Performance and confidence in feasibility analysis of real-time multi-core distributed systems

Etienne Hamelin, Alexandre Berne, Paul Dubrulle, Myrhal Boudiaf

Performance and confidence in feasibility analysis of real-time multi-core distributed systems

Etienne Hamelin*, Alexandre Berne*, Myrhal Boudiaf*, Paul Dubrulle†
*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France (email: firstname.lastname@cea.fr)
†Alkalee, France (paul.dubrulle@alkalee.fr)

***Abstract*—With the trend towards software-defined vehicles, the scale and complexity of automotive software application is increasing rapidly, so that classical timing analysis methods become hardly practical. This paper proposes a new method, where a system model, formalized in an abstract multi-rate dataflow model of computation, is refined into a precedence-constrained scheduling problem. We characterize, and extend where needed, several schedulability analysis techniques to tackle this problem, and we demonstrate its use in the exploration of partitioning choices.**

***Index Terms*—real-time, multicore, distributed, response time analysis, schedulability, dataflow**

I. Introduction

The automotive industry is facing many challenges in its transformation towards the Software Defined Vehicle (SDV). The number of software components or services is vigorously expanding, as is the complexity of their interaction, and the associated integration issues, especially the real-time aspects. SDV applications represent a new step in complexity, involving thousands of mixed-criticality, both hard and soft real-time tasks, deployed on heterogeneous computing architectures made of microcontrollers and multicore processors, distributed across multiplexed networks, where configurations are moreover subject to over-the-air (OTA) updates during vehicle road life. Due to this whole new range of complexity, classical approaches of software timing analysis face significant challenge.

For these applications, many system-level properties, including timing feasibility (schedulability, reaction latency) cannot be left to be verified in the late integration testing phase. To avoid costly redesign cycles, we advocate a model-based approach to manage software timing properties, from service design to implementation and integration, enabling early in the process to predict application performance and timing feasibility.

In this paper, we present an approach, and a model-based analysis toolkit, based on the real-time multi-rate dataflow language PolyGraph, that allows to determine the feasibility of real-time constraints early in the design cycle.

First, in Section II we position our research with respect to previous work, then we present the context of modern SDV applications and a simple illustrative use-case in Section III. In Section IV-A we present the timing annotations added on the PolyGraph language. Our main contribution is a model of execution that relates the abstract behavior of the dataflow with a real-time, precedence-constrained, task scheduling model. This scheduling model is then analyzed via both simulation and response-time analysis in Section IV-E and IV-G. Finally, in Section IV-H we compare the outcomes of this schedulability study with an actual execution on an embedded platform.

This research was partially supported by project DeepSEA, Grant agreement ID: 955606

II. Related works

In particular, the scientific community has developed a large knowledge base on models of computation, on the deployment of corresponding applications onto embedded computing platforms, and various approaches which enable formal reasoning about safety-related properties.

The embedded and cyberphysical systems community is progressively adopting technologies from the Internet world. For instance, the adoption of embedded Service-Oriented Architecture (SOA) frameworks like the Robot Operating System (ROS) allows engineers to easily develop, deploy and maintain complex processing chains made of many software services, by abstracting behavior from the actual details of the distributed execution platform, in particular the complex effects of dynamic scheduling. However, ROS builds a layer of cooperative scheduling of callbacks within each executor, on top of a typical priority-driven scheduling of processes. This complex scheme makes timing analysis extremely complex [1]. In comparison, some other SOA frameworks (e.g. ZMQ) provide message-oriented abstractions independent of any task or scheduling model.

Since the seminal works by Lee & Messerschmitt [3] on Synchronous Dataflow graphs (SDFG), many variants of the dataflow graph paradigm have been proposed. They explore various trade-offs between expressiveness of the model on one hand (related to the ability to accurately model a diversity of practical industrial systems), and the potential for formal analysis of useful properties based on a system model on the other hand [4]. Properties can be formally analyzed such as causal determinism, consistency, liveness, static schedulability, memory-boundedness, which are of particular interest in safety-relevant applications.

The SDFG paradigm does not explicitly model passing of physical time, therefore does not lend itself to timing analysis. Similarly, the synchronous language paradigm focuses

on logical instants [5], and requires all reactions to happen between consecutive instants – which limits the potential integration of longer-running tasks. In contrast, in the Time-Triggered (TT) paradigm [6] and Logical-Execution-Time (LET) [7], time is the principal means chosen to ensure deterministic communication among software nodes. In these models, specifying a physical time instant for all task release (TT) or communication events (LET) leads to deterministic communications, but at the price of adding new tasks or new constraints to the online scheduler [8].

As a consequence, the models of computation listed above do not scale well to support modern, multi-rate, SOA-distributed SDV applications. In our study, we select the PolyGraph language introduced in [9], [17]. It inherits the determinism and causal actor semantics of dataflow networks, but adds a formal model of time, similar to TT however it constraints only a specific subset of tasks. This model lends itself particularly well to data-driven applications where multi-rate behavior is dictated by the diversity of sensors or actuators typically used in modern automotive, such as Advanced Driver Assistance Systems (ADAS). However the mapping of those applications onto schedulable task sets is yet to be defined.

Several implementations derived from dataflow graphs use static scheduling strategies [3] or static time-triggered implementations [15], as often used in aeronautic environments that require strong time- and space-partitioning, it seems comparatively few approaches have been proposed to deploy dataflow networks using fixed priority [16].

Many real-time software systems rely on a fixed-priority preemptive scheduling policy. While often dominated by other policies (e.g. deadline-driven), it has the advantage of being simpler to implement reliably, to understand and therefore to analyze. Since the formalization of the Response-Time Analysis (RTA) algorithm by [10], [11], many have extended this approach to support multicore parallelism and precedence constraints among tasks [12]. A Directed-Acyclic Graph (DAG) is typically used to represent precedence constraints among sub-tasks that run at the same rate. In particular, [13] extends the traditional RTA with a probabilistic model to estimate response time bounds of a DAG task model. This DAG-based approach however is not sufficient to model all the constraints that emerge from a multi-rate timed dataflow.

In the classical RTA algorithm [11] as well as the probabilistic multicore extension [13], the authors only consider the response time of tasks, i.e. the span from release to completion, wheras the PolyGraph-derived precedence scheme imposes to also model the variability of jobs release times. The Compositional Performance Analysis approach (CPA) [14] uses a set of event curves to model the inter-arrival times.

In this paper, we define an extension to the PolyGraph language to generate the scheduling constraints for large SoA-oriented systems with timing constraints, accounting for their high variability; from these constraints we define an execution model respecting the precedence and real-time constraints, and characterize different analysis methods to determine the schedulability of the resulting system. Our results show that classical approaches have limitations, and we introduce an extension to the RTA analysis framework to overcome them.

III. Context

In this paper, we first explore the challenges to overcome during the development of real-time systems using Service-Oriented Architecture (SOA), powerful centralized computing, distributed sensors/actuators, all concepts required for the advent of a Software-Defined Vehicle (SDV). In this section we present the new challenges that emerge in the SDV domain, we recall the main aspects of the PolyGraph dataflow language, then we present some notations useful to the following analyses.

A. The Software Defined Vehicle challenge

The future generation of vehicles or other domains will implement more and more software components distributed over multiple multi-core Electronic Control Units (ECUs) interacting in real time. From the increasing amount of distributed data, the growing number of software components (SWCs) running on a single target and to keep the safety in the complete system, methods of design and validation need to adapt. By combining model-based engineering and timing analysis, we propose a feasibility assessment method based on the PolyGraph formal model to tackle those evolving challenges.

In a modern vehicle, multiple sensors such as cameras, lidars, radars, produce data at different rates; these signals are fed through various software components, some of which support lightweight state machines, and some support very computationally-heavy image-processing or trajectory prediction. Moreover, both computing and network resources are typically shared among both critical and quality-managed loads. PolyGraph allows to formally model all these constraints, defining rigorously the expected behavior (timing wise), and assess latencies, schedulability and deployment possibilities. Through those analysis, we can refine the previous model to have a better accuracy of the final system.

We illustrate our work on the use-case proposed in [18]. Fig. 1 presents the multirate dataflow structure, and Fig. 2 illustrates the vision processing stages. This system represents the perception stage of an ADAS, where images from a front-facing camera are fed by actor $ImgSrc$ at $15Hz$ into two parallel processing chains. The lane-detection subchain detects road lane markings using classical computer-vision algorithms. First, a perspective transform creates a bird-eye-view, a color filter outlines the white and yellow markings. A boxed-search filter identifies lane keypoints, a 2^{nd} order polynomial is fitted through the identified keypoints, so that the extrapolated lane can be drawn on the bird-eye view. This bird-eye view of the lane is perspective-transformed back into a front view for display. Its output is used for lateral control, typically in lane keeping assistance. The object-detection subchain detects and classifies obstacles like vehicles and pedestrians using a neural network ; its output typically drives an automated emergency braking (AEB) system or other longitudinal control

functions. Due to its heavier computation load, the object-detection subchain runs at a lower frequency, processing only every 5^{th} frame. Both subchains are subject to a strict end-to-end latency constraint, imposed here through the $Display$ sink actor. Although not representative of the scale of modern automotive SOA applications, this didactic use-case already showcases both some limits of existing approaches and the practical usability of our method in this context.

B. PolyGraph summary

PolyGraph is a formal dataflow modeling language, proposed in [17]. We recall here only the main notions. It defines the expected real-time behavior of *actors* (which model the software components), communicating through read-blocking FIFO *channels* (which model message-based communication). Actors *fire* atomically to consume and produce a fixed number of tokens on the incident channels.

The real-time constraints are modeled by defining the expected firing frequency of the actors modeling sensors and actuators in the multi-rate system. Maximal end-to-end latency on channel paths can be defined by adding phase offsets for the first expected firing. The other actors without explicit timing constraints represent *reactive* software components triggered by their inputs.

The formal model allows to infer inherited timing constraints for the reactive actors. Model-checking can then be applied to verify *consistency* and *liveness* properties (hence, memory-boundedness and absence of starvation, including missing inputs at expected real-time start date).

These abstract properties are verified with the minimum set of parameters derived from system engineering constraints. While this abstraction is an advantage in functional modeling to determine the feasibility and coherence of the timing requirements, it lacks precision when taking into account the possible variations in the software implementation. To account for variability in a concrete execution in software, we introduce in Section IV additional notions to capture execution time and jitter, without impacting the verdict on consistency and liveness.

C. Notations

Following [17], a polygraph is a tuple containing a set $\mathcal{V} = \{V_1, ..., V_N\}$ of actors and a set $\mathcal{E} = \{E_1, ..., E_M\}$ of channels. We denote by V_j^n the n^{th} firing of actor V_j in an execution of the polygraph.

A subset $\mathcal{T} \subseteq \mathcal{V}$ of actors are *strictly-timed*. To each actor $V_j \in \mathcal{T}$ is associated a frequency ω_j (equivalently a cycle time Π_j) and phase offset φ_j. For any actor $V_j \in \mathcal{T}$ and any of its firings V_j^n, its cycle time and phase offset determine an exact firing date $\tau_j^n = \Pi_j \times n + \varphi_j$.

From Propositions 1 and 2 in [19], we denote $\rho_i(n)$ the (non-decreasing) count of tokens produced on channel E_i by the first n firings of its producer actor i.e. V_j^1 to V_j^n, and $\sigma_i(p)$ the (non-decreasing) to count of tokens consumed by firings V_k^1 to V_k^p.

Depending on the order of the firings in a polygraph's execution, it may be *non-blocking*, i.e. for any channel $E_i = \langle V_j, V_k \rangle$, the firings of V_j and V_k are ordered so that any firing V_k^n has sufficient input data tokens to prevent a blocking read. We denote $\mathcal{H} = \langle J, \prec \rangle$ the directed acyclic graph (DAG) encoding the partial order of the actor firings in the polygraph's non-blocking executions (with an edge if V_j^n must occur before V_k^p to preserve the non-blocking property, i.e. $V_j^n \prec V_k^p$). Fig. 3 represents the structure of the precedence graph for the first jobs of our ADAS use-case.

IV. Contributions

A. Variations and execution times in PolyGraph

As explained in the previous section, the PolyGraph language reasons on firing instants for the verification of consistency and liveness. In an actual software execution, this restriction is not realistic and maintaining the abstraction comes with advantages and drawbacks (see Section II). To enable reasoning on firing time frames instead of firing instants, we add two modeling parameters: *completion jitter* and *timing budgets*.

Completion jitter allows to define the maximal acceptable deviation from explicit periodicity constraints. For example, an actor V_j with a strict cycle time of $\Pi_j = 100ms$ with a jitter of $10ms$ models a requirement to refresh its output data every $100ms$, plus or minus $10ms$. We denote z_j the completion jitter of a strictly timed actor V_j, and require that $z_j \leqslant \Pi_j$.

Timing budgets are associated to actor firings, to define the maximal acceptable execution time to process input data for the production of output data. It can for example be defined as an estimate of the maximal execution time, resulting from a static or dynamic worst-case execution time (WCET) analysis, or coarsely estimated through benchmarking statistics (depending on stringency of the actor's constraints). We denote b_j^n the budget of firing V_j^n.

The addition of timing budgets and jitter maintains the decidability for the consistency and liveness properties. For the consistency property, only the topology matrix of the polygraph and the frequencies are relevant, and they are not modified by the addition of budgets and completion jitters. The purpose of the liveness property is to determine the existence of at least one valid execution. Budgets and completion jitters can thus be ignored in the liveness property definition. Budgets indicate a maximal execution time delaying the occurrence of a firing, but the minimum remains 0. The completion jitters indicate a variation of the firing date of strictly timed actors, which can be ignored by choosing the fixed date τ_j^n for any firing V_j^n (equivalently, all jitters being set to 0). The Algorithm 1 in [17] thus remains applicable by choosing these values for the additional parameters. A positive verdict on liveness is a required starting point to further analyze the polygraph with other values for budget and jitter.

Given the precedence graph $\mathcal{H}$ for a polygraph, the completion jitters and budgets allow to generate expected execution windows, or time frames, in so-called *optimistic* and *pessimistic* scenarios. In the optimistic scenario, the execution

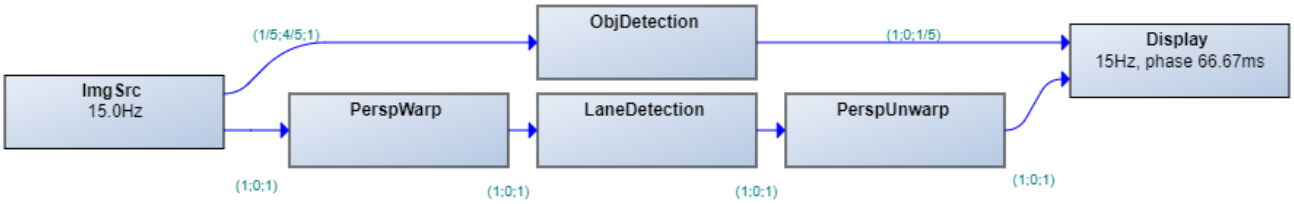

Fig. 1. ADAS use-ecase dataflow

Fig. 2. ADAS vision processing stages: front view (ImgSrc output), bird-eye view (PerspWarp output), lane detection (2 stages), inverse perspective (PerspUnwarp output), and object detection.

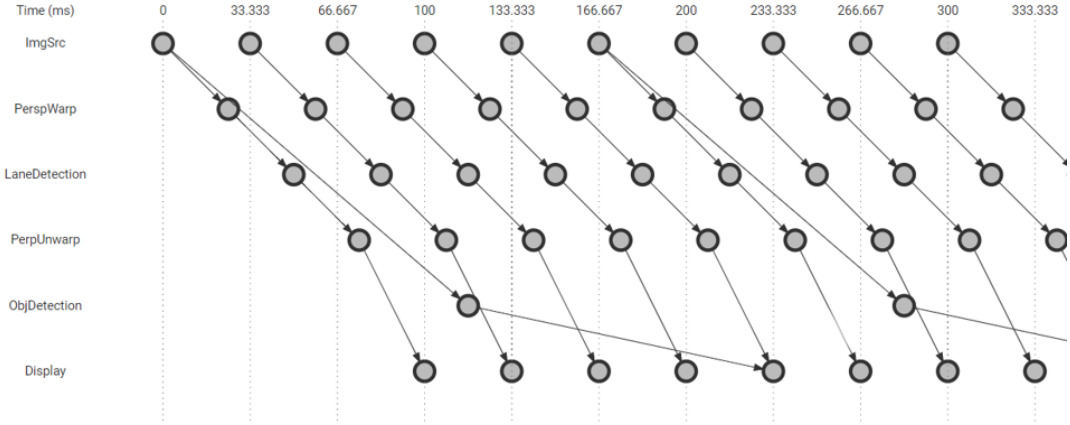

Fig. 3. Jobs precedence graph

times are ignored to define the bounds, and in that case the time frames are maximal. In the pessimistic scenario, the size of the time frames are minimal, considering execution times such that all actors consume the entirety of their timing budgets.

For each firing V_j^n, we define the following time frames:

- **allowed execution**: defines in the optimistic scenario when the actor has all the required data on its input queues at the earliest, and when the data is required at the latest on all its output queues. That frame is thus the maximum time frame during which input data is available for computation and output data is relevant to successors in the precedence graph. We denote al_j^n the lower bound of this frame and au_j^n its upper bound.
- **pessimistic execution**: defines the same instants in the pessimistic scenario, when all firings consume their whole timing budget. That frame is thus the minimum equivalent of the allowed time frame. We denote pl_j^n the lower bound of this frame and pu_j^n its upper bound
- **realization**: defines the instants when the firing (atomic consumption and production) may occur. We denote rl_j^n the lower bound of this frame and ru_j^n its upper bound.

We detail in the following how to determine these time frames. Before going into more details, Fig. 4 illustrates their definition for a polygraph with 3 actors and 2 channels modeling a simple functional chain Sensor $\rightarrow$ Compute $\rightarrow$ Actuator. Sensor and Actuator are strictly timed, with frequencies respectively $10Hz$ and $5Hz$. Actuator has an offset of $50ms$. Compute is a reactive actor with inherited frequency of $5Hz$. The jitters of Sensor and Actuator are set respectively to $10ms$ and $20ms$. If we assume that Sensor produces every two firings the number of tokens consumed by Compute, the polygraph is consistent and live. The precedence graph is then quite simple, Sensor(1) $\rightarrow$ Sensor(2) $\rightarrow$ Compute(1) $\rightarrow$ Actuator(1). The timing budgets are 10ms for the firings of Sensor, 30ms for the firing of Compute, and 20ms for the firing of Actuator.

In Fig. 4, note first the position of the realization time frame of firing *Sensor(2)* and the allowed time frame of *Compute(1)*. The firing *Sensor(2)* is constrained to be realized between 190 and $200ms$ by the timing parameters of the Sensor actor. In

any valid execution, the data is thus produced within this frame. Since it is required by *Compute(1)* as an input, its allowed time frame cannot start before $190ms$. Then note that in the pessimistic scenario, *Compute(1)* ends at the soonest at $220ms$, which constrains the pessimistic lower bound for $Actuator(1)$.

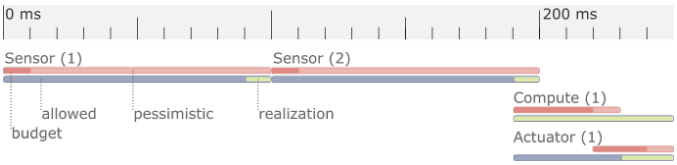

Fig. 4. Firing time frames. The time budget is figured in dark red, the pessimistic frame in pale red, the allowed frame in blue, realization frame in green.

To determine these time frames, the model parameters are used as follows. The allowed time frame for V_j^n is initially defined by interval $[\tau_j^n, \tau_j^n + \Pi_j]$ if V_j is strictly timed. The realization time frame is defined by $[\tau_j^n + \Pi_j - z_j, \tau_j^n + \Pi_j]$. For reactive actors, these frames are initially equal, and defined by $[0, +\infty[$. All pessimistic frames are initially equal to the allowed time frame.

In addition to these initial definitions, every precedence constraint in $\mathcal{H}$ of the form $V_j^n \prec V_k^p$ requires that the data produced by V_j^n be available on time for execution of V_k^p in all scenarios, that is:

$$\begin{aligned} \forall (V_j^n, V_k^p) \in \mathcal{J}, V_j^n \prec V_k^p &\Rightarrow rl_j^n \leqslant al_k^p \\ &\Rightarrow \max(rl_j^n, pl_j^n + b_j^n) \leqslant pl_k^p \\ &\Rightarrow ru_j^n = au_j^n \leqslant au_k^p \\ &\Rightarrow pu_j^n \leqslant pu_k^p - b_k^p \end{aligned} \tag{1}$$

Then, a propagation of these constraints through the graph $\mathcal{H}$ starting with the initial values (for example using a variant of the Bellman-Ford algorithm) refines the initial time frame definitions to ensure causality of communication and real-time constraints are enforced. If a reactive actor's firing has no successor in $\mathcal{H}$, the upper bounds for its time frames can be chosen (as long as it is less than or equal to the hyperperiod of the polygraph, to preserve the consistency property).

For a consistent and live polygraph, the initial allowed and realization time frames can always be refined while respecting the inequalities of Equation (1), since $\mathcal{H}$ is an encoding of the partial order of non-blocking executions and there is at least one valid execution for the polygraph from the liveness property. As such, with the completion jitter extension, any sequence of firings ordered by firing date, such that the firings occur within their realization frame, is a valid execution for the polygraph.

Note that if refining the time frames results in pessimistic frames smaller than the budget or frames where the pessimistic upper bound is out of the realization frame, the system is infeasible regardless of scheduling and resources. Indeed, a frame smaller than the budget implies that the actor have insufficient time to complete in the pessimistic scenario. A pessimistic upper bound out of the realization frame translates in a requirement to produce data sooner that the firing can be realized, which is a contradiction.

For final time frames respecting the inequalities of Equation (1), different approaches can then be used to assess the feasibility of the system timing requirements defining these frames, taking into account a scheduling policy and execution resources. We explore and characterize some approaches in the following sections.

B. Execution model

In this section, we suppose now that a polygraph system is implemented by mapping actors to a set of real-time, partitioned, fixed-priority tasks, connected through first-in, first-out (FIFO) message queues, where each token on channel represents a distinct message. In a service-oriented implementation, these message queues are typically implemented as a publish/subscribe channel supported by a communication middleware such as ROS, DDS, MQTT or ZMQ. We suppose that a common time source is available to all tasks.

We consider that each actor V_j is implemented by a single task, which loops infinitely over the following 4 states. For job index $n = 1, 2, ...$:

- **1. Wait**: If V_j is strictly timed, i.e. has a period/phase offset constraint specified, block until the allowed time frame lower bound is met. If V_j is loosely-timed, switch immediately to Pend state.
- **2. Pend**: block until all the required input data is available, that is for each input channel E_i of actor V_j, block until at least $\rho_i(n) - \rho_i(n-1)$ tokens are available in the corresponding FIFO.
- **3. Compute**: this stage performs the application-specific business logic computation associated with this job. This typically involves processing the payload of the available input messages and computing the payload of the output messages, using CPU time less than or equal to the budget.
- **4. Send**: Block until the start of the realization time frame, then atomically pop required input data from the input queues and push the produced data samples to the output queues, that is on each output channel E_i, $\sigma_i(n) - \sigma_i(n-1)$ tokens.

We further suppose that during the *Wait* and *Pend* phases the processor is left free to run other jobs, i.e. the processor does not waste time in busy-waiting or input polling loops; and that the time spent in the communication stack in the *Send* phase is negligible – or at least, negligible in comparison to the time spent in the *Compute* phase. This is a reasonable hypothesis for several communication stacks such as ROS or ZMQ. For now, we also neglect message transmission times, i.e. a given message is available to its consumer immediately after it was sent by its producer job.

With these assumptions, and given the definition of the time frames in Section IV-A, scheduling a polygraph system

becomes equivalent to scheduling the *Compute* phase of all jobs, subject to the following constraints:

- the release, jitter and deadline constraints imposed on strictly-timed actors,
- the precedence constraints imposed by messages tokens.

Since all these time and precedence constraints can be predetermined off-line from the polygraph system definition, we can lean on the body of knowledge accumulated about precedence-constrained scheduling.

C. Absolute execution window

Scheduling using the allowed time frames (neglecting the budgets) gives the most flexibility to the execution (as the time frames are maximal). When guarantees on the feasibility of a schedule are required, it is best to account for budgets and use the pessimistic frames instead.

When a feasibility test is required, it is mandatory to have a characterization of the execution times of the jobs, we thus consider that BCET and WCET are available.

From there, setting the budgets to the BCET and refining the time frames as defined in Section IV-A provides pessimistic time frames that are the largest possible frames for the Compute phase of the jobs (since they are built considering that all jobs perform as fast as possible without processor resource constraints). We call this time frame for any job V_j^n its *absolute execution window*, noted $[aes_j^n, alf_j^n]$.

As a corollary we have a first *necessary* condition on schedulability: if, for any actor/job indices j, n, the absolute execution window is not long enough for the worst-case execution time, i.e. $aes_j^n + WCET_j > alf_j^n$, then it is not possible to schedule that job within these bounds. Such a system is proven unfeasible.

If this condition is verified, it makes sense to analyze further, by accounting contention on each processor resources.

D. Priority-driven scheduling

If we further suppose that each actor V_j is scheduled, using the Fixed-Priority Preemptive (FPP) policy, on a processor π_j, with distinct priority P_j, then we can refine our estimates of when each job will be executed, and whether processors can schedule all jobs.

In priority-driven scheduling, several jobs might compete for execution time on the same processor. Henceforth a high-priority job might defer or preempt a lower-priority job, hence causing *scheduling interference* (as illustrated in Fig. 5). To model the execution of tasks in the FPP policy, we introduce the following notions.

For any job V_j^n in a FPP schedule, we denote by s_j^n the start time of its Compute phase, f_j^n the completion of its Compute phase, and C_j^n its execution time.

In addition, we denote by if_j^n the amount of *interference* that V_j^n suffers from other, higher-priority jobs (i.e. the total time during which V_j^n is ready to run, but not actually running, on processor pi_j). Finally, we denote R_j^n the *response time* $R_j^n = C_j^n + if_j^n$.

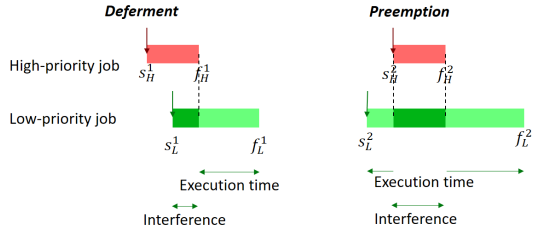

Fig. 5. Scheduling interference caused by a high-priority job HP on a lower-priority job LP. Jobs become ready at ↓. Bright green: LP job is running; dark green: LP suffers interference from HP.

The FPP schedule is then subject to the following constraints:

$$\begin{aligned} \forall V_j^n \in \mathcal{J}, & f_j^n = s_j^n + R_j^n \\ & BCET_j \leqslant C_j^n \leqslant WCET_j \\ & aes_j^n \leqslant s_j^n \\ & f_j^n \leqslant alf_j^n \\ \forall (V_j^n, V_k^p), V_j^n \prec V_k^p & \Rightarrow f_j^n \leqslant s_k^p \end{aligned} \tag{2}$$

Our task is now to estimate, or bound, the amount of interference that any job suffers.

E. FPP simulation

In order to estimate interference, we developed a simulator for FPP-scheduled polygraph systems. This simulator is derived from Equation (2), using a *discrete-event simulation* strategy: at each scheduling point, we compute the state of each actor, and each processor elects the highest priority job among those in the *Compute* state, to run until completion or preemption by another event. Fig. 6 illustrates the chronogram generated by this simulation.

If actor jobs have constant execution times (i.e. $BCET_j = WCET_j$), such a simulation might provide a sufficient schedulability test. If execution times are variable however, it is not sufficient to simulate a schedule with all actors' $WCET$ to get a worst-case response time estimate. Indeed, in certain circumstances, a job taking less time to execute might cause another job to have a longer response time and even miss a deadline. Fig. 7 illustrates such a *scheduling anomaly*, where the first job of A executes for its full $WCET$ duration, then releases the first job of B after C has completed. During the second run at $t = 40ms$ however, A completes earlier, releasing B sooner. This time, B preempts C, increasing its response time.

To build a higher confidence in system schedulability, we simulate many runs of the system (actually, many hyperperiods) in a Monte-Carlo fashion. For each job simulated, a execution time sample is drawn from a uniform distribution between $BCET_j$ and $WCET_j$. The outcome of the simulation is a chronogram, and a histogram for each actor of its response times such Fig. 8. Remember that the maximum *observed* response time is only a lower bound on the actual maximum response time, therefore this method does not constitute a

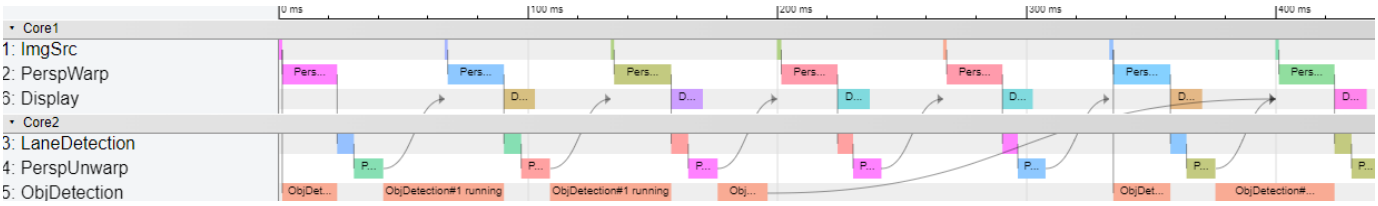

Fig. 6. Simulated FPP schedule

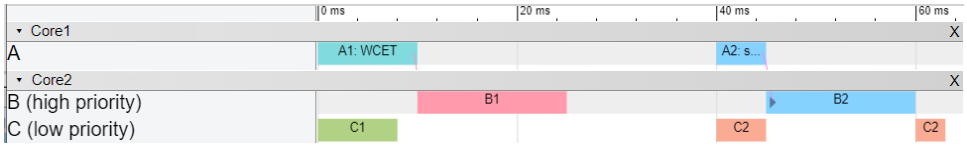

Fig. 7. Illustration of a scheduling anomaly: a shorter execution of actor A at $t = 40ms$ leads to a longer response time of C.

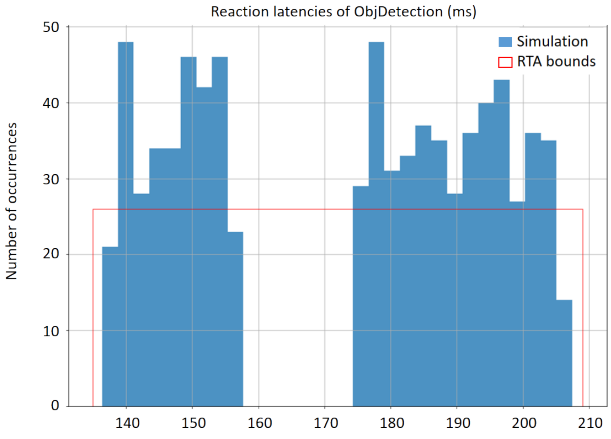

Fig. 8. Response latency histogram of the ObjDetection actor.

sufficient schedulability test. Indeed, a rare interference pattern might remain unveiled during any fixed-length simulated run.

F. Compositional performance analysis

In complement to the simulation approach, which provides a lower bound to job response times, several formal approaches are known to provide upper bounds. The Compositional Performance Analysis method seems an interesting approach, which supports modeling fixed-priority jobs with precedence constraints, deployed over multicore platforms.

We experimented with the open-source pyCPA library however, the response times estimates were very pessimistic. Indeed, current pyCPA implementation does not take into account the phase offsets in job release times. When we attempted to transform a polygraph job graph into a pyCPA precedence-constrained task system, many time constraints were therefore lost in translation, and as a consequence the pyCPA solver accounted many interferences that can not occur in practice. For this reason, we tried to reduce pessimism by adapting another method: the response-time analysis approach.

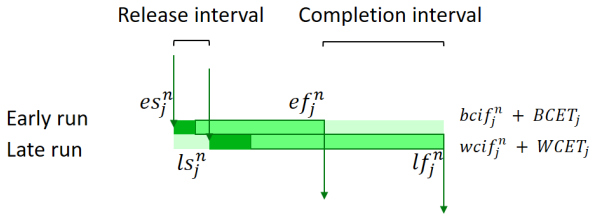

Fig. 9. Interval-based representation of job timing uncertainty. Dark green: best/worst-case total amount of interference; bright: best/worst case execution time.

G. Interval-based response time analysis

From the classical RTA algorithm, we derive an interval-based variant. Its main stage consists in rewriting the execution-time equation 2 using intervals to bound the uncertainty on each operand, then using the precedence graph to propagate these uncertainty intervals to neighboring jobs (similar to the method used for absolute execution windows), and then using these intervals to estimate possible job interference.

If we suppose that the release time of a job V_j^n lies within an interval $s_j^n \in [es_j^n, ls_j^n]$ (*early start*, *late start*), and that the total interference is bounded within $if_j^n \in [bcif_j^n, wcif_j^n]$, then from (2) we can bound the job completion time $f_j^n \in [ef_j^n, lf_j^n]$ (*early finish*, *late finish*) with:

$$ef_j^n = es_j^n + BCET_j + bcif_j^n \tag{3}$$

and

$$lf_j^n = ls_j^n + WCET_j + wcif_j^n \tag{4}$$

Fig. 9 illustrates the notations for early/late start and finish times, and their relation with best/worst-case interference and execution times.

Equation (4) resembles the classic RTA update equation. Note that updating a job's completion date might result in discovering new interfering jobs, so the classic RTA algorithm is applied recursively, until either fix point convergence or a deadline miss is identified. In our case, another argument calls for recursive application: updating the completion interval of a job might also postpone the release time of its successor jobs.

Now let's estimate the interference that a job V_j^n might suffer from another job V_k^p, noted as above as an interval $[bcif(V_j^n, V_k^p), wcif(V_j^n, V_k^p)]$. Consider two arbitrary jobs, with their execution windows as indicated by their early and late start/finish dates. A job V_k^p might defer or preempt a job V_j^n, therefore cause interference, if and only if:

- both are deployed on the same processor, and the victim job has lower priority than its interferer, i.e. $\pi_j = \pi_k$, and $P_j < P_k$,
- and at some point in time, they are simultaneously ready to run.

Remark that if we're sure that an interferer's release time occurs while the victim is running, then the interferer will *certainly* interfere with the victim. We therefore set $bcif(V_j^n, V_k^p) = BCET_k$ when $[es_k^p, ls_k^p] \subseteq [ls_j^n, ef_j^n]$. In all other case, 0 is a safe lower bound to interference.

Similarly, an interference can occur only if the execution frame of the interferer overlaps that of the victim: $wcif(V_j^n, V_k^p) = 0$ when $[es_j^n, lf_j^n] \cap [es_k^p, lf_k^p] = \emptyset$, in all other cases, $WCET_k$ is a safe upper bound.

By further noting that two jobs can't interfere with each other if they are linked by a precedence chain, we can reduce the set of potentially interfering jobs to consider:

$HP(V_j^n) = \{V_k^p | \pi_j = \pi_k \; and \; P_j < P_k \; and \; V_k^p \notin Pred^*(V_j^n) \; and \; V_j^n \notin Pred^*(V_k^p)\}$

Hence, we define:

$$bcif(V_j^n, V_k^p) = \begin{cases} BCET_k \text{ if } V_k^p \in HP(V_j^n) \text{ and} \\ [es_k^p, ls_k^p] \subseteq [ls_j^n, ef_j^n], \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

and

$$wcif(V_j^n, V_k^p) = \begin{cases} WCET_k \text{ if } V_k^p \in HP(V_j^n) \text{ and} \\ [es_k^p, lf_k^p] \cap [es_j^n, lf_j^n] \neq \emptyset, \\ 0 \text{ otherwise} \end{cases} \quad (6)$$

It follows that a lower bound the total amount of interference suffered by job V_j^n is the cumulative interference from all jobs: $bcif_j^n = \Sigma_{V_k^p \in HP(V_j^n)} bcif(V_j^n, V_k^p)$. It might be tempting to similarly sum all $wcif(V_j^n, V_k^p)$, however that would lead to accounting multiple times across a precedence chain the interference from a single job: Fig. 10 shows a schedule where a single high-priority job HP which might interfere with either $LP2$ (at $t = 100ms$) or $LP1$ (at $t = 1100ms$), which are part of a precedence chain $LP1 \rightarrow LP2$; but HP can obviously not interfere simultaneously with both. For this reason, we add to $wcif_j^n$ only the interference that was not already accounted on predecessors of V_j^n, i.e.:

$wcif_j^n = \Sigma_{V_k^p \in HP^*(V_j^n)} wcif(V_j^n, V_k^p)$

with

$HP^*(V_j^n) = HP(V_j^n) \setminus \{V_k^p | \exists V_l^q \in Pred^*(V_j^n), wcif(V_l^q, V_k^p) > 0\}$

This last remark leads to the interval-based response-time analysis Algorithm 1.

Algorithm 1 Interval-based response-time analysis

Input: polygraph system
Output: early and late release/completion times

Initialisation: for all jobs
$es_j^n \leftarrow aes_j^n$
$ls_j^n \leftarrow aes_j^n$
$ef_j^n \leftarrow aes_j^n + BCET_j$
$lf_j^n \leftarrow aes_j^n + WCET_j$
repeat
  for all jobs in topological order **do**
    Update the release interval
    $es_j^n \leftarrow \max(aes_j^n, \max_{V_k^p \in Pred(V_j^n)} ef_k^p)$
    $ls_j^n \leftarrow \max(aes_j^n, \max_{V_k^p \in Pred(V_j^n)} lf_k^p)$
    Update best/worst case interference
    $bcif_j^n \leftarrow \Sigma_{V_k^p \in HP(V_j^n)} bcif(V_j^n, V_k^p)$
    $wcif_j^n \leftarrow \Sigma_{V_k^p \in HP^*(V_j^n)} wcif(V_j^n, V_k^p)$
    Update completion interval
    $ef_j^n \leftarrow es_j^n + BCET_j + bcif_j^n$
    $lf_j^n \leftarrow ls_j^n + WCET_j + wcif_j^n$
    if $lf_j^n > alf_j^n$, schedulability is not guaranteed
    if $ef_j^n > alf_j^n$, deadline miss is certain; return.
  end for
until fix-point convergence

If this algorithm converges, then the system is schedulable, and for each job V_j^n we have computed a lower- and upper-bound to its release and completion times. Fig. 12 visualizes the outcome in a chronogram: each job is represented by two slices: an "early run" between es_j^n and ef_j^n and a "late run" between ls_j^n and lf_j^n.

H. Evaluation

Our performance analysis was evaluated on both our ADAS-inspired use-case, as well as with the use-case proposed by [13], which illustrates a precedence-constraint task set similar to our model.

Reference [13] presents a DAG task model with 2 tasks, detailed in a total of 8 precedence-constrained sub-tasks which are deployed on a 2-processor machine. By modeling message transmission times as additional non-interfering sub-tasks, this DAG task model injects easily into our PolyGraph language, which allows us to analyze it through both simulation and interval-based RTA analysis. This comparison confirms that for most tasks both [13]'s probabilistic method and our interval- based RTA algorithm gives a tight bound, equal to the maximum response time observed in simulation for most sub-tasks, with an overestimation corresponding to 1 message transmission time for two of them.

If we chose to deploy our ADAS use-case on a 2-core machine with the parameters in Table I, Core 2 is overloaded. As a result, both the simulation and interval-based RTA detect a deadline miss. Since the miss probability is low (see the histogram in Fig. 11), many hyperperiods are simulated before detecting a deadline miss.

If however we change the configuration, for instance by deploying the $PerspWarp$ actor on core 1 instead, the system

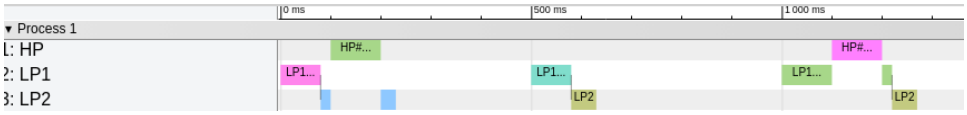

Fig. 10. Example interference on a precedence chain: high-priority job HP may defer or preempt either LP1 or LP2.

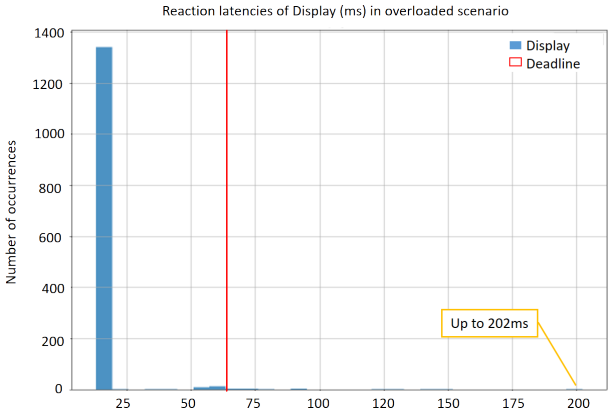

Fig. 11. Response time histogram of the Display actor, in overloaded configuration

Actor	Core	Prio.	BCET - WCET (ms)	Simul. WCRL (ms)	RTA WCRL (ms)
ImgSrc	1	5	1-2	2	2
PerspWarp	2	4	22-23	24.9	25
LaneDetection	2	3	6-7	31.8	32
PerspUnwarp	2	2	11-12	43.8	44
ObjDetection	2	0	100-150	206.7	209
Display	1	1	12-13	37.9	38

TABLE I
ADAS USE-CASE PARAMETERS AND WORST-CASE RESPONSE LATENCIES

becomes schedulable. The last two columns of Table I show the worst-case response latency computed by our interval-based RTA algorithm, and observed over a simulation of 100 hyperperiods. Since the simulation approach provides a lower bound on the actual maximum response times, and the interval-based RTA proves an upper bound, the small gap between both predictions indicates that we have a rather accurate estimation of the actual maximum possible response times.

In addition, from this polygraph system definition we generated a set of task threads communicating through ZMQ messages, respecting the same scheduling parameters. We ran the generated code on a physical multicore target and traced the effective execution during a few minutes. The actual trace of Fig. 13 confirms that each job ran within the early/late bounds observed in simulation and computed with interval-based RTA.

V. Conclusion

The PolyGraph language supports modeling of a rich set of SDV applications, typically in the form of reaction chains involving services running at various rates, deployed over multi-core and distributed ECUs. Equipped with the model of execution proposed in Section IV-B, we have shown that two powerful verification methods can be extended to support the timing analysis of polygraph systems, namely scheduling simulation (Section IV-E) and an interval-based variant of the response-time analysis method (Section IV-G). This approach is validated by comparing the response latencies measured on a simulation, computed with the interval-based RTA method, and measured on an actual target execution.

This paper focuses on the partitioned, fixed-priority preemptive scheduling policy, however both the simulation and response-time analysis tools could be extended as future work, to support other policies such as deadline-driven policies – especially the Constant-Bandwidth Server configuration which is also often used in SDV applications. Similarly, the effect of network scheduling policies on message transmission times (typically in Time-Sensitive Networking configurations) could be integrated in our analysis to refine the estimated response latencies in network-distributed applications. Moreover, we consider adapting some heuristics that have been proposed in literature to assign priorities and explore partitioning configurations.

References

[1] Casini, D., Blaß, T., Lütkebohle, I., & Brandenburg, B. (2019). Response-time analysis of ROS 2 processing chains under reservation-based scheduling. In 31st Euromicro Conference on Real-Time Systems (pp. 1-23). Schloss Dagstuhl.

[2] Lee, E. A., & Messerschmitt, D. G. (1987). Synchronous data flow. Proceedings of the IEEE, 75(9), 1235-1245.

[3] Lee, E. A., & Messerschmitt, D. G. (1987). Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on computers, 100(1), 24-35.

[4] Roumage, G., Azaiez, S., & Louise, S. (2022, December). 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) (pp. 1-9). IEEE.

[5] Potop-Butucaru, D., De Simone, R., & Talpin, J. P. (2018). Synchronous hypothesis and polychronous languages. Embedded Systems Handbook: Embedded Systems Design and Verification.

[6] Kopetz, H., & Bauer, G. (2003). The time-triggered architecture. Proceedings of the IEEE, 91(1), 112-126.

[7] Kirsch, C. M., & Sokolova, A. (2012). The logical execution time paradigm. Advances in Real-Time Systems, 103-120.

[8] Breaban, G., Stuijk, S., & Goossens, K. (2017, March). Efficient synchronization methods for LET-based applications on a multi-processor system on chip. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017 (pp. 1721-1726). IEEE.

[9] Dubrulle, P., Gaston, C., Kosmatov, N., Lapitre, A., & Louise, S. (2019, April). A data flow model with frequency arithmetic. In International Conference on Fundamental Approaches to Software Engineering (pp. 369-385). Cham: Springer International Publishing.

[10] Harter, Paul K. Response times in level-structured systems. ACM Transactions on Computer Systems 5 (1987): 232-248.

[11] Audsley N., Burns A., Richardson M., Tindell K., & Wellings A.J. Applying New scheduling Theory to Static Priority Preemptive Scheduling. Software Engineering Journal, 8(5):284–292, 1993.

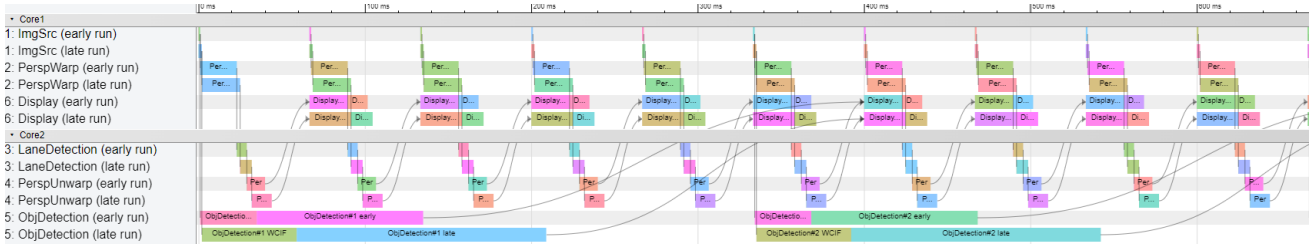

Fig. 12. RTA analysis of our ADAS use-case

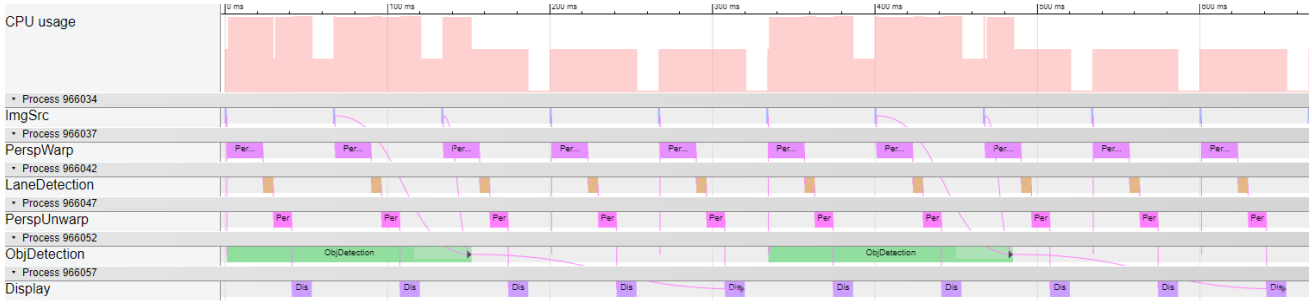

Fig. 13. Trace from a physical execution

[12] Langer T., Osinski L., & Mottok J. (2017, April). A survey of parallel hard-real time scheduling on task models and scheduling approaches, In 30th International Conference on Architecture of Computing Systems.

[13] Slim, B. A., Liliana, C. G., Mezouak, M., & Sorel, Y. (2020, September). Probabilistic schedulability analysis for precedence constrained tasks on partitioned multi-core. In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (Vol. 1, pp. 345-352). IEEE.

[14] Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., & Ernst, R. (2005). System level performance analysis–the SymTA/S approach. IEE Proceedings-Computers and Digital Techniques, 152(2), 148-166.

[15] Carle, T., Potop-Butucaru, D., & Lesens, D. (2015). From dataflow specification to multiprocessor partitioned time-triggered real-time implementation. Leibniz Transactions on Embedded Systems.

[16] Honorat, A., Tran, H. N., Gautier, T., Besnard, L., Bhattacharyya, S. S., & Talpin, J. P. (2023). Real-Time Fixed Priority Scheduling Synthesis using Affine DataFlow Graphs: from Theory to Practice. ACM Transactions on Embedded Computing Systems.

[17] Dubrulle, P., Kosmatov, N., Gaston, C., & Lapitre, A. (2021). PolyGraph: a data flow model with frequency arithmetic. International Journal on Software Tools for Technology Transfer, 23(3), 489-517.

[18] Hamelin, E., Asavoae, M., Azaiez, S., Berne, A., Faure, C., & Trabelsi, K. (2022, June). Multilayer monitoring for real-time applications. In ERTS 2022-11th European Congress Embedded Real Time Systems.

[19] Dubrulle, P., Gaston, C., Kosmatov, N., & Lapitre, A. (2019, November). Dynamic reconfigurations in frequency constrained data flow. In International Conference on Integrated Formal Methods (pp. 175-193). Cham: Springer International Publishing.

[20] R. Pathan, P. Voudouris and P. Stenström, "Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms," IEEE Trans. on Parallel and Distributed Systems, vol. 29, no. 4, pp. 915-928, 1 April 2018, doi: 10.1109/TPDS.2017.2777449.,

[21] A. Singh, P. Ekberg, S. Baruah, "Uniprocessor scheduling of real-time synchronous dataflow tasks." Real-Time Syst. vol.55, pp.1–31, 2019.