Using CPAL to model and validate the timing behaviour of embedded systems

Sebastian Altmeyer, Nicolas Navet
University of Luxembourg
FSTC/Lassy
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
firstname.lastname@uni.lu

Loïc Fejoz
RealTime-at-Work (RTaW)
615, Rue du Jardin Botanique
F-54600 Villers-lès-Nancy, France
firstname.lastname@realtimeatwork.com

***Abstract*—This work presents a solution to the *Formal Methods for Timing Verification (FMTV) Challenge 2015* using CPAL. CPAL stands for the Cyber-Physical Action Language and is a novel language to model, simulate and verify cyber-physical systems as those described in the challenge. We believe that the complexity of the challenge mainly stems from the complex interactions of the tasks and processes composing the aerial video tracking system of the challenge. Using CPAL we have derived a complete and unambiguous description of the system that supports timing verification. The different sub-challenges were solved by timing-accurate simulation and/or schedulability analysis. Even though simulation does not provide firm guarantees on the worst-case behaviour, it helps the system designer solve scheduling problems and validate the solutions, where verification tools can not be applied directly due to the complexity of the model as in the 2015 FMTV challenge.**

I. Introduction

The *Formal Methods for Timing Verification (FMTV) Challenge 2015* is an instance of a schedulability problem where the expressiveness of the traditional real-time scheduling theory reaches its limits. Existing task and processor models simply fail to account for the specific features of the system model; the computational complexity of the problem itself is comparably low. We advocate the use of CPAL (Cyber Physical Action Language), a modelling framework to describe, analyse and program cyber-physical systems. CPAL offers various means to help cope with the inherent complexity of the system, starting with an unambiguous description and simulation of the system model. The CPAL model reveals all ambiguities in the description and thus forces the system designer to consider each important aspect of the modelled system. The simulation environment of CPAL allows to explore the timing behaviour at design-time and to validate or disprove assumptions about it.

In this paper, we exemplify these uses of CPAL and show how it helps to solve the FMTV challenge. The table below summarizes our contributions to the 4 sub-challenges:

	Description	Simulation	Scheduling Analysis
1A	✓	✓	✓
1B	✓	✓	•
2A	✓	•	✓
2B	✓	•	✓

II. CPAL - a modelling and a programming language for embedded systems

CPAL is an acronym for the Cyber Physical Action Language. CPAL is a new language meant to model, simulate, verify and program the kind of Cyber-Physical Systems (CPS) that can be found in cars, planes, robots, UAV, medical devices, home appliance, factory and home automation, power production and distribution, etc. CPAL serves to describe both the functional behaviour of activities, that is the code of the function itself, as well as the functional architecture of the system (i.e., the set of functions, how they are activated, and the data flow among the functions). CPAL is a formal language in the sense that it has well defined concepts of states and transitions. An excerpt of a CPAL source code defining the functional architecture of the model used for challenge 1 is shown in Figure 3. CPAL is meant to support two use-cases:

- a *design exploration platform* for CPS with main features being currently the formal description, the edition, graphical representation and simulation of CPS models. This is how CPAL is being used in this paper, and
- a *development and execution platform*: the vision behind CPAL is that programs can be executed and verified in simulation mode on a workstation and the exact same code can be later run on an embedded board with the same run-time timing behaviour. As CPAL models are currently executed through interpretation, this second use-case is only for the CPS that can afford the performance loss due to interpretation (versus compiled code).

CPAL has been inspired by a number of very diverse languages such as Eiffel, MISRA C and Erlang, model-based design products such as Matlab/Simulink and Scade, verification frameworks such as Promela/Spin and more generally what is usually referred to as the synchronous programming approach (Lustre, Signal, Prelude, etc). However, CPAL has been designed with the requirement to remain a small, simple and unambiguous language. CPAL purposely does not provide constructs that are hard to handle, or lead to convoluted code. To increase the developers productivity, CPAL offers high-level abstractions well suited for the domain of CPS such as

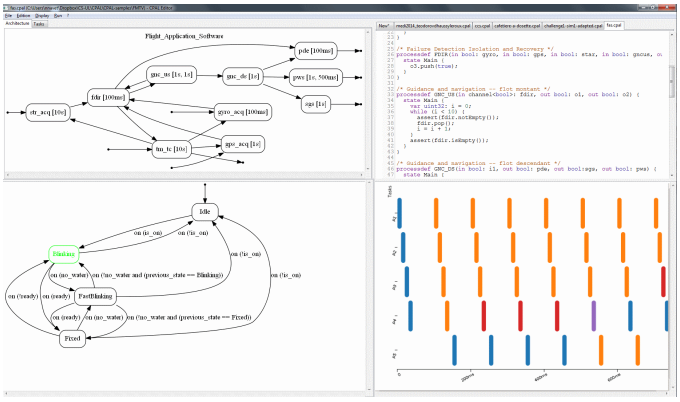


Fig. 1: The CPAL editor shows the functional architecture of a system (top-left), the FSM describing the logic of a process (bottow-left) and the scheduling of the processes as seen during a simulation.

- *Real-time scheduling mechanisms*: processes can be activated with a user-defined period and offset relationships, or upon the occurrence of some external events.
- *Finite State Machines (FSM)*: the logic of a process is defined as a Finite State Machine (FSM) possibly organized in a hierarchical manner where code can be executed in the states, or upon the firing of transitions. The semantics that is implemented in CPAL is the Mealy semantics which enables the control program to react faster on external events,
- *Communication channels* to support control and data flow exchanges between processes, and read/write to hardware I/O ports with well-defined policies (FIFO or LIFO buffering, data overwriting, etc).

Since the language is simple and the abstractions are natural for the domain of CPS, developing CPAL programs is quick and intuitive. In addition, both the FSMs and the data-flow between processes can be visualized graphically in a CPAL-Editor (see Figure 1), enabling some visual verification, and providing a convenient way to explain a program, or part of, to the stakeholders of the project. Model visualization and simulation are already available in CPAL, and for instance a CPAL model was used in [2] to simulate the SOME/IP Service Discovery protocol used to manage service-oriented communication in automotive Ethernet networks. Upcoming releases of the development environment and the CPAL interpretation engine will gradually offer support for both on-line and off-line form verification so that correctness properties can be comprehensively verified. A first step in that direction is the availability of a small utility that extracts the characteristics of the tasks and data flows from CPAL source to connect with external schedulability analysis tools.

CPAL is a language jointly developed by our research group at the University of Luxembourg and the company RTaW. The CPAL documentation, graphical editor and simulation engine for Windows and Linux platforms are freely available for all use from http://www.designcps.com.

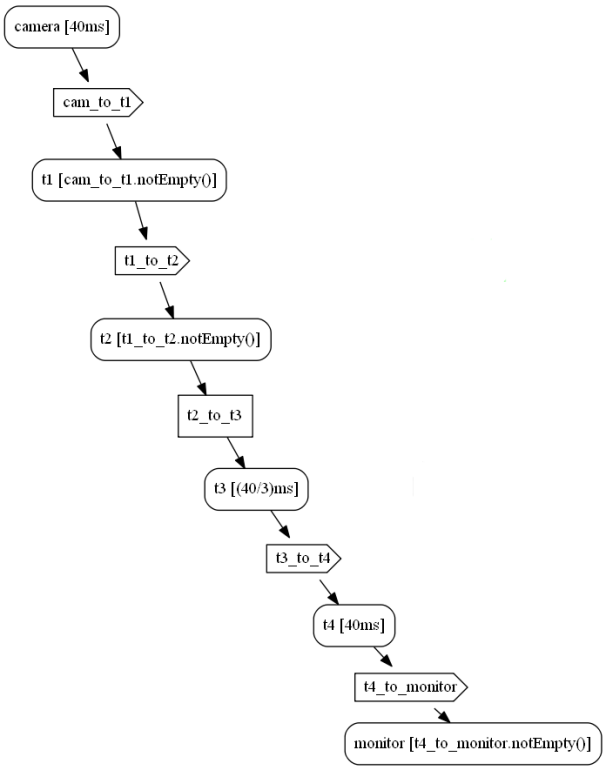


Fig. 2: Graphical representation of the CPAL Model for Challenge 1, the Video Frame Processing. This view is the functional architecture of the model with the activities (processes in CPAL) and their activation conditions, as well as the flows of data between activities.

III. Challenge 1: Video Frame Processing

The timing behaviour and the characteristics of the video-frame processing use case can be modelled and simulated completely in CPAL. Figure 2 shows the graphical representation of the model (as shown in the CPAL Editor) and Figure 3 shows significant parts of the CPAL code. The complete code of the model can be downloaded at http://www.designcps.com/wp-content/uploads/fmtv15.zip. The modelling effort was limited and the complete model was written in CPAL within less than 3 hours.

For the sake of simplicity, we only explain the features of the modelling language relevant for the challenge: *processes*, *queues and registers* and *timing annotations*. The tasks defined in the challenge are represented as processes in CPAL. Figure 3 shows the process definition of *T1_PreProcessor* in line 6 and its instantiation in line 35. The process is activated on the condition that the queue *cam_to_t1* is not empty. Its execution time is defined by the annotation in line 37 to be 28ms. In the graphical representation, rounded boxes correspond to the processes, the angular boxes correspond to the queues and registers. A detailed description of the language can be found online http://www.designcps.com

A. Simulation

CPAL is able to simulate the complete model for the first challenge. The simulation exposed a number of ambiguities in

```
struct Frame {
  uint32: id;
  uint32: emission_time;
};

processdef T1_PreProcessor(
  in channel<Frame>: input,
  out channel<Frame>: output)
{
  state Main {
    /* removes reflections
        normalizes intensity, etc.
        */
    assert(input.notEmpty());
    output.push(input.pop());
  }
}

processdef T2_Processor(...) { ... }
processdef T3_Filter(...) { ... }
processdef T4_DAConvertor(...) { ... }
processdef Camera(...) { ... }

var queue<Frame>: cam_to_t1[1];
var queue<Frame>: t1_to_t2[1];
var Frame: t2_to_t3;
var queue<Frame>: t3_to_t4[n];
var queue<Frame>: t4_to_monitor[1];

process Camera:
    camera[40ms](cam_to_t1);
@cpal:time {
  var uint32: drift = uint32.rand_uniform(999900, 1000100);
  camera.period = (40 * drift)ns;
}

process T1_PreProcessor:
    t1[cam_to_t1.notEmpty()](cam_to_t1, t1_to_t2);
@cpal:time {
        t1.execution_time = 28ms;
        /* assert(t1.bcet == t1.wcet and
                  t1.wcet == t1.execution_time);*/
}

process T2_Processor:
    t2[t1_to_t2.notEmpty()](t1_to_t2, t2_to_t3);
@cpal:time {
        t2.bcet = 17ms;
        t2.wcet = 19ms;
}
...
```

Fig. 3: Excerpt of the CPAL Code for Challenge 1. Process activation conditions are specified at the definition of the processes (e.g., t1 is activated upon the arrival of a frame from the camera). The annotations in the comments are used for the simulation and the analysis of the model.

the challenge description, which directly influence the timing behaviour of the model. The clock drifts and period jitters are given as percentage of the periods, but it is not defined whether clock drifts are considered mutable or immutable. The former case where clock drifts may vary over time can be due for instance to temperature variations (see [1]).

Also the probability distributions of both clock drifts and task execution times are not defined. Yet, the most significant uncertainty were the tasks' release times. A synchronous release of all task may be at first considered as a reasonable assumption, but may not be feasible to implement in practice.

As these factors have not been specified, we had to evaluate the different configurations and selected for each challenge the configuration which led us to the corner cases of the timing behaviour of the model.

B. Solution to Challenge 1A

The main focus of CPAL is on the modelling and the simulation of Cyber Physical Systems. CPAL does not provide any fully automatic analysis to compute a solution to the FMTV challenge. However, it helps to identify and validate best and worst-case scenarios.

The minimum latency for a frame occurs (i) when each task executes for its minimal execution time, (ii) when the buffer is empty, (iii) and when each task is released exactly when the task's input is ready. We note that the minimal execution time of task $T4$ when it processes a frame is $10ms$. Hence, the minimum latency is $63ms$. As the buffer size has no impact on the best-case latency, the minimum latency for the case $n = 3$ is given by the minimum latency for the case $n = 1$. The corresponding scenario is highly unlikely and can only occur for the very first frame. For any other frame $i > 1$, the buffer will not be empty if task $T4$ is released exactly when task $T3$ has produced frame i for the very first time. The best-case scenario for frames with index $i > 1$ is given when frame $i-1$ has been consumed by task $T4$ right after it was produced by $T3$, so minimizing the delay frame i has to wait in the buffer for the next execution of task $T4$. The additional delay is given by the shortest period of task $T4$ minus the longest period of task $T3$, i.e.,

$$40ms - 40ms * 0.00001 - (40/3ms + 40/3ms \cdot 0.00005) = 26.6656ms. \quad (1)$$

The minimum latency for the first frame is thus given by $63ms$, and for all other frames by $89.6656ms$.

$$L^{\min}_{n=1} = L^{\min}_{n=3} = \begin{cases} 63ms & \textit{if frame index } = 1 \\ 89.6656ms & \textit{if frame index } > 1 \end{cases} \quad (2)$$

The maximum latency for buffer size of 1 and of 3 can be considered infinite, as frames may get lost (see Challenge 1B). For the sake of completeness, we derive the maximum latency for frames which are not lost, i.e. frames which are delayed as long as possible but will eventually be displayed.

We start with a buffer size of 1. The worst-case scenario for frame i occurs if the buffer and the register contain the information for frame $i-1$. Task $T4$ empties the buffer right before task $T3$ writes the same information to the buffer again, which happens an instance before task $T2$ updates the register to frame i. Task $T4$ then converts frame $i-1$ twice, empties the buffer right before task $T3$ writes the information for frame i into it. Frame i is thus delayed by two complete cycles of task $T4$; one cycle before it is written to the buffer, and one cycle in which task $T4$ process the previous frame $i-1$ for the second time. Hence, the maximum latency is upper-bounded by the sum of the maximum execution times $65ms$ and twice the period of task $T4$.

$$L^{\max}_{n=1} \leq 146ms. \quad (3)$$

For the sake of simplicity, we only provide the results rounded to the next larger ms.

For a buffer size of 3, a frame can only be displayed twice, if the buffer is empty. The worst-case situation is thus different. It occurs if the buffer is full and frame i can be written to the buffer at the very last moment before it would be otherwise discarded. At that moment, the buffer contains the frames $i-2$, $i-1$ and i and task $T4$ converts frame $i-3$. Frame i is thus delayed by four cycles of task $T4$, and hence the worst-case latency is:

$$L_{n=3}^{\max} \leq 226ms \tag{4}$$

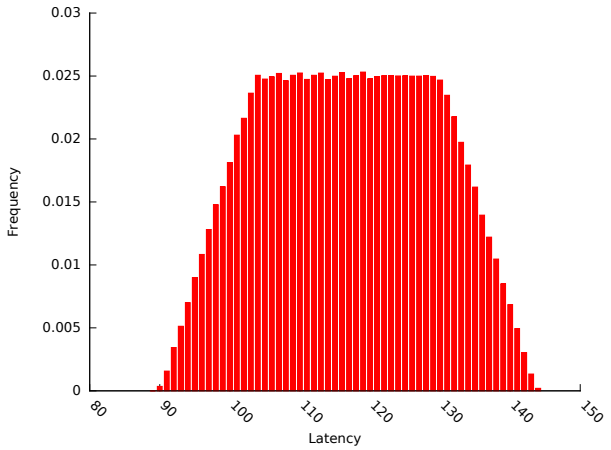
We were able to simulate the Challenge 1 in CPAL and to validate our analyses. We have generated 10^3 different task release patterns and simulated for each pattern the generation of 10^5 frames. We have assumed mutable clock drifts. The total simulation time was less than 6 hours on an *Intel Core I7* clocked at 2.30GHz. Figure 4a shows the distribution of the latencies for $n = 1$ and Figure 4b for $n = 3$. The simulation is very close to our estimated minimum and maximum latencies for a buffer size of 1. The simulation of the system with a buffer size of 3 diverges stronger from our estimates, especially for the worst-case latency, where we have computed a bound of $226ms$, but have only observed latencies of up to $220ms$. Also the shape of the distribution is less uniform and higher latencies (starting from $160ms$) occur with significantly smaller frequency. We presume that the larger state-space of the second case reduces the chance to see the extreme cases.

We also observed that starting from a single task release pattern will require significantly longer simulation time to cover a similar space of possible timing behaviours of the model, compared to aggregating results from a set of simulation runs with different initial release times.

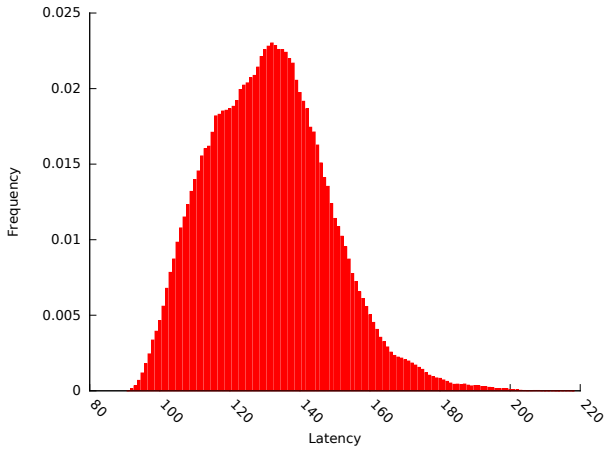
C. Solution to Challenge 1B

We solve challenge 1B by simulation and measure the time between two discarded frames. To maximize the number of discarded frames, we have assumed immutable clock drifts and the maximum period of the camera (producer task) and minimum period of the task $T4$ (consumer task). For each configuration ($n = 1$ and $n = 3$), we have simulated the generation of 10^8 frames. The total simulation time for both configurations combined was less than 4 hours on an *Intel Core I7* clocked at 2.30GHz.

Figure 5a shows the distance for a buffer size of 1, and Figure 5a for a buffer size of 3. In case of a buffer size of 1, the minimum observed distance in between two discarded frames is 2. In this setting, frames are often discarded in bursts, meaning that in an overload condition, many frames can be lost in a relatively small time window. The higher distances shown in Figure 5a show the number of frames in between such bursts. The scenario leading to a discarded frame i occurs when the previous frame $i-1$ is written to the buffer right after $T4$ has read the same frame already before. Frame $i-1$ is thus displayed twice and so, 'blocks' the time window of frame i. Before frame i can be written to the buffer, it has already been overwritten in the register by task $T2$, and so frame $i+1$ will be written to the buffer by task $T3$ instead of frame i.



(a) n=1



(b) n=3

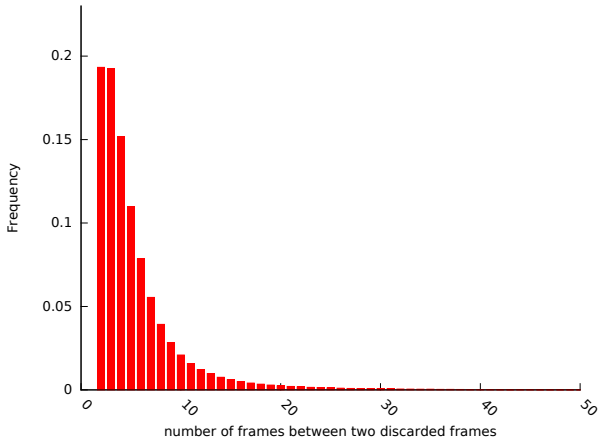
Fig. 4: Latency distribution for Challenge 1A.

A buffer size of 3 prevents such burst situations and ensures a minimum observed distance in between two discarded frames of more than 4500 frames. The discarded frames are well distributed over the life-time of the system and occur (in case of constant worst-case drifts) with a distance of at least 3800 and at most 5500 frames. Due to the policy that the buffer can store each frame at most once, no frame can be displayed twice in case of a buffer size of 3. A frame i is only discarded if the buffer is already full when i arrives and the buffer can not be freed before frame $i+1$ will arrive. After such an overload condition, the contention on the buffer will be reduced and will only slowly increase again due to the clock drifts.

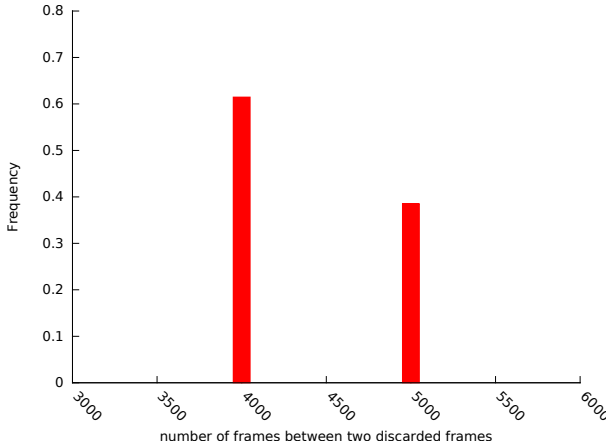
We note that the correctness of these bounds for $n = 3$ strongly depends on the correctness of the simulation parameters and of the selected random distribution. Higher clock drifts can be used to derive lower bounds on the minimum distance.

IV. Challenge 2: Tracking & Camera Control

Challenge 2, the tracking and camera control use case exhibits properties which are not natively supported in CPAL. CPAL is intended to model the behaviour of an embedded real-time system and to specify the timing properties and



(a) n=1



(b) n=3

Fig. 5: Distance between two discarded frames (in number of frames) Challenge 1B.

requirements. The second use-case, however, already specifies scheduling decisions, i.e., pre-emptive scheduling and assigns priorities to the system's components. Especially pre-emptive scheduling is not currently supported by CPAL. We can nevertheless use CPAL to describe the task dependencies and the timing properties of the challenge except for the priority assignment and pre-emption.

The graphical representation of the CPAL model of Challenge 2A is shown in Figure 6a. Figure 6b shows the CPAL model of 2B, i.e., including the additional activation condition of task $T2$ and task $T5$. The dots connected to task $T2$ represent the processes defined in Challenge 1A. The complete model can be downloaded at http://www.designcps.com/wp-content/uploads/fmtv15.zip.

A. Solution to Challenge 2A

We first consider the execution of the subset $\{T5, T6, T7\}$ independently of the pre-empting task $T2$. Due to the priority assignment, i.e, $T2 > T6 > T5 > T7$, the synchronous call of task $T6$ and the asynchronous call of task $T7$ only one execution sequence is possible. We thus combined these tasks to an artificial task Tx.

The execution time of Tx is given by the sum of the execution times of the individual components:

$$[4{+}4{+}9{+}4{+}11ms : 4{+}7{+}10{+}5{+}15ms] = [33ms : 41ms] \quad (5)$$

and the period of this task $100ms + j$. The behaviour of task Tx in isolation is independent of the pre-empting task $T2$. We were thus able to model and simulate the timing behaviour of Tx in CPAL and so, to validate computed execution times. Due to the trivial distribution of the simulation, we have omitted the graph.

The best-case end-to-end latency from activation of task $T5$ to completion of $T7$ is given by the minimum execution time of task Tx, irrespective of the jitter:

$$L^{\min}_{j=0} = L^{\min}_{j=20} = 33ms \quad (6)$$

The worst-case latency can be computed by the traditional response-time analysis:

$$R = 41ms + \left\lceil \frac{R + j}{40.0004ms} \right\rceil 17ms \quad (7)$$

By solving Equation (7), we get the following worst-case latencies:

$$L^{\max}_{j=0} = 75ms \quad (8)$$

and

$$L^{\max}_{j=20} = 112ms \quad (9)$$

B. Solution to Challenge 2B

The latencies are not affected by the shared resource, as the artificial task Tx has lower priority than the pre-empting task $T2$. The best-case scenario remains the same, and so do the minimum latencies:

$$L^{\min}_{j=0} = L^{\min}_{j=20} = 33ms, \quad (10)$$

as well as the maximum latencies:

$$L^{\max}_{j=0} = 75ms \quad (11)$$

and

$$L^{\max}_{j=20} = 112ms \quad (12)$$

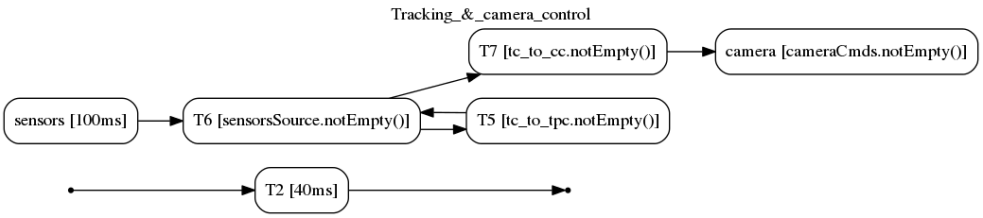
The priority assignment that minimizes the worst-case latency is $T7 > T6 > T5 > T2$ with which task $T2$ can not pre-empt any other task any more, and task $T7$, which controls the camera, can do so before task $T5$, which has no observable output finishes. We have modelled the various priority assignment *within* the subset $\{T5, T6, T7\}$ of the complete task set, i.e. without the independent task $T2$, in CPAL and have been able to validate this conjecture.

The latency is given by the sum of the maximum execution time of the tasks $T7$, $T6$, $T5$, except for the last fragment of task $T6$ and by the maximal delay task $T2$ can block the execution of task $T5$, i.e. $2ms$. Hence

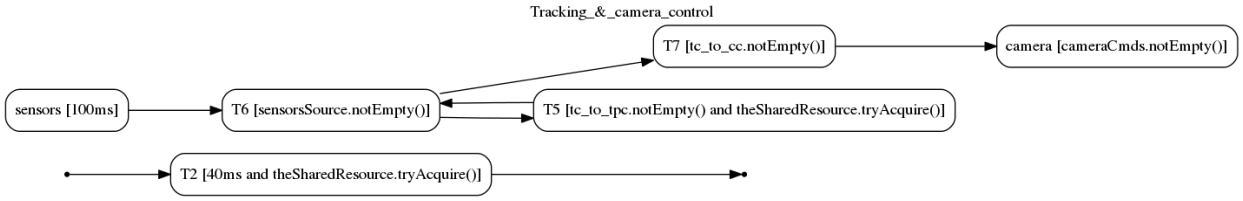
$$L^{\max} = 37ms \quad (13)$$

V. Discussion

In the process of modelling the systems under study we identified a few undefined properties: whether the tasks had offsets relationships or not, the probability distribution for



(a) 2A



(b) 2B

Fig. 6: Functional architecture of the CPAL Model for Challenge 2.

random numbers, and whether clock drifts are set for once or dynamic at run-time. We believe that the use of a precise description language like CPAL, with well understood abstractions, helps to limit the ambiguities among the stakeholders of a project, and ultimately improve the quality of the delivered product and time-to-market. In addition to an unambiguous description, provided that the functional behaviour of the tasks is coded, it becomes possible with CPAL from the same code to simulate, prototype and execute the application.

Deriving the solution for the first challenge by hand proved to be error-prone, and the use of simulation was helpful to better understand the dynamics of the system, and specifically check whether the worst-case conditions we devised could actually happen. In order to increase the likelihood to meet unfavourable scheduling scenarios, we used a random number generator that gave higher probability to the bounds of the interval, instead of a uniform distribution. This simple strategy was effective in creating situations leading to the maximum interferences in our experiments on the first challenge. Simulation is also helpful to gain confidence in the schedulability analysis results. For instance, a flaw in the analysis can be detected if the maximum values observed by simulation exceed the bounds derived by analysis, which never happened here. Finally, simulation provides more fine-grained information than schedulability analysis, such as here the minimum observed distance between two discarded frames, which may permit to consider a solution with lost frames provided that the application is sufficiently robust to tolerate the loss rate observed.

With the help of a simple utility it is possible to extract from the CPAL system's description the characteristics of the tasks and automate a schedulability analysis. We can possibly add more semantics in CPAL with new language constructs or annotations, for instance to specify complex activation patterns like multirate tasks or activation upon the reception of several messages. However, we do not see how to answer in an automated manner complex questions like asked in this challenge without resorting to ad-hoc analyses. Identifying the scope of what can be fully, or partially, automated is in our view a question that deserves future work.

VI. Conclusion and Future Work

In this paper, we have derived a solution to the 2015 FMTV Challenge using CPAL, a new modelling language for cyber-physical systems. The correctness of our solution depends on the correctness of the CPAL model: does the model capture the behaviour of the system correctly? We believe the unambiguous while rather intuitive semantics of CPAL helps to perform this verification, and gain confidence in the model.

Verification in the time domain in the early phases of the development cycle, where the costs of repairing errors is the smallest, is mainly non-existing today and this is one of the limitations of Model-Driven Development for real-time systems. A main reason for this is that verification techniques are not sufficiently well automated and integrated within development environments. CPAL, a modelling language supporting timing verification, is a contribution in that direction.

The intention of CPAL is to provide not a only a modelling language, but also an interpreter which ensures equivalence between the simulated behaviour of the model and the behaviour on the execution platform. CPAL code can be currently executed on a workstation, as a stand-alone interpreter or within Matlab-Simulink, or on an embedded target through model interpretation. For CPS requiring maximal performances, code generation or hooks to call native object code from CPAL programs would be feasible options.

References

[1] A. Monot, N. Navet, and B. Bavoux. Fine-grained simulation in the design of automotive communication systems. In *Embedded Real-Time Software and Systems (ERTS2 2012)*, Toulouse, France, 2012.

[2] J. R. Seyler, T. Streichert, M. Glaß, N. Navet, and J. Teich. Formal analysis of the startup delay of SOME/IP Service Discovery. In *Proc. 2015 Design, Automation & Test in Europe Conference*, DATE '15, pages 49–54, San Jose, USA, 2015.