M = \{(3,1),(4,2),(5,4),(6,4)\}$ (from $order \leftarrow^* vz$), we have $[M]_1 = \{(3,1)\}$, $[M]_2 = \{(4,2)\}$, $[M]_3 = \{(5,4),(6,4)\}$. We also have $in([M]_3) = 4$ and $outs([M]_3) = \{5,6\}$.

We note that relevant inputs in the case of precedence matrices correspond to the successive values of $in([M]_i)$:

Property 2. *Let $\tau_n \overset{M}{\leftarrow}^* \tau_1$: $\forall x \in \mathbb{N}^*, rlv(x) = in([M^\omega]_x)$*

Proof. Deduced from dependence matrices definition. □

Then, we can compute last and first successors as follows:

Property 3. *Let* $\tau_n \overset{M}{\leftarrow}^* \tau_1$, $\forall q \in \mathbb{N}^*$: $\begin{cases} last(rlv(q)) = max(outs([M]_q)) \\ first(rlv(q)) = min(outs([M]_q)) \end{cases}$

Proof. Deduced from dependence matrices definition. □

Based on these properties and on the definition of $etime()$ and $ltime()$ in Section 5.2, we can compute the value of terms of the language rule *date*.

6.2 Computing term formula

To verify that an end-to-end property holds, we need to compute terms of the *date* rule of the language for *every pair* of dependent values of the concerned tasks. We will show that formula values follow a pattern repeated periodically, due to the fact that we are restricted to periodic systems with periodic communication patterns. As a consequence, we only need to check the validity of the formula for every pair of the pattern. First, we note some auxiliary properties:

Property 4. *Let* $\tau_n \overset{M}{\leftarrow}^* \tau_1$, $H = lcm(T_n, T_1)$. *For all* $x \in \mathbb{N}$*:*

$$rlv(x + H/T_1) = rlv(x) + H/T_1$$
$$last(x + H/T_1) = last(x) + H/T_n$$
$$first(x + H/T_1) = first(x) + H/T_n$$

Proof. Due to the definition of M^ω. □

Now we show that *duration* values (i.e. *date*−*date*) for increasing values of x follow a periodic pattern.

Property 5. *Let* $\tau_n \overset{M}{\leftarrow}^* \tau_1$ *and* $H = lcm(T_n, T_1)$. *Let* $f(x) = date_1(x) - date_2(x)$*, where* $date_1(x)$ *and* $date_2(x)$ *are terms corresponding to the syntactic rule* date *of our language. We have:*

$$\forall x > 0, f(x) = f(x + H/T_1)$$

Proof. As an example, let us consider the worst-case latency. We have $WCL(x) = \max_{x \in \mathbb{N}^*}(date_1(x) - date_2(x))$ with $date_1(x) = ltime(\tau_n^{first(rlv(x))})$ and $date_2(x) = etime(\tau_1^{rlv(x-1)+1})$. Then:

$$\begin{aligned} WCL(x + H/T_1) =& first(rlv(x + H/T_1)) \times T_n - \\ & (rlv(x + H/T_1 - 1) + 1) \times T_1 \\ =& first(rlv(x)) \times T_n - (rlv(x-1) + 1) \times T_1 + \\ & H/T_n \times T_n - H/T_1 \times T_1 = WCL(x) \end{aligned}$$

The same proof principles hold for any other pair of terms $date_1(x)$ and $date_2(x)$. □

As a consequence of this property, to check a formula of our language we only need to check the formula for each relevant input in one pattern of $|M|$. Therefore, we can state the complexity of a formula verification as follows.

Theorem 1. *Let* $(\tau_1, \ldots, \tau_n)$ *be a functional chain where data-dependencies are specified by dependence matrices. Any formula of* $\mathcal{E}$ *for this functional chain can be verified with complexity* $\mathcal{O}(lcm_{k \leq n}(T_k) \times n)$.

Proof. There are two steps in the verification: 1) Compute the dependence matrix M for the whole chain; 2) Check the formula for every relevant input of one pattern of M.

For step 1, we compute M by recursive applications of Algorithm 1. The complexity of this Algorithm is linear in size of the resulting matrix. Due to Lemma 1, the size of this matrix is lower than $lcm\{T_i, 1 \leq i \leq n\}$. So step 1 is linear in $lcm\{T_i, 1 \leq i \leq n\} * n$.

For step 2, we note that any term of rule *date* requires to compute a value of $rlv(x)$ and either $first(rlv(x))$ or $last(rlv(x))$. Any formula requires to compute the value of two terms of *date* for all the relevant inputs of M. Since obviously we can compute the set of all relevant inputs, first and last successors of one pattern of M by a single traversal of M, the complexity of step 2 is linear in the size of M. □

Note that this means that our verification has a time-complexity linear in the size of the hyper-period of the tasks of the functional chain (Lemma 1). The

hyper-period is in the worst-case exponential with respect to the number of tasks of the chain [1]. However, this rarely occurs in practice, since it requires all tasks to have co-prime periods. Furthermore, this complexity seems inevitable, since it is hard to think of an analysis that would not *at least* consider every pair of dependent values in the hyper-period.

6.3 Implementation

The verification procedure we just presented has been implemented in OCaml (about 300 lines of code). We do not present detailed experiments since the procedure has linear time-complexity and thus scales really well. For instance, it takes less than 10ms on an Intel Core i7-2.8GHz, with 4Gb of RAM, to check a property on a chain of 1000 tasks with an hyperperiod of 10000. Verifying the properties of our case study takes less than 1ms.

7 Conclusion

In this paper we have studied the formal verification of end-to-end real-time constraints at the model level. We first proposed a language for specifying formally end-to-end constraints. Then, we detailed how to check properties of this language on a multi-periodic synchronous model.

We plan to extend this work to support more complex end-to-end properties. For instance, we wish to check that two functional chains have similar latencies (when two actuators are related to the same sensor). We also plan to consider the synthesis problem: how to choose task periods and communications so as to satisfy a set of specified end-to-end constraints.

References

[1] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*, 4th ed. Addison-Wesley Educational Publishers Inc, 2009.

[2] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): an introduction," Carnegie Mellon University, Tech. Rep., 2006.

[3] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens, "Multi-task implementation of multi-periodic synchronous programs," *Discrete Event Dynamic Systems*, vol. 21, no. 3, pp. 307–338, 2011.

[4] R. Kirner and P. P. Puschner, "Classification of WCET analysis techniques," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, Seattle, USA, 2005.

[5] V. Bertin, E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine, "Taxys=esterel+kronos. a tool for verifying real-time properties of embedded systems," in *40th IEEE Conference on Decision and Control*, vol. 3, 2001.

[6] M. Kuo, R. Sinha, and P. Roop, "Efficient wcrt analysis of synchronous programs using reachability," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11, 2011.

[7] K. Tindell, "Adding time-offsets to schedulability analysis," Department of Computer Science, University of York, Tech. Rep., 1994.

[8] J. C. Palencia and M. G. Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *20th IEEE Real-Time Systems Symposium*, Washington, DC, USA, 1999.

[9] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[10] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of the IEEE 2000 International Symposium on Circuits and Systems (ISCAS)*, vol. 4, 2000.

[11] *Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2*, Aug 2012.

[12] C. André and F. Mallet, "Combining CCSL and Esterel to specify and verify time requirements," in *Proceedings of Conf. on Languages, Compilers, and Tools for Embedded Systems*, Dublin, Ireland, 2009.

[13] A. Goknil, J. Suryadevara, M.-A. Peraldi-Frati, and F. Mallet, "Analysis support for tadl2 timing constraints on east-adl models," in *European Conference on Software Architecture*, Montpellier, France, 2013.

[14] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different paths semantics," in *Proceedings of the IEEE Real-Time System Symposium Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Barcelona, Spain, 2008.

[15] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ecu networks: Formal modeling and precise estimation," in *Proceedings of the Conference on Embedded Software (EMSOFT'10)*, Scottsdale, USA, 2010.

[16] S. Mohalik, D. B. Chokshi, M. G. Dixit, A. C. Rajeev, and S. Ramesh, "Scalable model-checking for precise end-to-end latency computation," in *2013 IEEE International Symposium on Computer-Aided Control System Design (CACSD)*, Hyderabad, India, Aug. 2013.

[17] S. Mubeen, M. Sjödin, T. Nolte, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, "End-to-end timing analysis of black-box models in legacy vehicular distributed embedded systems," in *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015.

[18] M. Geilen, S. Tripakis, and M. Wiggers, "The earlier the better: A theory of timed actor interfaces," in *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, 2011.

[19] R. Wyss, F. Boniol, C. Pagetti, and J. Forget, "End-to-end latency computation in a multi-periodic design," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC)*, 2013.

[20] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *The 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2016.

[21] J. Khatib, A. Munier-Kordon, E. C. Klikpo, and K. Trabelsi-Colibet, "Computing latency of a real-time system modeled by synchronous dataflow graph," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016.

[22] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing end-to-end timing constraints by calibrating intermediate processes," in *Real-Time Systems Symposium, Proceedings*, Dec 1994.

[23] J. Li, M. Xiong, V. Lee, L. Shu, and G. Li, "Workload-efficient deadline and period assignment for maintaining temporal consistency under edf," *Computers, IEEE Transactions on*, vol. 62, no. 6, pp. 1255–1268, 2013.

[24] E. Wozniak, M. Di Natale, H. Zeng, C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard, "Assigning time budgets to component functions in the design of time-critical automotive systems," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14, 2014.

[25] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron, "The ROSACE case study: From

simulink specification to multi/many-core execution," in *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[26] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, "Scheduling Dependent Periodic Tasks Without Synchronization Mechanisms," in *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Stockholm, Sweden, 2010.

[27] J. Forget, E. Grolleau, C. Pagetti, and P. Richard, "Dynamic Priority Scheduling of Periodic Tasks with Extended Precedences," in *IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA)*, Toulouse, France, Sep. 2011.

[28] A. Curic, "Implementing Lustre programs on distributed platforms with real-time constraints," Ph.D. dissertation, Univ. Joseph Fourier, Grenoble, 2005.

[29] A. Benveniste and G. Berry, "The synchronous approach to reactive and real-time systems," in *Readings in hardware/software co-design.* Kluwer Academic Publishers, 2001.

[30] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems*, vol. 10, no. 1, January 2013.