Static timing analysis of cyber-physical systems with relaxed real-time constraints

Guillaume Roumage, Cyril Faure, Selma Azaiez, Stephane Louise

Static Timing Analysis of Cyber-Physical Systems with Relaxed Real-Time Constraints

Guillaume Roumage, Selma Azaiez, Cyril Faure, Stéphane Louise
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
firstname.lastname@cea.fr

***Abstract*—Modern Cyber-Physical Systems (CPS) often involve dependent real-time processes bound to different timing constraints. These CPSs can be modeled using PolyGraph, a formalism that extends the Cyclo-Static Dataflow model with explicit real-time constraints. Unlike traditional real-time approaches where timing constraints are specified for every process, PolyGraph can offer greater flexibility by binding real-time constraints for only a subset of processes (e.g., processes associated with hardware components such as sensors and actuators), along with end-to-end latency specifications. In that case, processes without specified timing constraints only inherit ones from processes with explicit and specified timing constraints due to data dependencies. This paper aims to formalize a method to derive timing constraints for an entire PolyGraph model where only a subset of processes has explicit timing constraints in the model. We demonstrate how this approach eases scheduling constraints and enables early detection of missed deadlines through two examples from a vehicle ADAS and the Ingenuity Mars helicopter.**

***Index Terms*—Static Timing Analysis, Dataflow Model, System-Level Verification, Real-Time Systems**

I. Introduction

Cyber-Physical Systems (CPSs) are reactive systems that detect environmental shifts through sensors, process this information using computational processes, and then use the output to control actuators. CPSs range from digital signal processing systems to centralized/distributed systems, embedded/cloud infrastructures, soft/hard real-time systems, and even a mix of all the above. These complex systems must operate reliably without threatening their internal processes or the safety of their users. For example, a failure of the actuator in an autonomous car can lead to catastrophic consequences such as a car crash or a pedestrian accident.

To ensure the safety of such systems, CPS designers can use analysis tools based on deterministic Dataflow Models of Computation (DF MoCs) [1]. These tools can formally prove the correctness of CPSs regarding safety and temporal properties, such as consistency, liveness, latency, and throughput. In particular, some DF MoCs can ensure such guarantees on CPSs with real-time constraints [2], [3]. In this paper, we analyze CPS models with the PolyGraph formalism [2], a DF MoC that expands the Cyclo-Static Dataflow (CSDF) model [4] with explicit real-time constraints for a subset of processes. Specifically, we go one step further by analyzing how the explicit timing constraints in the CPS's model are propagated across the entire model, especially to processes without explicit timing constraints in the model. Within the scope of the paper, the processes are called *actors*. In other words, our research question is: "Within a CPS model with real-time constraints on a subset of actors, how do these actors influence the timing behavior of the non-explicitly real-time ones?"

The PolyGraph model allows designers to avoid imposing real-time constraints on actors of the CPS model that do not naturally endorse such constraints. This capability expands the search space of static analysis algorithms and the optimization potential for system execution. Actors with non-explicit real-time constraints still inherit temporal limitations from their real-time counterparts through their runtime dependencies.

1) Contribution and Scope of the Paper: Based on our observations, we have developed an approach to improve the laxity of timing constraints (release time, earliest finish time, latest start time, deadline) for an entire dataflow model of a CPS when only a subset of actors have real-time constraints in the model specification. Our approach uses the PolyGraph formalism to analyze the model of a CPS and assumes that the Best-Case Execution Time (BCET) and Worst-Case Execution Time (WCET) of the actors are provided. Challenges related to system schedulability and scheduling, accurate computation of BCETs and WCETs, and deployment, execution, and monitoring of the system on typical CPS hardware, such as distributed heterogeneous MP-SoCs[1], are orthogonal to the problem addressed here.

2) Paper Organization: The paper starts by presenting the background and the terminology of the PolyGraph model in Section II. Our approach is presented in Section III. We present our experimentation and discussion in Section IV and related works in Section V. Section VI concludes the paper and presents our future works.

II. Background and Terminology

Our contribution is based on the PolyGraph model [2], a superset of the SDF model [5]. We present the SDF model, and then explain how the PolyGraph model expands it.

A. The SDF Model

An SDF model is an oriented graph $G = (V, C)$ where V is a finite set of *actors* and C is a finite set of *channels*. An SDF model is characterized by a *topology matrix* Γ_G of size $|C| \times |V|$. The entry (i, j) of Γ_G is the number of tokens produced or consumed by the actor v_j on the channel c_i each time the actor v_j is executed (this number is positive if the tokens are produced, and negative otherwise). An execution of an actor is called a *job*.

[1]Multi or Many-Core System(s) on Chip.

An actor $v \in V$ has (optional) input and output ports connected to input and output channels of C. A channel $c_i = (v_j, v_k, n_{ij}, n_{ik}, init_i) \in C$ connects an output port of the actor $v_j \in V$ to an input port of the actor $v_k \in V$. This channel also has a production rate $n_{ij} \in \mathbb{N}^*$ (which is the entry (i, j) of Γ_G), a consumption rate $n_{ik} \in \mathbb{N}^*$ (the entry (i, k) of Γ_G), and a number of initial tokens $init_i \in \mathbb{N}$. An actor's job produces/consumes tokens to/from the buffer of its output/input channels according to the production/consumption rate. For the sake of simplicity in the rest of this paper, we denote $[c_i]$ the number of initial tokens of the channel c_i.

B. The PolyGraph Model

1) Brief Syntax and Semantic: The PolyGraph formalism is a superset of the SDF formalism and expands it in two ways: the production and consumption rates become periodic sequences (as in CSDF [4]), and a subset of actors may have timing constraints. The terminology of the static analysis of an SDF model, i.e., consistency, liveness, consistent and live execution, and hyperperiod [5], is extended to a PolyGraph model. Consistency ensures the system can be executed within a bounded memory, while liveness guarantees deadlock-free execution. A hyperperiod is a partially ordered set of actors' jobs that returns the system to its initial state.

a) Rational production and consumption rates: Let $P = (V, C)$ be a PolyGraph model and let $\Gamma_P = (\gamma_{ij}) \in \mathbb{Q}^{|C|\times|V|}$ be its topology matrix. The production and consumption rates are rational[2]: a channel $c_i \in C$ is a tuple $(v_j, v_k, \gamma_{ij}, \gamma_{ik}, init_i)$ such that $\gamma_{ij} \in \mathbb{Q}^*$ and $\gamma_{ik} \in \mathbb{Q}^*$, and $init_i \in \mathbb{Q}$ ($\mathbb{Q}^* = \mathbb{Q} \setminus \{0\}$). Rational rates imply that the number of tokens produced/consumed can differ for each job. Specifically, a token is produced and stored in a channel if and only if sufficient fractional token parts are symbolically produced. Indeed, only an integer number of tokens may be produced or consumed. In other words, an actor with a rational production rate such as $1/n$ in the dataflow model actually produces data once every n jobs during runtime, thus validating the precedence constraint toward the consumer once every n of its jobs. These peculiarities create non-trivial cases when timing constraints (releases and deadlines) of actors without real-time specification in the model are made explicit, as we will see in our contribution.

The rational production and consumption rates are natural consequences of the periodicity of real-time actors (with specified and imposed frequencies) together with the consistency of the system's model imposed by the topology matrix. The rational rates are a compact notation for a CSDF equivalency [4].

b) Timing constraints: The actors of a PolyGraph model may have a frequency constraint. Furthermore, they may also have a phase (which usually models the system's end-to-end latency) if they have a frequency constraint. The frequency dictates the actor's execution at the specified rate, while the phase postpones its initial execution. Actors with frequency constraints are referred to as *timed actors*. An execution of a consistent and live PolyGraph model is an infinite repetition of its first hyperperiod. Although the number of jobs is not bounded during an execution, their timing constraints are cyclic over a hyperperiod.

2) Visual Example: The system in Fig. 1 has three actors and two channels. An actor's job increases/decreases the number of tokens in its output/input channels according to the production/consumption rate. Production and consumption rates are rational, so the number of tokens in the channels is also rational. Table I illustrates the evolution of the tokens inside the system's channels in Fig. 1 for one hyperperiod.

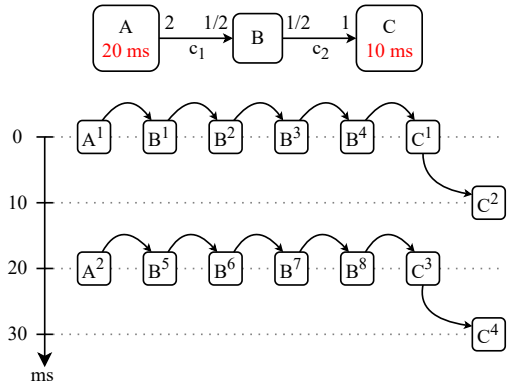

Fig. 1. A PolyGraph model is on the top side, and its execution diagram of the first and second hyperperiods is below. The actor A has a period of 20 ms (i.e., a frequency of 50 Hz), and the actor C has a period of 10 ms (i.e., 100 Hz). Neither A nor C has a phase specified.

TABLE I
THE EVOLUTION OF THE TOKENS INSIDE THE CHANNELS OF THE POLYGRAPH MODEL OF FIG. 1 DURING THE FIRST HYPERPERIOD.

Actor's job		A^1	B^1	B^2	B^3	B^4	C^1	C^2
Tokens in channels	c_1	2	3/2	1	1/2	0	—	—
	c_2	0	1/2	1	3/2	2	1	0

III. APPROACH

A. Overview and Scope of the Approach

We propose an approach to maximize the execution windows (release time and deadline), as well as the earliest finish time and the latest start time, of all the actors of a PolyGraph model at the job level while maintaining the real-time constraints of the system. The job level granularity is essential as the number of tokens produced or consumed may differ for each job, influencing the execution windows of the actors' jobs. We assume that the shortest execution time of an actor, called the Best-Case Execution Time (BCET), and the longest execution time of an actor, called the Worst-Case Execution Time (WCET), are provided [6][3]. We also assume that the sensors and actuators of the PolyGraph model are timed actors

[2]A channel of a PolyGraph model as defined in [2] has at least one integer rate. However, for the sake of simplicity, we consider in this paper that both rates are rational.

[3]The assumption of available BCET and WCET values implies that we have a prior allocation of jobs on the execution cores of any MP-SoC.

(they have a frequency constraint), the communication time between processes should be considered as integrated into the BCETs and WCETs, and the number of times an actor can be preempted is bounded (the additional execution time induced by the preemptions is included in the BCET and WCET).

Fig. 2 places our work in the workflow of a system design process. We will discuss in section IV how the results of our approach can be used to improve scheduling, monitoring, and sizing methods. Note that an actual deployment and execution of the model on hardware platforms brings additional challenges, e.g., the scheduling of real-time systems [7], [8] and the computation of BCETs and WCETs [6]. Those challenges fall out of the scope of this paper.

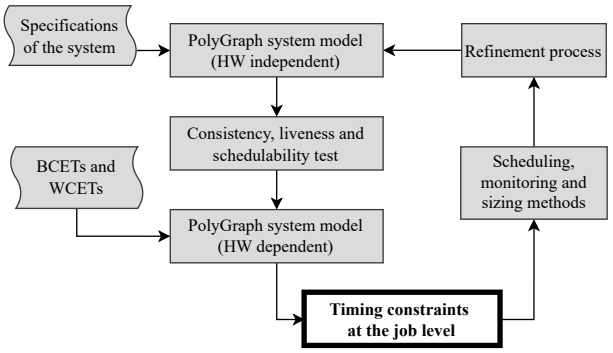

Fig. 2. The workflow of a system design process. Our contribution is the derivation of the timing constraints at the job level from a hardware dependent PolyGraph system model. HW stands for hardware.

B. Preliminary Definitions

We assume the static analysis we want to perform is applied to consistent and live PolyGraph models extended with BCET and WCET on their actors. We refer to such PolyGraph models as *well-defined PolyGraph models*.

Definition 1 (Well-defined PolyGraph model):

A well-defined PolyGraph model is a weakly connected[4], consistent and live graph. In addition, actors without predecessors/successors (i.e., sensors/actuators) are timed actors, meaning they have a frequency constraint. All actors within the model have both a BCET and a WCET, such that the BCET is less than or equal to the WCET.

The remainder of the paper is based on arithmetic relations between the *index* of tokens and the *instance* of an actor's job.

Definition 2 (Token index and job instance):

Let $P = (V, C)$ be a well-defined PolyGraph model, and let σ be a consistent and live execution of P. The *index* of a token identifies the tokens transiting through a channel $c \in C$ along σ. The *instance* of an actor's job $v \in V$ is the number of jobs of v that have been executed along σ. In other words, the n-th job of v is the job of instance n of v.

C. A PolyGraph Execution Model

A PolyGraph execution model is a well-defined PolyGraph model along with the release time, the earliest finish time, the latest start time, and the deadline computed with Theorem 1 and Corollary 3. The static timing analysis we proposed is performed on a well-defined PolyGraph model with an additional requirement: no actor may have all its precedence constraints met before runtime. In other words, all actors must have at least one input channel with a number of initial tokens below its consumption rate[5].

Before delving into our contribution, we remind that for a given channel c_i, its number of initial tokens is denoted $[c_i]$.

Lemma 1 (Proposition 1 of [9]):

Let $P = (V, C)$ be a well-defined PolyGraph model, let $v_j, v_k \in V$ be two actors of P, and let $c_i \in C$ be a channel of P from v_j to v_k. We define $\Gamma_P = (\gamma_{ij}) \in \mathbb{Q}^{|C|\times|V|}$ to be the topology matrix of P, and $r_i = [c_i] - \lfloor [c_i] \rfloor$ with $\lfloor [c_i] \rfloor$ being the floor function applied to $[c_i]$. Let σ be a consistent and live execution of finite length of P and let $\mathbf{x}$ be a vector such that the component $\mathbf{x}_i$ is the number of jobs of the actor v_i executed along σ. Then, the following holds:

(i) The number of tokens produced by the actor v_j to the channel c_i along σ is $\lfloor \mathbf{x}_j \cdot \gamma_{ij} + r_i \rfloor$.

(ii) The number of tokens consumed by the actor v_k from the channel c_i along σ is $\lceil \mathbf{x}_k \cdot |\gamma_{ik}| - r_i \rceil$.

Corollary 1:

Under the assumptions of Lemma 1, the following holds:

(i) The total number of tokens produced to the channel c_i after the n-th job of the actor v_j is $\lfloor n \cdot \gamma_{ij} + r_i \rfloor$.

(ii) The total number of tokens consumed from the channel c_i after the p-th job of the actor v_k is $\lceil p \cdot |\gamma_{ik}| - r_i \rceil$.

Lemma 2:

Under the assumptions of Lemma 1, also assuming that the initial tokens of c_i (if any) have an index from 1 to $\lfloor [c_i] \rfloor$, and each subsequent token increments its index by one, then the following holds:

(i) The job of instance n of the actor v_j produces to the channel c_i the tokens from index $\lfloor (n-1) \cdot \gamma_{ij} + [c_i] + 1 \rfloor$ to $\lfloor n \cdot \gamma_{ij} + [c_i] \rfloor$, or no token at all if $\lfloor (n-1) \cdot \gamma_{ij} + [c_i] \rfloor = \lfloor n \cdot \gamma_{ij} + [c_i] \rfloor$. Moreover, the first token produced by or after the job of instance n of v_j has index $\lfloor (n-1) \cdot \gamma_{ij} + [c_i] + 1 \rfloor$.

(ii) The job of instance p of the actor v_k consumes from the channel c_i the tokens from index $\lceil (p-1) \cdot |\gamma_{ik}| - r_i \rceil + 1$ to $\lceil p \cdot |\gamma_{ik}| - r_i \rceil$, or no token at all if $\lceil (p-1) \cdot |\gamma_{ik}| - r_i \rceil = \lceil p \cdot |\gamma_{ik}| - r_i \rceil$. Moreover, the last token consumed by or before the job of instance p of v_k has index $\lceil p \cdot |\gamma_{ik}| - r_i \rceil$.

Proof of assertion (i) of Lemma 2:

Let us first define $tokens_{prod}(n, i, j) = \lfloor n \cdot \gamma_{ij} + r_i \rfloor$. From Corollary 1, the actor v_j has produced a total of $tokens_{prod}(n-1, i, j)$ tokens after its job of instance $n-1$ and a total of $tokens_{prod}(n, i, j)$ tokens after its job of instance n. Furthermore, the token with index $tokens_{prod}(n-1, i, j)$ is produced by the job of instance $n-1$ of v_j. Therefore, the job of instance n of v_j produces the tokens with index from $tokens_{prod}(n-1, i, j)+1$ to $tokens_{prod}(n, i, j)$. Adding $\lfloor [c_i] \rfloor$ to both interval boundaries permits us to consider the initial tokens of the channel c_i: the initial tokens of c_i have an index from 1 to $\lfloor [c_i] \rfloor$.

[4]A directed graph is weakly connected when its undirected induced graph is connected [7].

[5]In this case, a pre-processing step may perform "offline executions" prior to runtime until this requirement is met.

Since $r_i = [c_i] - \lfloor [c_i] \rfloor$ and $\forall x, y \in \mathbb{N} : \lfloor x \rfloor + \lfloor y \rfloor = \lfloor x+y \rfloor$, we can rewrite $tokens_{prod}(n-1,i,j)+1+\lfloor [c_i] \rfloor$ as $\lfloor (n-1) \cdot \gamma_{ij} + [c_i] + 1 \rfloor$. The same logic holds for $tokens_{prod}(n,i,j) + \lfloor [c_i] \rfloor$, which is equal to $\lfloor n \cdot \gamma_{ij} + [c_i] \rfloor$.

If $\lfloor (n-1) \cdot \gamma_{ij} + [c_i] \rfloor = \lfloor n \cdot \gamma_{ij} + [c_i] \rfloor$, then no tokens are produced by the job of instance n of v_j since the token of index $\lfloor (n-1) \cdot \gamma_{ij} + [c_i] \rfloor$ has already been produced. Finally, the index of the first token produced by or after the job of instance n of v_j is equal to $tokens_{prod}(n-1,i,j)+1+\lfloor [c_i] \rfloor$ (which is the left boundary of the interval of tokens produced by the n-th job of v_j, if any). ∎

Proof of assertion (ii) of Lemma 2:

Let us first define $tokens_{cons}(p,i,k) = \lceil p \cdot |\gamma_{ik}| - r_i \rceil$. From Corollary 1, the actor v_k has consumed a total of $tokens_{cons}(p,i,k)$ tokens after its job of instance p and a total of $tokens_{cons}(p-1,i,k)$ after its job of instance $p-1$. Furthermore, the token with index $tokens_{cons}(p-1,i,k)$ is consumed by the job of instance $p-1$ of v_k. Therefore, the job of instance p of v_k consumes the tokens with index from $tokens_{cons}(p-1,i,k)+1$ to $tokens_{cons}(p,i,k)$.

If $\lceil (p-1) \cdot |\gamma_{ik}| - r_i \rceil = \lceil p \cdot |\gamma_{ik}| - r_i \rceil$, then no tokens are consumed by the job of instance p of v_k since the token of index $tokens_{cons}(p-1,i,k)$ has already been consumed. Finally, the index of the last token consumed by or before the job of instance p of v_k is equal to $tokens_{cons}(p,i,k)$ (which is the right boundary of the interval of tokens consumed by the p-th job of v_k, if any). ∎

Lemma 3 (Proposition 2 of [9]):

Under the assumptions of Lemma 1, the following holds:

(i) The token of index m that transit through the channel c_i along σ is consumed by the job of instance $1 + \lfloor \frac{m-1+r_i}{|\gamma_{ik}|} \rfloor$ of the actor v_k.

(ii) The token of index m that transit through the channel c_i along σ is produced by the job of instance $\lceil \frac{m-[c_i]}{\gamma_{ij}} \rceil$ of the actor v_j.

Lemma 4:

Under the assumptions of Lemma 1, the following holds:

(i) The tokens produced by the job of instance n of the actor v_j are consumed by the job of instance $1 + \lfloor \frac{\lfloor (n-1) \cdot \gamma_{ij} + [c_i] \rfloor + r_i}{|\gamma_{ik}|} \rfloor$ to $1 + \lfloor \frac{\lfloor n \cdot \gamma_{ij} + [c_i] \rfloor - 1 + r_i}{|\gamma_{ik}|} \rfloor$ of the actor v_k.

(ii) The tokens consumed by the job of instance p of the actor v_k are produced by the job of instance $\lceil \frac{\lceil (p-1) \cdot |\gamma_{ik}| - r_i \rceil + 1 - [c_i]}{\gamma_{ij}} \rceil$ to $\lceil \frac{\lceil p \cdot |\gamma_{ik}| - r_i \rceil - [c_i]}{\gamma_{ij}} \rceil$ of the actor v_j.

Proof of Lemma 4:

The proof is a composition of Lemma 2 and Lemma 3. ∎

Corollary 2:

Under the assumptions of Lemma 1, the following holds:

(i) The first token produced by the job of instance n of the actor v_j is consumed by the job of instance $1 + \lfloor \frac{\lfloor (n-1) \cdot \gamma_{ij} + [c_i] \rfloor + r_i}{|\gamma_{ik}|} \rfloor$ of the actor v_k.

(ii) The last token consumed by the job of instance p of the actor v_k is produced by the job of instance $\lceil \frac{\lceil p \cdot |\gamma_{ik}| - r_i \rceil - [c_i]}{\gamma_{ij}} \rceil$ of the actor v_j.

Theorem 1 (Derivation of the execution windows):

Let $P = (V, C)$ be a well-defined PolyGraph model. We define $\texttt{period}_v$ and $\texttt{phase}_v$ to be the period and phase of an actor $v \in V$ if v is a timed actor. Let $bcet_v / wcet_v$ be the BCET/WCET of an actor $v \in V$. Under the assumptions of Lemma 1, the following holds:

(i) Let $v_k \in V$ be a non-timed actor of P. Then, assuming $release^n_{v_j}$ returns the release of the n-th job of an actor $v_j \in V$, the release of the p-th job of v_k is computed as follows:

$$release^p_{v_k} = \max_{\substack{i \in [0,|C|] \\ j \in [0,|V|] \\ \text{st } \gamma_{ij} < 0}} release^{\alpha_1(p,i,j,k)}_{v_j} + bcet_{v_j} + \big(p - \beta_1(p,i,j,k)\big) \cdot bcet_{v_k} \tag{1}$$

with

$$\alpha_1(p,i,j,k) = \lceil \frac{\lceil p \cdot |\gamma_{ik}| - r_i \rceil - [c_i]}{\gamma_{ij}} \rceil$$

and

$$\beta_1(p,i,j,k) = 1 + \Big\lfloor \frac{\lfloor \big(\alpha_1(p,i,j,k) - 1\big) \cdot \gamma_{ij} + [c_i] \rfloor + r_i}{|\gamma_{ik}|} \Big\rfloor$$

(ii) Let $v_k \in V$ be a timed actor of P. Then, by denoting $\overline{release}^p_{v_k}$ the release computed by equation (1), the release of the p-th job of v_k is computed as follows:

$$release^p_{v_k} = \max\Big(\overline{release}^p_{v_k}\ ,\ \texttt{period}_{v_k} \cdot (p-1) + \texttt{phase}_{v_k}\Big) \tag{2}$$

(iii) Let $v_j \in V$ be a non-timed actor of P. Then, assuming $deadline^p_{v_k}$ returns the deadline of the p-th job of an actor $v_k \in V$, the deadline of the n-th job of v_j is computed as follows:

$$deadline^n_{v_j} = \min_{\substack{i \in [0,|C|] \\ k \in [0,|V|] \\ \text{st } \gamma_{ik} > 0}} deadline^{\alpha_2(n,i,j,k)}_{v_k} - wcet_{v_k} - \Big(\beta_2(n,i,j,k) - n\Big) \cdot wcet_{v_j} \tag{3}$$

with

$$\alpha_2(n,i,j,k) = 1 + \Big\lfloor \frac{\lfloor (n-1) \cdot \gamma_{ij} + [c_i] \rfloor + r_i}{|\gamma_{ik}|} \Big\rfloor$$

and

$$\beta_2(n,i,j,k) = \lceil \frac{\lceil \alpha_2(n,i,j,k) \cdot |\gamma_{ik}| - r_i \rceil - [c_i]}{\gamma_{ij}} \rceil$$

(iv) Let $v_j \in V$ be a timed actor of P. Then, by denoting $\overline{deadline}^n_{v_j}$ the deadline computed by equation (3), the deadline of the n-th job of v_j is computed as follows:

$$deadline^n_{v_j} = \min\Big(\overline{deadline}^n_{v_j}\ ,\ \texttt{period}_{v_j} \cdot n + \texttt{phase}_{v_j}\Big) \tag{4}$$

Proof of assertion (i) of Theorem 1:

Let v_k be a non-timed actor of P. The releases of v_k are computed from the ones of its predecessors. Thus, we first prove how the release is computed when v_k has (1) *one* predecessor and then when it has (2) *many* predecessors. If v_k has *no* predecessor, it is a sensor. Thus it is a timed actor, which contradicts our assumption.

1. Let $v_j \in V$ be the unique predecessor of v_k such that $v_j \neq v_k$ and let $c_i \in C$ be the channel from v_j to v_k. Let

p be the job instance of v_k for which we compute the release. Assume that the release of the n-th job of v_j is returned by $release^n_{v_j}$. The release of the p-th job of v_k will be computed in five steps: (1.1) determine the index of the last token consumed by or before the p-th job of v_k, (1.2) determine which job instance of v_j produced that token, (1.3) determine which job instance of v_k consumed that token, (1.4) from that latter job instance, determine the required number of jobs of v_k such that the p-th job of v_k is executed, and (1.5) combine the previous steps.

1.1. From (ii) of Lemma 2, the function $index(p,i,k) = \lceil p \cdot |\gamma_{ik}| - r_i \rceil$ returns the index of the last token consumed by or before the p-th job of v_k.

1.2. From (ii) of Lemma 3, the token with the index $index(p,i,k)$ is produced by the job of instance $\alpha_1(p,i,j,k) = \lceil \frac{index(p,i,k)-[c_i]}{\gamma_{ij}} \rceil$ of v_j. Here, we have $release^p_{v_k} \geq release^{\alpha_1(p,i,j,k)}_{v_j} + bcet_{v_j}$.

1.3. From (i) of Corollary 2, the first token produced by the job of instance $\alpha_1(p,i,j,k)$ of v_j is consumed by the job of instance $\beta_1(p,i,j,k) = 1 + \lfloor \frac{\lfloor(\alpha_1(p,i,j,k)-1)\cdot\gamma_{ij}+[c_i]\rfloor+r_i}{|\gamma_{ik}|} \rfloor$ of v_k.

1.4. As p is the job instance of v_k for which we compute the release, $p - \beta_1(p,i,j,k)$ gives the number of jobs of v_k required until the p-th job of v_k is executed.

1.5. Hence: $release^p_{v_k} = release^{\alpha_1(p,i,j,k)}_{v_j} + bcet_{v_j} + \big(p - \beta_1(p,i,j,k)\big) \cdot bcet_{v_k}$.

2. Stage (1) allows for the computation of the release of the p-th job of v_k, under the assumption that v_k has a unique predecessor. However, if v_k has multiple predecessors, it inherits a release constraint from all its predecessors. Hence, the release becomes the most restrictive one, that is, the maximum: $release^p_{v_k} = \max_{\substack{i\in[0,|C|]\\ j\in[0,|V|]\\ \text{st } \gamma_{ij}<0}} release^{\alpha_1(p,i,j,k)}_{v_j} + bcet_{v_j} + \big(p - \beta_1(p,i,j,k)\big) \cdot bcet_{v_k}$, with $\alpha_1(p,i,j,k)$ and $\beta_1(p,i,j,k)$ defined as in stage (1).

■

Proof of assertion (ii) of Theorem 1: If v_k is a timed actor, it has a frequency constraint. Thus, it also inherits a release from that frequency constraint. By definition of the frequency constraint, the p-th job of v_k must start after $\texttt{period}_{v_k} \cdot (p-1) + \texttt{phase}_{v_k}$. The final release of the p-th job of v_k is the most restrictive release, that is, the maximal, between the inherited release from its predecessors, as computed by the assertion (1) of the current theorem, and the inherited release from the frequency constraint.

■

Proof of assertion (iii) of Theorem 1: The principles follow the same reasoning as the proof of assertion (i) of the current theorem. Let us consider an actor $v_j \in V$ that has a unique successor $v_k \in V$ such that $v_j \neq v_k$. Let n be the job instance of v_j for which we compute the deadline. First, determine the index of the first token produced by or after the n-th job of v_j. Second, determine which job instance of v_k consumed that token. Third, determine which job instance of v_j produced that token. Fourth, from the n-th job of v_j, determine the required number of jobs of v_j such that this token is produced. Finally, combine the previous steps. The proof is then extended to the case when v_j has multiple successors. ■

Proof of assertion (iv) of Theorem 1:

This proof follows the same reasoning as the proof of assertion (ii) of the current theorem. The n-th job of v_j must finish before its deadlines inherited from the successors and the one inherited from the frequency constraint.

■

Corollary 3:

Under the assumptions of Theorem 1, the following holds:

(i) Let $v \in V$ be an actor of P. Then, the earliest finish time (eft) of the n-th job of v is $eft^n_v = release^n_v + bcet_v$ where $release^n_v$ is given by assertion (2) or (1) of Theorem 1, depending on that v is a timed actor or not.

(ii) Let $v \in V$ be an actor of P. Then, the latest start time (lst) of n-th of v is $lst^n_v = deadline^n_v - wcet_v$ where $deadline^n_v$ is given by assertion (3) or (4) of Theorem 1, depending on that v is a timed actor or not.

IV. Application and Discussions

A. The ADAS Use Case

We apply our approach to a PolyGraph model from an Advanced Driver-Assistance System (ADAS) presented in Fig. 3. The ADAS system is an MP-SoC (e.g., R-CAR H3). Its PolyGraph model has 14 actors and only six with specified timing constraints. The construction of the model is detailed in reference [2]. We adjusted both the frequency of the *LDR* actor from 30 to 40 Hz to ensure an integer period value and the production rate of the *OBD* - *SPC* channel accordingly to maintain model consistency and liveness.

We have set the BCET/WCET of all actors at 3 ms/5 ms. Thus, the ADAS model is well-defined.

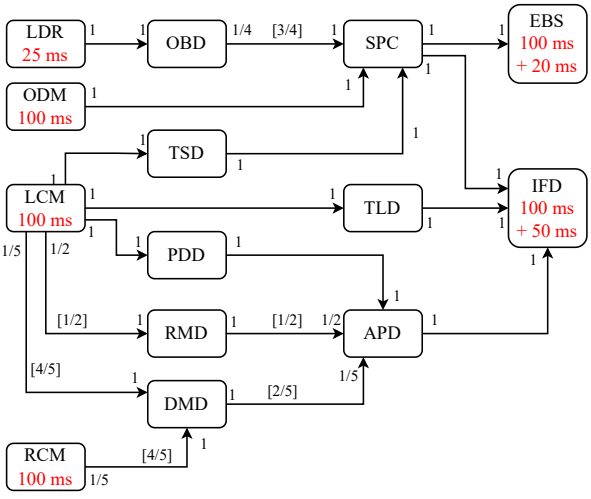

Fig. 3. A PolyGraph model of an ADAS. Actors *EBS* and *IFD* have a phase of 20 ms and 50 ms, respectively.

B. CPSs Sizing

Table II (rows *LDR* to *IFD*) presents the application of our method on the PolyGraph model from the ADAS and shows the release time, earliest finish time, latest start time, and deadline for all actors at the job level. Since the ADAS model is well-defined, the timing analysis of the jobs over the first hyperperiod suffices to determine the timing constraints at the job level for an entire execution.

The length of the jobs' execution windows for actor *OBD* varies from 107 ms to 167 ms (cf. column 6 of Table II). Although the release of all *OBD* jobs remains the same, their deadlines differ. This is primarily due to two reasons. Firstly, *OBD* has no frequency constraint. Therefore, its job's deadlines are inferred from its successor (the actor *SPC*). Secondly, with a production rate of $1/4$, only one job in every four has a precedence constraint with a job of *SPC*. Therefore, as long as the jobs of *OBD* that produce a token are executed early enough to meet the deadlines of *SPC*, the timing constraints of the jobs of *OBD* that do not produce a token are relaxed. This leads to an increased execution window length.

The derivation of execution windows at the job level enables more flexible scheduling constraints. Jobs can be executed at any time within their execution windows and distributed over time, thus decreasing the computing performance requirement of the CPS. Consequently, less powerful CPUs are actually required, leading to a potential reduction in system costs.

Valuable insights can be extracted from the length of the execution windows. In particular, it can help to choose the appropriate algorithm for an actor. This ensures that the actor's execution time remains within the limits of its execution window. For instance, a more precise algorithm may require additional computation time.

C. Optimizing Processor Utilization

Our method computes the largest laxity allowed for every job instance that preserves real-time properties. Our definition of "largest laxity" involves ensuring that the execution window is as long as possible such that no deadlines are missed, assuming a fault-free execution. Our approach minimizes the earliest finish time to maximize the gap between a job's completion and its corresponding deadline, and maximizes the latest start time so that any further delay would result in the possibility of missed deadlines.

For the purpose of processor utilization comparison, we consider in this section an ADAS model specified with a real-time approach. Such a model is the same as the one in Fig. 3 but with timing constraints specified on all actors. The timing constraints on the sensors and actuators are kept the same. We also impose the natural period inherited from the predecessors and successors for all other actors. For instance, as the production and consumption rate on the *LDR* - *OBD* channel is 1, then the period of the *OBD* actor is 25 ms. As the production rate of the channel *OBD* - *SPC* is $1/4$, and the consumption rate is 1, then the period of *SPC* actor is 100 ms. The same reasoning is applied to all other actors: TSD has a period of 100 ms, TLD of 100 ms, PDD of 100 ms, RMD of 200 ms, DMD of 500 ms, and APD of 100 ms.

Our method yields timing constraints that lower the processor utilization of the system. To illustrate, suppose there is only one CPU to execute the ADAS model. The processor utilization of the ADAS model with a strictly periodic real-time approach is 94%, while with the timing constraints derived from our method, the processor utilization is only 72%[6]. Thus, the processor utilization can be reduced by up to 22%. This decrease in processor utilization can be allocated to other tasks by the operating system or contribute to reducing the system's energy consumption.

D. Early Detection of a Runtime Fault: Ingenuity Use Case

Besides CPS sizing and optimizing processor utilization, our method also enables early detection of missed deadlines. We showcase this approach with an application on a simplified PolyGraph model of the Ingenuity Mars helicopter, an aerospace vehicle that completed over 70 flights above the Martian surface from 2021 to 2024.

1) PolyGraph Model of the Vision Processing System of Ingenuity: The vision processing system of the Ingenuity Mars helicopter [10] is modeled in Fig. 4. This system detects visual features on camera frames, and either stores them as reference features (modeled as *pseudo-landmarks*) or tracks them from one frame to another. After a filtering procedure, the tracked visual features are matched with pseudo landmarks, enabling the computation of visual drift to enhance position estimation.

2) Early Detection of Missed Deadlines: In the 6$^{\text{th}}$ flight of Ingenuity, a runtime fault occurred with its vision processing system, resulting in a loss of a camera frame. This fault can be replicated in the model of Fig. 4, with a job of *pseudo-landmarks* taking longer to execute than its WCET, leading to a missed deadline. Since Ingenuity's vision processing operates on a single CPU [10], the job that missed a deadline could be preempted, resulting in an undefined behavior such as loss of camera frame. If we assume the BCETs and WCETs of the actors in Fig. 4 are provided, e.g., 3 ms/5 ms (as for the ADS use case), our methodology allows us to compute the deadlines of the n-th job of *pseudo-landmarks* which is equals to $75 + n \cdot 80$ (cf. row *PL* of Table II). Thus, a runtime verification procedure could check if a job finish before its deadline and engage a fault mitigation procedure otherwise.

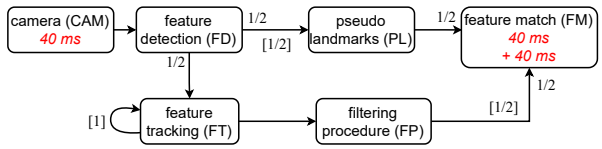

Fig. 4. A PolyGraph model of a simplified vision processing system of the Ingenuity Mars helicopter. Rates of $1/2$ are also a simplified model of the mode-controlled dataflow graph, which can be considered in extensions of PolyGraph . Initial tokens (between brackets) model that the first set of features is pseudo landmarks. Rates of 1 are omitted for clarity.

[6]Note that this processor utilization is a lower bound of the real utilization as we work at the model level. We do not fully consider the number of cores or the scheduling policy. However, it would not be too difficult to derive a linear program to give a tighter evaluation.

TABLE II

TIMING CONSTRAINTS AT THE JOB LEVEL OF THE POLYGRAPH MODEL OF THE ADAS (CF. FIG. 3) AND THE VISION PROCESSING SYSTEM OF INGENUITY (CF. FIG. 4). *EFT* STANDS FOR EARLIEST FINISH TIME, *LST* FOR LATEST START TIME, *Exec. Windows* FOR EXECUTION WINDOWS, *CAM* FOR CAMERA, *FD* FOR FEATURE DETECTION, *PL* FOR PSEUDO LANDMARKS, *FP* FOR FILTERING PROCEDURE, AND *FM* FOR FEATURE MATCH.

	Actor	Timing constraints of the n-th job				
		Release	EFT (= Release + 3)	LST (= Deadline - 5)	Deadline	Exec. Windows
ADAS (Fig. 3)	LDR	$25 \cdot (n-1)$	$3 + 25 \cdot (n-1)$	$-5 + 25 \cdot n$	$25 \cdot n$	25 ms
	OBD	$3 + 25 \cdot (n-1)$	$6 + 25 \cdot (n-1)$	$\begin{cases} 105 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor & \text{if } (n-1) \mod 4 = 0 \\ 190 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor & \text{if } (n-1) \mod 4 = 1 \\ 195 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor & \text{if } (n-1) \mod 4 = 2 \\ 200 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor & \text{if } (n-1) \mod 4 = 3 \end{cases}$	$\begin{cases} 110 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor \\ 195 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor \\ 200 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor \\ 205 + 100 \cdot \lfloor \frac{n-1}{4} \rfloor \end{cases}$	$\begin{cases} 107 \text{ ms} \\ 167 \text{ ms} \\ 147 \text{ ms} \\ 127 \text{ ms} \end{cases}$
	SPC	$6 + 100 \cdot (n-1)$	$9 + 100 \cdot (n-1)$	$10 + 100 \cdot n$	$15 + 100 \cdot n$	109 ms
	EBS	$20 + 100 \cdot (n-1)$	$23 + 100 \cdot (n-1)$	$15 + 100 \cdot n$	$20 + 100 \cdot n$	100 ms
	ODM	$100 \cdot (n-1)$	$3 + 100 \cdot (n-1)$	$-5 + 100 \cdot n$	$100 \cdot n$	100 ms
	TSD	$3 + 100 \cdot (n-1)$	$6 + 100 \cdot (n-1)$	$5 + 100 \cdot n$	$10 + 100 \cdot n$	107 ms
	LCM	$100 \cdot (n-1)$	$3 + 100 \cdot (n-1)$	$-5 + 100 \cdot n$	$100 \cdot n$	100 ms
	PDD	$3 + 100 \cdot (n-1)$	$6 + 100 \cdot (n-1)$	$35 + 100 \cdot n$	$40 + 100 \cdot n$	137 ms
	TDL	$3 + 100 \cdot (n-1)$	$6 + 100 \cdot (n-1)$	$40 + 100 \cdot n$	$45 + 100 \cdot n$	142 ms
	RMD	$3 + 200 \cdot (n-1)$	$6 + 200 \cdot (n-1)$	$35 + 200 \cdot n$	$40 + 200 \cdot n$	237 ms
	DMD	$3 + 500 \cdot (n-1)$	$6 + 500 \cdot (n-1)$	$335 + 500 \cdot (n-1)$	$340 + 500 \cdot (n-1)$	337 ms
	RCM	$100 \cdot (n-1)$	$3 + 100 \cdot (n-1)$	$-5 + 100 \cdot n$	$100 \cdot n$	100 ms
	APD	$6 + 100 \cdot (n-1)$	$9 + 100 \cdot (n-1)$	$40 + 100 \cdot n$	$45 + 100 \cdot n$	139 ms
	IFD	$50 + 100 \cdot (n-1)$	$53 + 100 \cdot (n-1)$	$45 + 100 \cdot n$	$50 + 100 \cdot n$	100 ms
Ingenuity (Fig. 4)	CAM	$40 \cdot (n-1)$	$3 + 40 \cdot (n-1)$	$-5 + 40 \cdot n$	$40 \cdot n$	40 ms
	FD	$3 + 40 \cdot (n-1)$	$6 + 40 \cdot (n-1)$	$\begin{cases} 65 + 80 \cdot \lfloor \frac{n-1}{2} \rfloor & \text{if } (n-1) \mod 2 = 0 \\ 100 + 80 \cdot \lfloor \frac{n-1}{2} \rfloor & \text{if } (n-1) \mod 2 = 1 \end{cases}$	$\begin{cases} 70 + 80 \cdot \lfloor \frac{n-1}{2} \rfloor \\ 105 + 80 \cdot \lfloor \frac{n-1}{2} \rfloor \end{cases}$	$\begin{cases} 67 \text{ ms} \\ 62 \text{ ms} \end{cases}$
	FT	$46 + 80 \cdot (n-1)$	$49 + 80 \cdot (n-1)$	$105 + 80 \cdot n$	$110 + 80 \cdot n$	64 ms
	PL	$6 + 80 \cdot (n-1)$	$9 + 80 \cdot (n-1)$	$70 + 80 \cdot n$	$75 + 80 \cdot n$	69 ms
	FP	$49 + 80 \cdot (n-1)$	$52 + 80 \cdot (n-1)$	$110 + 80 \cdot n$	$115 + 80 \cdot n$	66 ms
	FM	$40 \cdot (n-1)$	$3 + 40 \cdot (n-1)$	$-5 + 40 \cdot n$	$40 \cdot n$	40 ms

E. Other Ways to Use the Proposed Method

1) Improving Scheduling and Energy Consumption: The timing constraints of the PolyGraph execution model can be used further in the system's design process, e.g., to improve the scheduling synthesis and simulation [7], [11]. As a concrete example, the earliest finish time, i.e., the time from which a job can release a contention on a shared resource (if any), can be used to improve scheduling.

Our method also shows potential for enhancing energy efficiency in CPSs. In reference [12], the authors present a technique for reducing power consumption in a CPS modeled with a directed acyclic graph where all actors have real-time constraints. The authors determine the execution windows of the actors and use these windows to minimize power consumption. Our approach permits the application of their findings on CPSs that contain only a select number of processes with real-time constraints.

2) Schedulability Test: Our approach offers a simple way to do a schedulability test. Let $P = (V, C)$ be a PolyGraph model, let V_T be the set of timed actors in P, and let $n\_exec(v)$ be the number of times an actor $v \in V$ is executed within a hyperperiod of P. Suppose that the allocation of processes on the system's cores is statically defined. In that case, the system is schedulable with the Earliest Deadline First scheduling if the value obtained from equation (5) is less than or equal to 1 for each core. It is also easy to proceed to the relevant test for other scheduling policies, such as RMS [8].

$$\sum_{v \in V_T} \frac{wcet_v}{period_v} + \sum_{v \in V \setminus V_T} \frac{1}{n\_exec(v)} \sum_{i=1}^{n\_exec(v)} \frac{wcet_v}{deadline_v^i - release_v^i} \tag{5}$$

V. RELATED WORKS

The reference [13] deals with the deadline assignment problem for systems constrained by an end-to-end deadline. The authors suggest multiple strategies to assign a deadline to the system's actors. The most straightforward strategy (known as *Ultimate Deadline* in [13]) assigns to each actor a deadline equal to the end-to-end deadline of the system, while the most complex one (known as *Equal Flexibility* in [13]) divides the laxity (the difference between the end-to-end deadline and the sum of the WCET of the actors) among the actors proportionally to their execution times and their position in the processing chain. Another relevant reference [14] addresses the deadline assignment problem in dynamic systems where configuration changes frequently, such as real-time distributed databases or video streaming systems. In this case, the system can reject an input flow if the workload it introduces may result in a missed deadline. Our approach differs by considering the sensors as periodic actors whose inputs cannot be dismissed. A third approach to distributing the system's laxity is presented in [15], which involves solving a Mixed Integer Linear Programming problem. These references [13]–[15] all aim to distribute the system's laxity among actors and derive their deadline accordingly, while our work seeks to derive a deadline that results in zero laxity.

The execution windows assignment problem, similar to the one addressed in this paper, is tackled in references [16]–[18]. As in [13]–[15], the references [16], [17] distribute the laxity of the system among the actors. The reference [16] presents two widely used metrics to divide the laxity among the actors: *pure laxity* and *norm laxity*. While the former divides fairly the overall laxity among the actors, the latter gives a laxity

proportional to the execution time of the actors. The work of [17] enhances the reference [16] by introducing an *execution time threshold* and a *virtual execution time* to influence the execution windows length. For instance, the virtual execution time can make a process appear to have a longer execution time than what it actually takes. The authors of [18] approach the execution windows assignment problem differently. In the context of mixed-criticality systems and with the help of integer linear programming, they assign execution windows in a fault-tolerant manner. Critical tasks have execution windows long enough to be re-executed in case of failure without compromising the system's end-to-end deadline.

In the literature, methods have been proposed to automatically derive real-time parameters (such as period, deadlines, and delay) for all actors. One approach, presented in [19], involves deriving a set of constraints with real-time parameters as free variables. Another approach, described in [20], utilizes an algorithm to derive real-time parameters for systems modeled with Homogeneous SDF (HSDF), a less expressive dataflow model than the PolyGraph model.

Our methodology differs from previous approaches, such as those outlined in [13]–[17], in that we do not distribute the system's laxity among its actors. Instead, we adopt the approach presented in the more recent work of [21], which determines the latest start time for each job to ensure that there is no laxity, i.e., the time by which a job can be delayed without missing its deadline. It is worth noting that while the work of [21] allows for event-triggered actors, a PolyGraph model only allows for time-triggered actors.

VI. Conclusion and Future Works

We have presented an offline and automatic method to improve a CPS's laxity (release time, earliest finish time, latest start time, deadline) with real-time constraints on some processes only. These results can be used for multiple purposes, including schedulability tests, enhancing the scheduling synthesis and simulation, and improving runtime monitoring procedures. Our method applies to CPSs modeled with the PolyGraph model. Moreover, as the PolyGraph model is a superset of some DF MoCs, such as SDF and CSDF, our method also applies to CPSs modeled with these DF MoCs.

Our upcoming research will explore the runtime monitoring of the latest start times and deadlines of processes to enable early detection of missed deadlines. More generally, we aim to improve the detection of timing anomalies during the CPS's design phase to enhance the CPS validation and verification process.

References

[1] G. Roumage, S. Azaiez, and S. Louise, "A Survey of Main Dataflow MoCCs for CPS Design and Verification," in *2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. Penang, Malaysia: IEEE, 2023, pp. 1–9.

[2] P. Dubrulle, N. Kosmatov, C. Gaston, and A. Lapitre, "PolyGraph: A Data Flow Model with Frequency Arithmetic," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 3, pp. 489–517, 2021.

[3] X. Khanh Do, S. Louise, and A. Cohen, "Transaction Parameterized Dataflow: A Model for Context-Dependent Streaming Applications," in *2016 Design, Automation & Test in Europe Conference & Exhibition*, 2016, pp. 960–965.

[4] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-Static Dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, 1996.

[5] E. A. Lee and D. Messerschmitt, "Synchronous Data Flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[6] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-Case Execution-Time Problem – Overview of Methods and Survey of Tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008.

[7] A. Honorat, K. Desnos, S. S. Bhattacharyya, and J.-F. Nezan, "Scheduling of Synchronous Dataflow Graphs with Partially Periodic Real-Time Constraints," in *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, 2020, pp. 22–33.

[8] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed., ser. Real-Time Systems Series, 2011.

[9] P. Dubrulle, C. Gaston, N. Kosmatov, and A. Lapitre, "Dynamic Reconfigurations in Frequency Constrained Data Flow," in *Integrated Formal Methods*, vol. 11918, 2019, pp. 175–193.

[10] D. S. Bayard, D. T. Conway, R. Brockers, J. H. Delaune, L. H. Matthies, H. F. Grip, G. B. Merewether, T. L. Brown, and A. M. San Martin, "Vision-Based Navigation for the NASA Mars Helicopter," in *AIAA Scitech 2019 Forum*. American Institute of Aeronautics and Astronautics, 2019.

[11] H. N. Tran, A. Honorat, S. S. Bhattacharyya, J.-P. Talpin, T. Gautier, and L. Besnard, "A Framework for Fixed Priority Periodic Scheduling Synthesis from Synchronous Data-Flow Graphs," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, vol. 13227. Springer International Publishing, 2022, pp. 259–271.

[12] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 156–168.

[13] B. Kao and H. Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time System," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, 1997.

[14] D. Marinca, P. Minet, and L. George, "Analysis of Deadline Assignment Methods in Distributed Real-Time Systems," *Computer Communications*, vol. 27, no. 15, pp. 1412–1423, 2004.

[15] S. Hong, T. Chantem, and X. S. Hu, "Local-Deadline Assignment for Distributed Real-Time Systems," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1983–1997, 2015.

[16] M. Di Natale and J. A. Stankovic, "Dynamic End-to-End Guarantees in Distributed Real Time Systems," in *1994 Proceedings Real-Time Systems Symposium*, 1994, pp. 216–227.

[17] J. Jonsson, "Robust Adaptive Metrics for Deadline Assignment in Distributed Hard Real-Time Systems," *Real-Time Systems*, vol. 23, pp. 239–271, 2002.

[18] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Mixed Criticality Scheduling in Fault-Tolerant Distributed Real-Time Systems," in *2014 International Conference on Embedded Systems (ICES)*. Coimbatore, India: IEEE, 2014, pp. 92–97.

[19] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, pp. 579–592, 1995.

[20] H. I. Ali, B. Akesson, and L. M. Pinho, "Generalized Extraction of Real-Time Parameters for Homogeneous Synchronous Dataflow Graphs," in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2015, pp. 701–710.

[21] A. Yano and T. Azumi, "Deadline Miss Early Detection Method for Mixed Timer-Driven and Event-Driven Dag Tasks," *IEEE Access*, vol. 11, pp. 22 187–22 200, 2023.