The CPAL programming language

Design, Simulate, Execute Embedded Systems

Lean Model-Driven Development through Model-Interpretation

BY NC ND

Nicolas Navet, University of Luxembourg

October 29th, 2015

Talk @ CEA LIST, Palaiseau

Outline

A Real-time embedded systems: where are we now?

B What is CPAL ?

C Processes are recurrent Finite State Machines

D Declarative programming & timing-augmented design flow

E CPAL at work : 4 case-studies

Real-time embedded systems: where are we now?

"The question [..] is no longer primarily, "can it be built", but should it be built?"

- **Cross-domain technologies** are there imo for the needs of the next 10-20 years: switched Ethernet, hypervisor, multicore, ...
- **From federated to integrated architecture**: complexity moved from hardware to software but remains high
- **Safety** : a large body of standards, processes, tools, and know-how available → process-based to product-based
- **Timing verification techniques**: Deterministic resources + bounded workload = worst-case timing verification, end-to-end verification with heterogeneous resources possible, accuracy excellent even for large systems
- **Ongoing R&D (most low risks imo)**: mixed-criticality systems, predictable multicore platforms, hierarchical scheduling, incremental verification/certification, correctness in the value domain

✓ Biggest threat to correctness is complexity

✓ Needed now is affordability (time, effort, money)

✓ We can simplify design phase & execution platforms thanks to computing power - our proposal: MBD with Model-Interpretation and Time-Triggered execution

What is CPAL?

A Contribution towards addressing what Thomas Henziger in [4] called the grand challenge in embedded software design

" Offering high-level programming models that
- *permits the programmer to express desired reaction and execution requirements,*
- *Permits the compiler and run-time systems to ensure that these requirements are satisfied "*

CPAL: an interpreted language running on a real-time execution engine

5-steps of MBD

Matlab/Simulink
Scade

CPAL

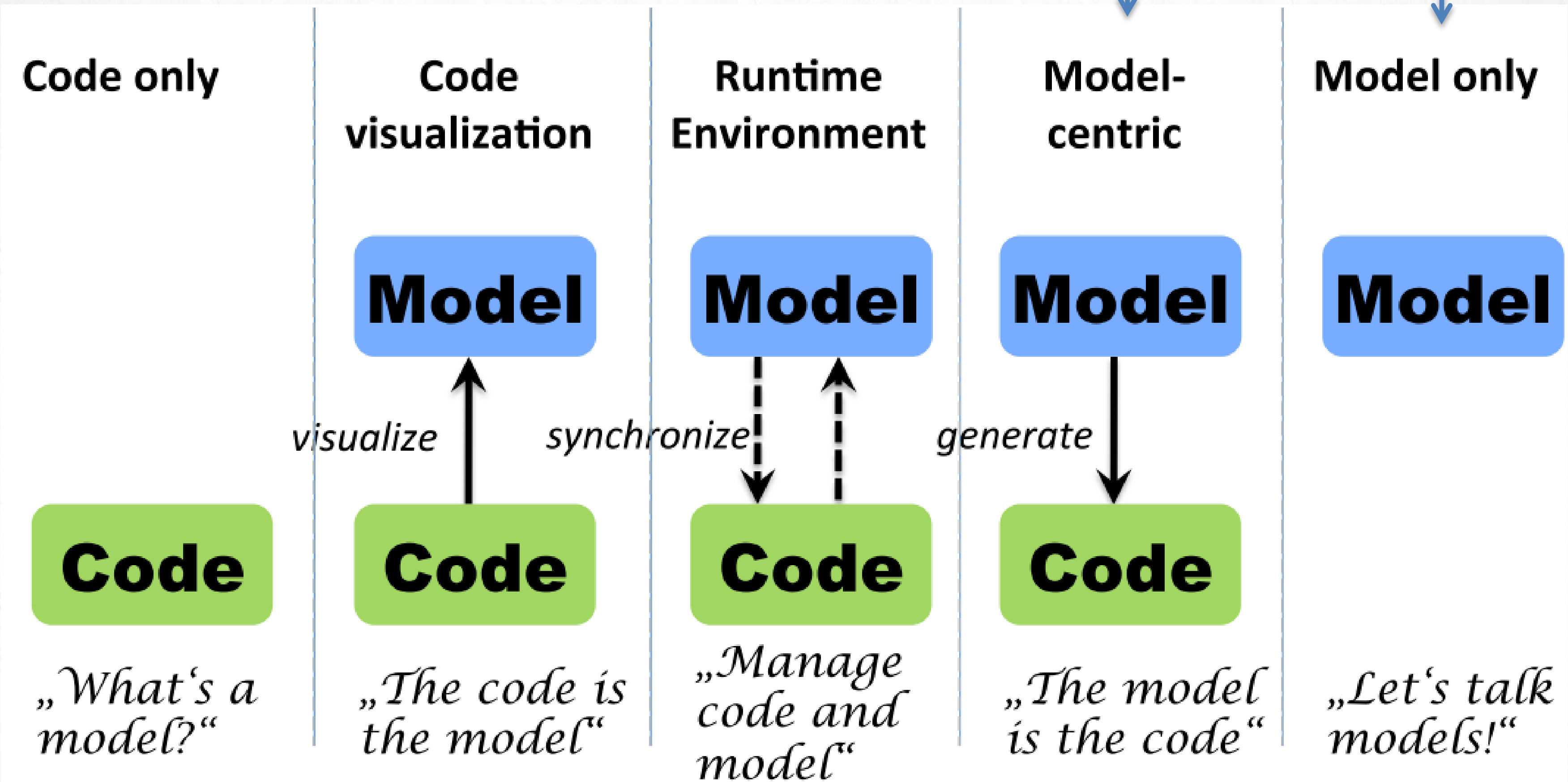


Figure from [2] and [3]

Inspired from interpreter-based SIL4 interlocking systems
e.g.: RATP, SNCF [5], Westingshouse

What is CPAL?

- **A** A language to develop CPS - offering the right abstractions for functional and non-functional properties : activation patterns, FSM, scheduling, communication channel, introspection, etc
- **B** A real-time execution engine that can be run on bare hardware
- **C** Write-Once Run-Everywhere with equally acceptable timing behaviors
- **D** Modelling and simulation language for Design Space Exploration
- **E** A design flow to learn and teach MDD

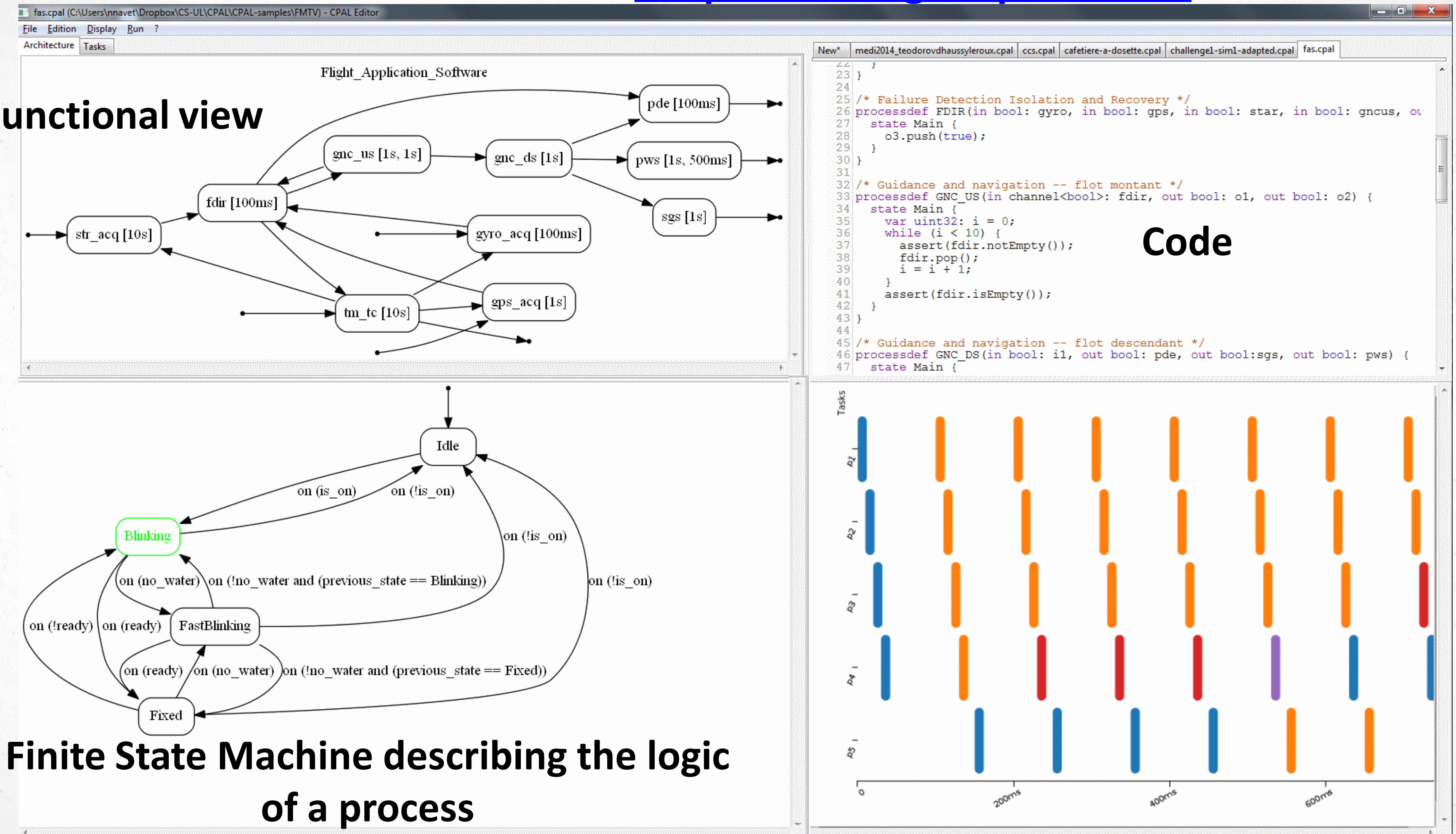
A joint project from RTaW and University of Luxembourg

Hello, world


processdef HelloWorld()
{
state Main {
println("Hello, world");
}
}
process HelloWorld: aTask[100ms]();

Development environment

available from http://designcps.com



Functional view

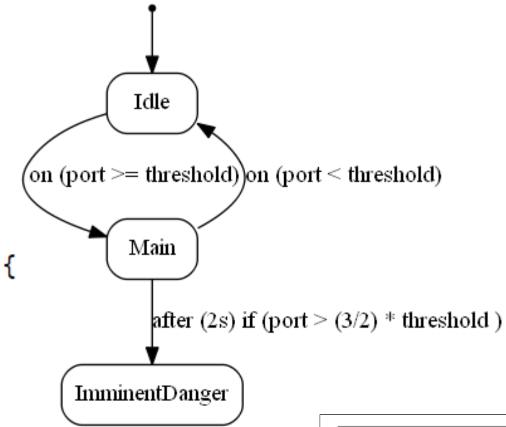
Code

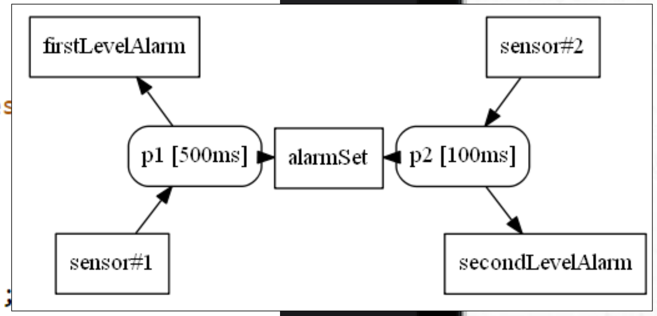
Finite State Machine describing the logic of a process

Activation of the tasks over time

Hello, world

```
processdef MonitorProc(in uint8: aPort, out bool: anotherPort, out bool: danger)
{
  const uint8: threshold = 30;
  state Idle{ }
  /* ... */
  state Main {
    assert(aPort >= (threshold/2));
    anotherPort = (aPort > threshold);
  }
  after (2s) if (aPort > (3 * threshold) / 2) {
    danger = true;
  } to ImminentDanger;
  on (aPort < (threshold/2)) to Idle;
  /* ... */
}

var uint8: sensor#1; /* mapped to some I/O port */
var uint8: sensor#2; /* and updated upon activation of the processes
var bool:  alarmSet = false;
var bool:  firstLevelAlarm = false;
var bool:  secondLevelAlarm = false;

/* Instantiation of periodic monitoring processes*/
process MonitorProc: p1[500ms](sensor#1, alarmSet, firstLevelAlarm);
/* Second process is only executed when firstLevelAlarm is true */
process MonitorProc: p2[100ms][firstLevelAlarm](sensor#2, alarmSet, secondLevelAlarm);
```





Why a new programming language for Embedded Systems ?

- General purpose programming languages do not offer **the right abstractions** for:
  - Periodic activities and real-time scheduling
  - Time measurements and manipulation
  - Finite state machines
  - High-level interfaces to I/Os
  - etc

Both functional and non-functional concerns

- Design for facilitating the writing of **correct embedded code** (incl. restrictions)
- "Write once, Run Anywhere" of Java does not **guarantee** anything about **timing behaviour** on different platforms
- Development environments are unnecessary complex and often expensive
- Model interpretation, although slower, brings benefits in terms of ease of development, error monitoring at run-time, security, no semantics distortion between model and code, scalable redundancy, independence from the platform, etc.

Through declarative programming, then system synthesis

Process introspection

```
processdef aProcess()
{
  state Main {
    println("pid %u", self.pid);
    println("period %t",self.period);
    println("offset %t",self.offset);
    println("curr %t",self.current_activation);
    println("last %t",self.previous_activation);
    if (self.current_activation > 0ms) {
      assert((self.current_activation-self.previous_activation) == self.period);
    }
  }
}

process aProcess: p1[100ms]();
```

First time when the current and previous instances obtained the CPU

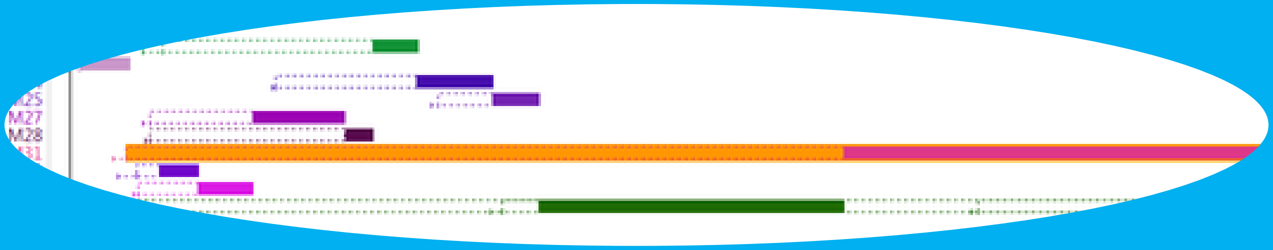
Introspection can serve to implement adaptive behaviours, such as algorithms that depend on the rate of execution or the jitter of the process

www.designcps.com

State-of-the art

- With respect to synchronous languages ?
  - Less demanding programming style
  - No time-determinism but rather timing-predictability
  - Not amenable yet to verification in the value domain
- Unlike pure Architecture Description Languages like Giotto and Prelude, CPAL is also a programming language and an execution platform
  - Same time-triggered execution model as Giotto
  - Could take advantage of the rich data-flow language of Prelude
- With respect to Papyrus-RT ?

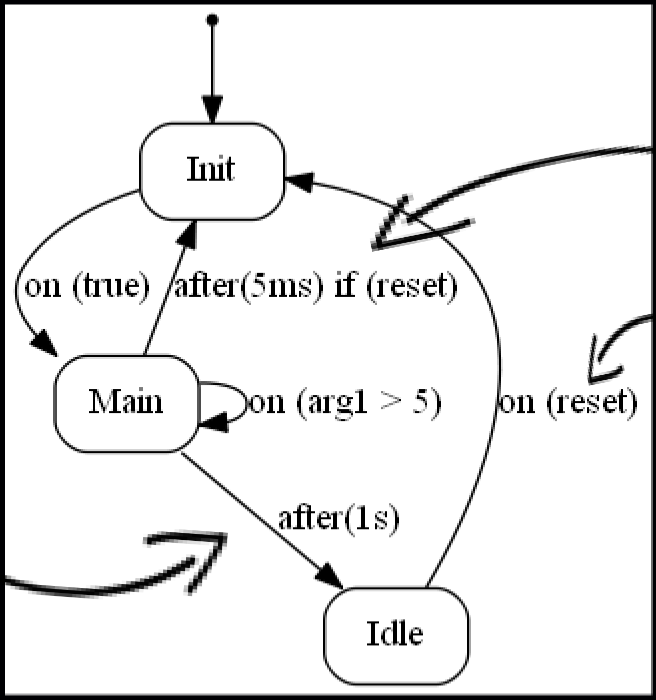
CPAL = Imperative programming in the functional domain + declarative programming in the non-functional domain + Time-Triggered execution platform



Processes: recurring activities whose logic is described as Finite State Machine

Finite-state Machines to describe the logic of processes

Code both in states and transitions

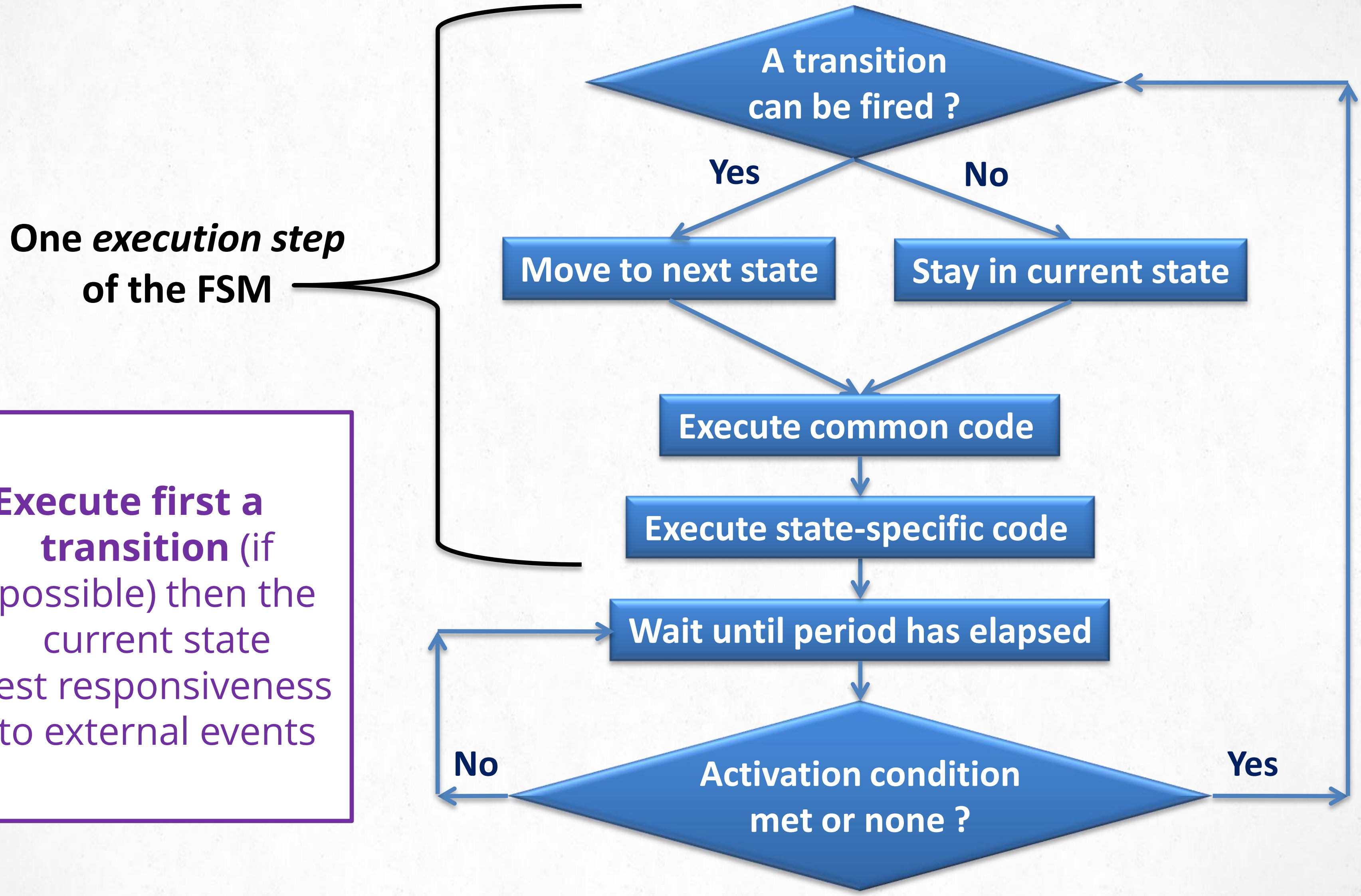


Timed transition and condition

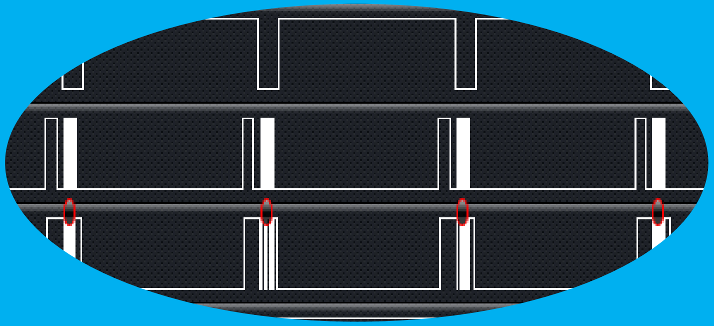
Boolean condition

Timed transition

Periodic activation of a process

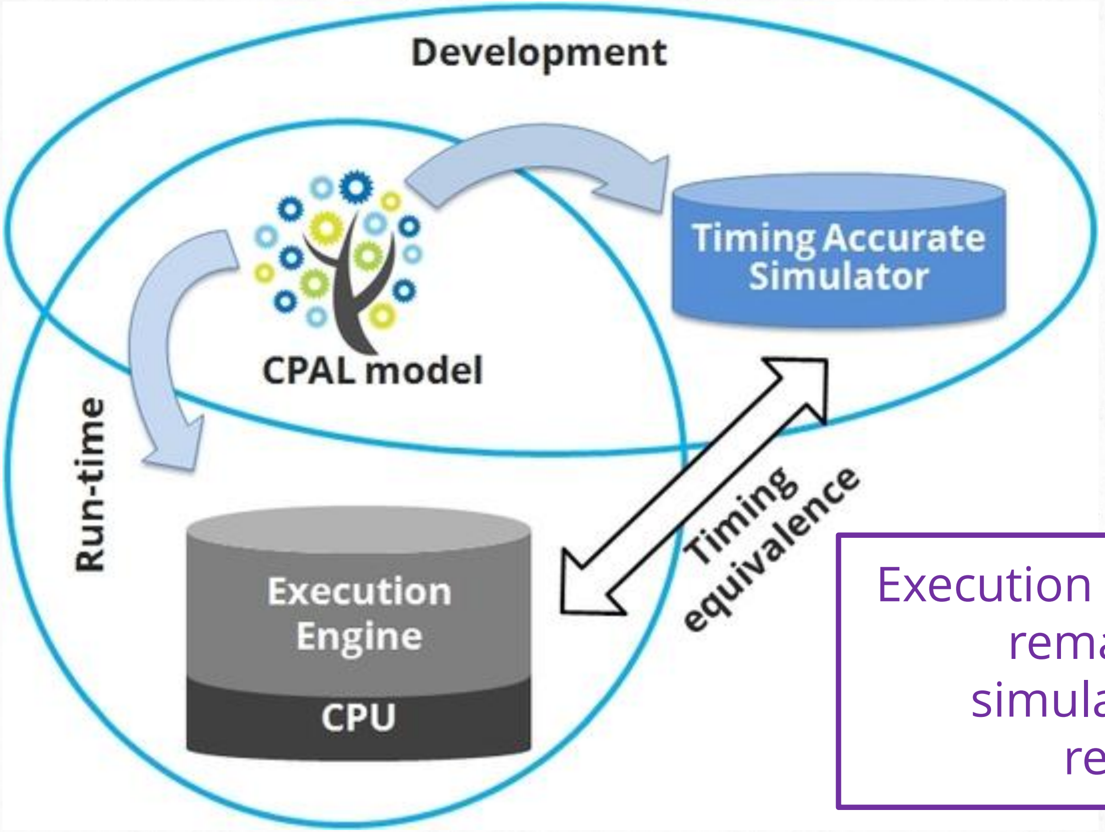


Execute first a transition (if possible) then the current state
→ best responsiveness to external events



Simulation and Real-Time Execution Mode

CPAL Execution Modes



Execution order of processes remains the same in simulation mode and in real-time mode

Event-order determinism is not always needed and is not always sufficient, need for a concept of "timing-equivalent execution"

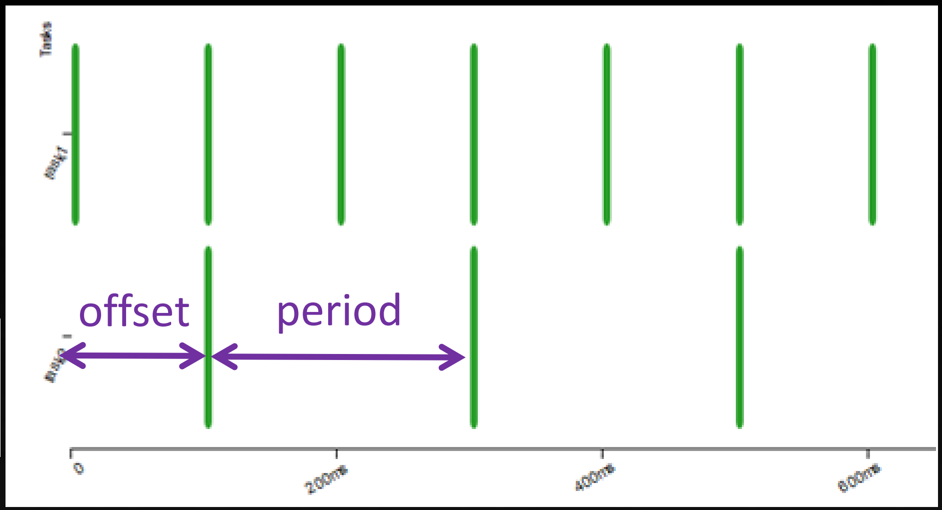
Simulating execution times

```
processdef OneShortOneLong()
{

  state State1 {
    @cpal:time {
      State1.execution_time = 20ms;
    }
  }
  on (true) to State2;
  state State2 {
    @cpal:time {
      State2.execution_time = 40ms;
    }
  }
  on (true) to State1;
}

process OneShortOneLong: aTask[60ms]();
```

Timing annotations can be inserted manually or by a Worst-Case Execution Time analyzer and are used by the simulator

Process activation model

```
/* Periodic process */
process MyProcess: task1[100ms]();
```

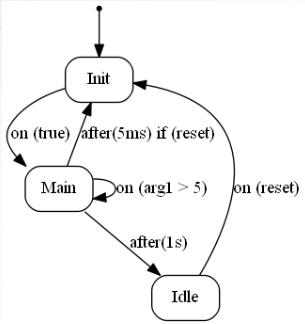


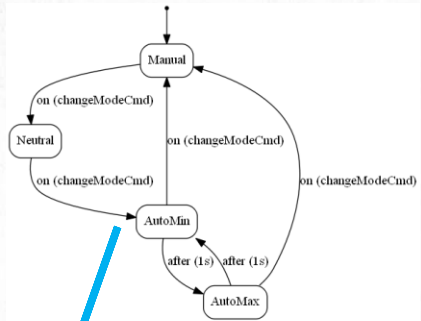

```
/* Periodic process with initial offset */
process MyProcess: task2[200ms, 100ms]();
```

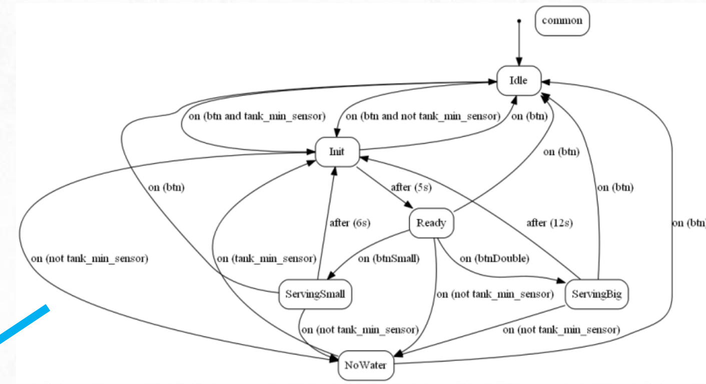
```
/* Periodic with additional execution condition */
process MyProcess: task3[600ms][aTriggerCondition]();
```

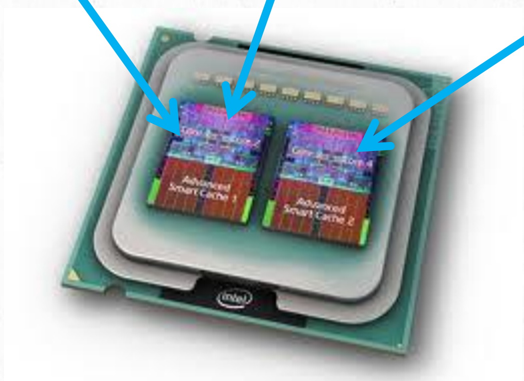
Activation conditions are for functioning modes and event-triggered activities

Declaring timing correctness









Constraints: deadline, frequency, jitters, data-flow (precedence, prod. rate), safety, etc

A Allocate the models to the core

B Set offsets and possibly periods

C Set scheduling parameters

Ideas drafted in [6] but scheduling synthesis not implemented yet

Basic schedulability analysis

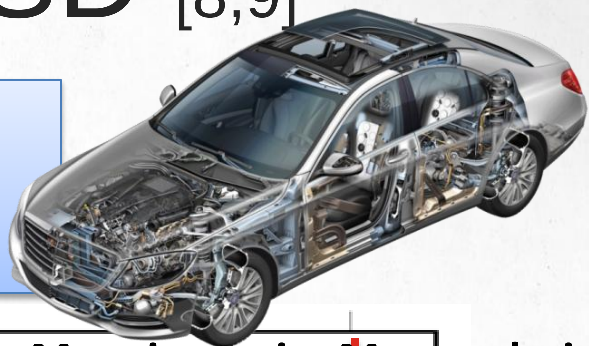
- WCET by measurements (runtime monitoring)
- Current scheduling policy is FIFO
  - Non-preemptiveness + enforce event-order determinism
  - Work-conserving unlike static cyclic scheduling
  - But limited resource usage, offsets helps here
- Schedulability analysis with offsets is difficult
  - Exact analysis but exponential time
  - Polynomial time but approximate
- Better resource usage with the digraph task model

Ongoing work [7]

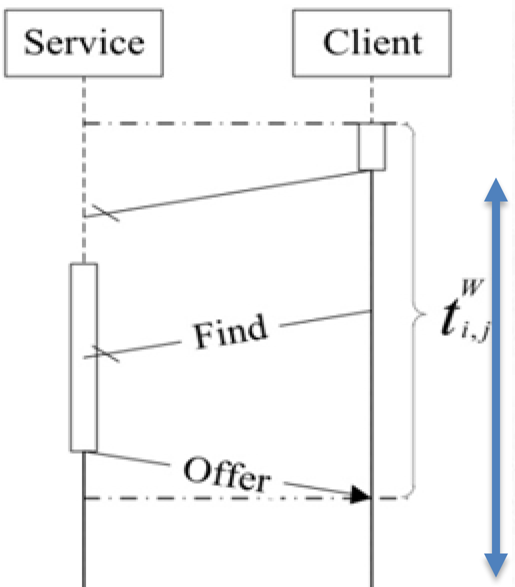
Use-Cases

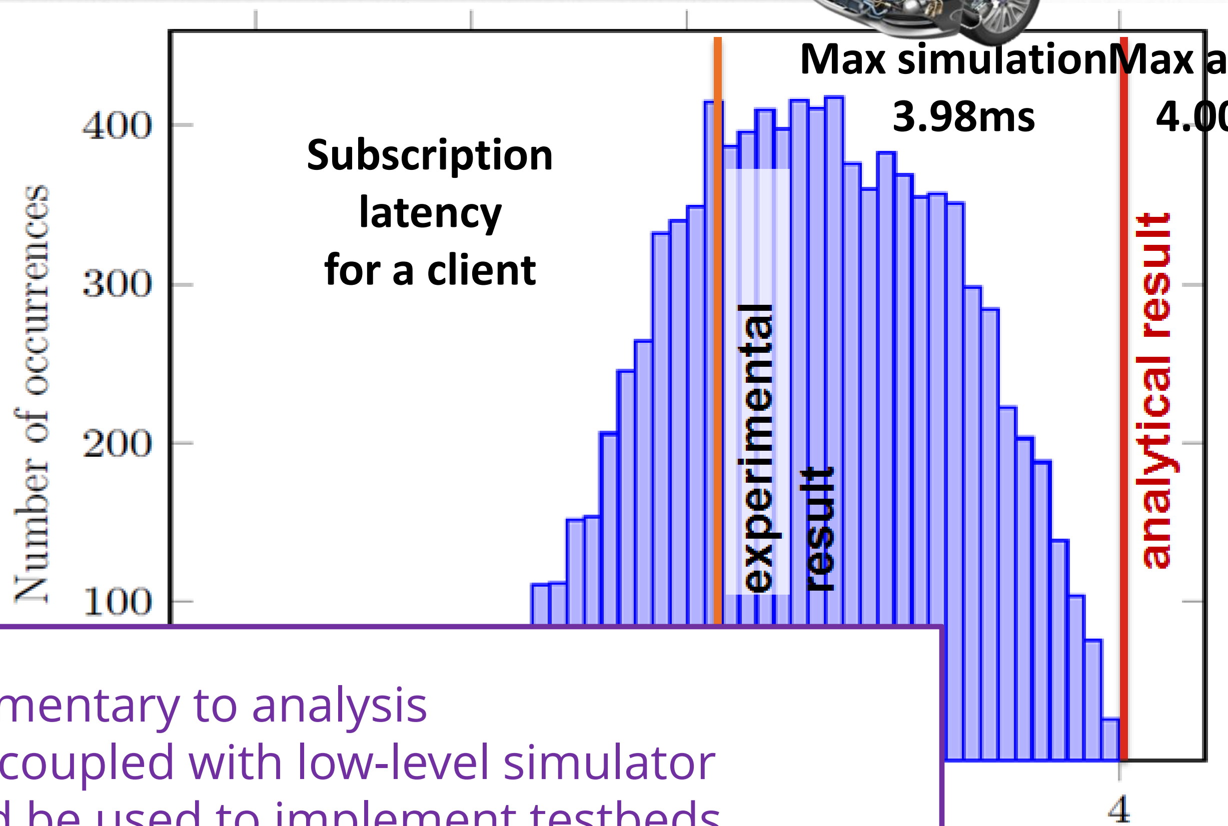
UC#1

Simulation: Some/IP SD [8,9]



SOME/IP SD: **service discovery for automotive Ethernet**
Objective: find the right tradeoff between subscription latency and SOME/IP SD overhead



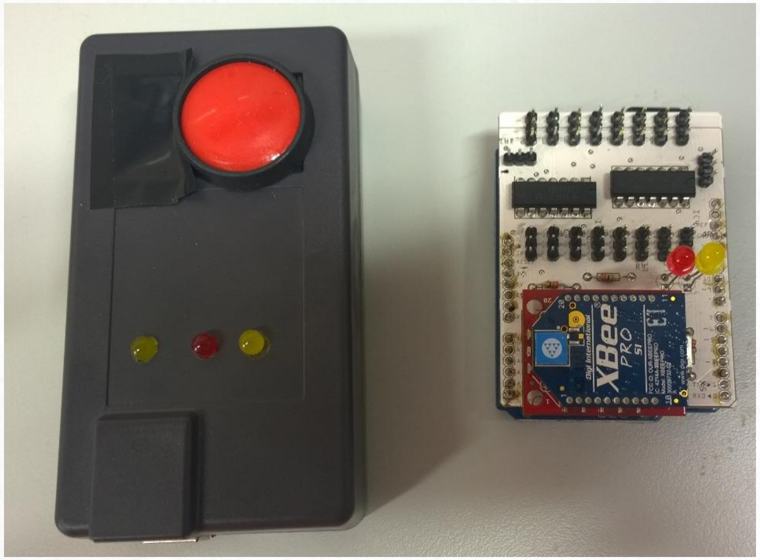


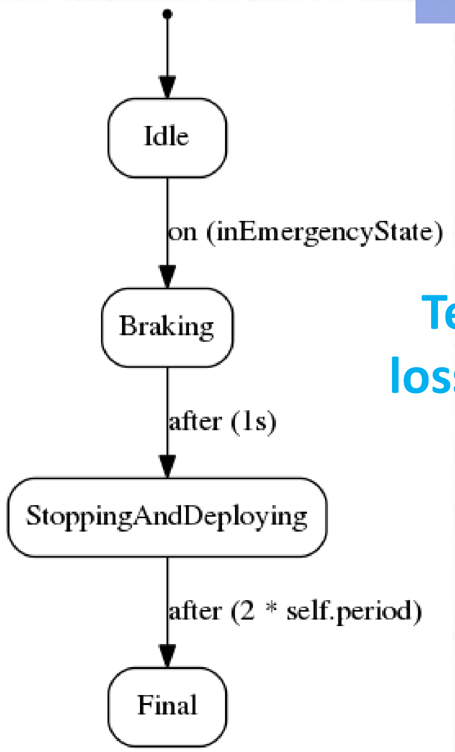
- ✓ Simulation complementary to analysis
- ✓ Models have been coupled with low-level simulator
- ✓ Same models could be used to implement testbeds

UC#2

Developing CPS: a smart parachute for UAV [11]

UAVs autopilots cannot be trusted –
minimal safety through a remote termination component
Partnership with Alérion company

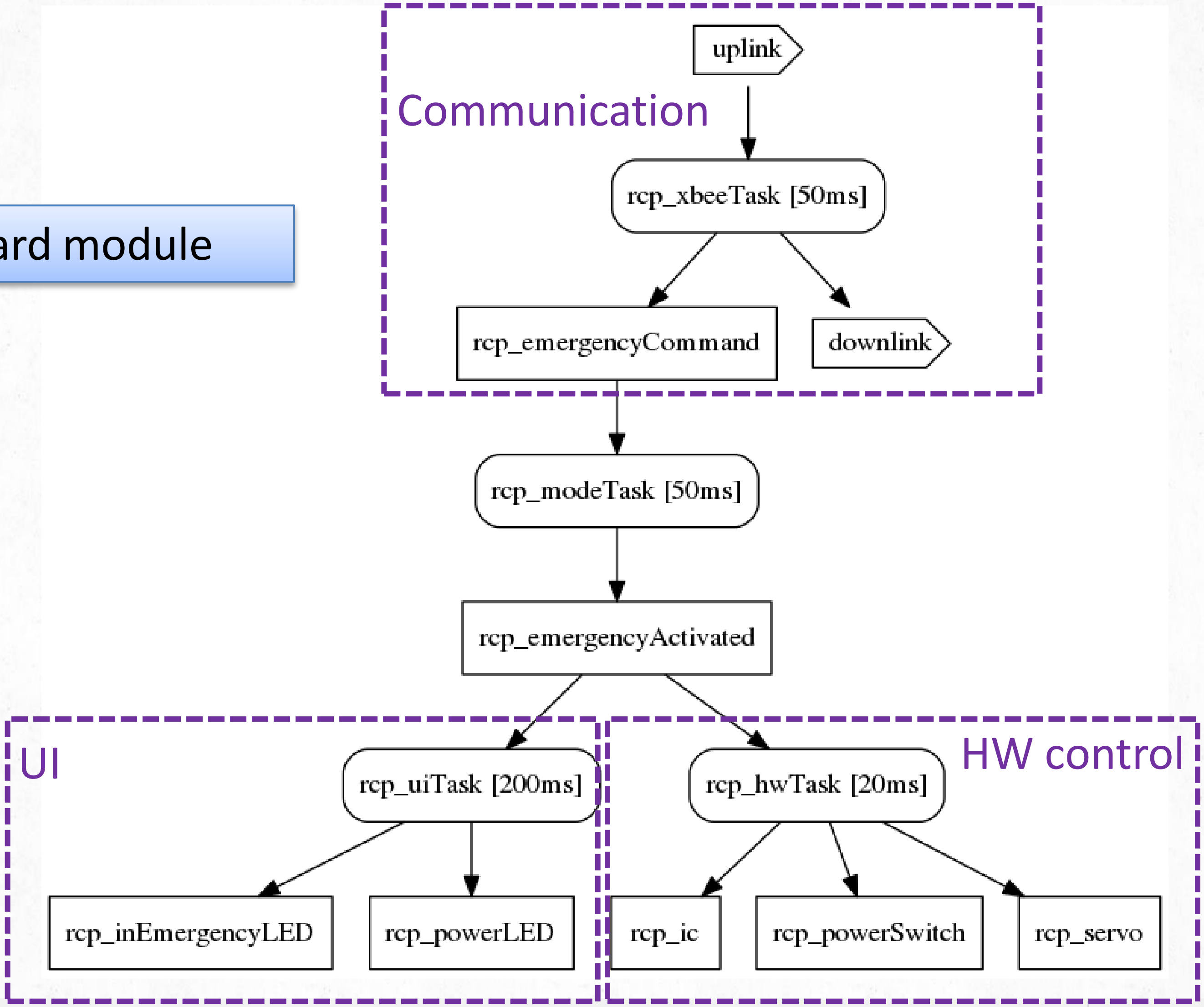




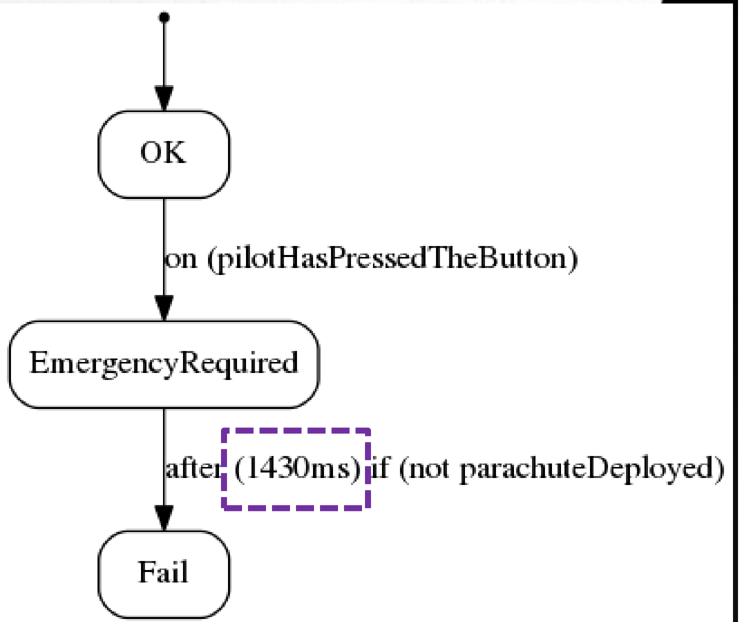
Termination upon loss of connection or pilot's decision

Software architecture

On-board module



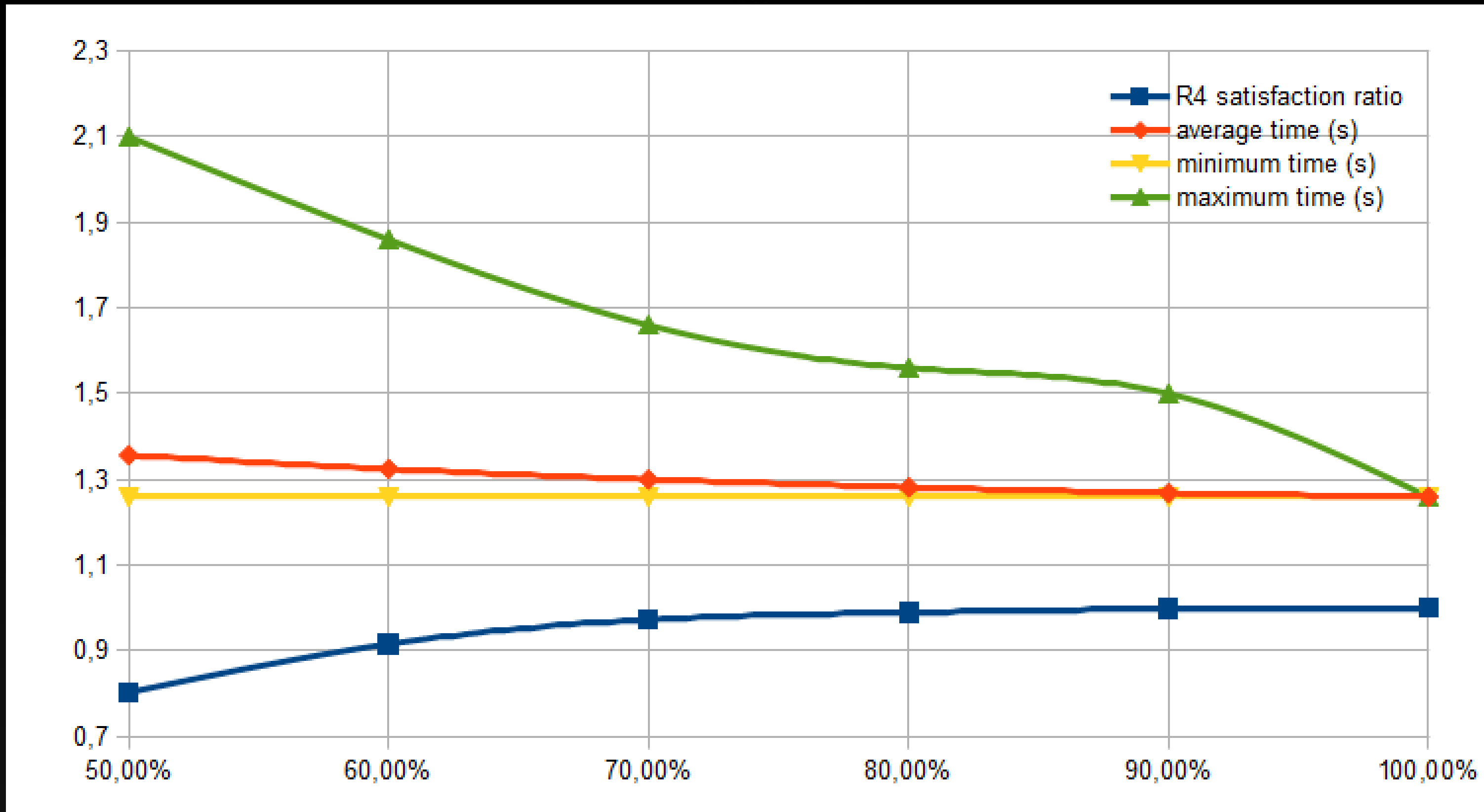
Executable requirements




```
processdef R4Observer (
   in bool : pilotHasPressedTheButton,
   in bool : parachuteDeployed)
{
   state OK {
   }
   on (pilotHasPressedTheButton)
     to EmergencyRequired;
   state EmergencyRequired {
   }
   after (1430ms) if (not parachuteDeployed)
     to Fail;
   state Fail {
     /* println("R4 FAILED"); */
     assert(false);
   }
}
```

- ✓ **Actual max. latency** depends on the ground speed target, the minimum acceptable altitude, the weight of the UAS and the characteristics of the parachute (opening time, lift, etc)

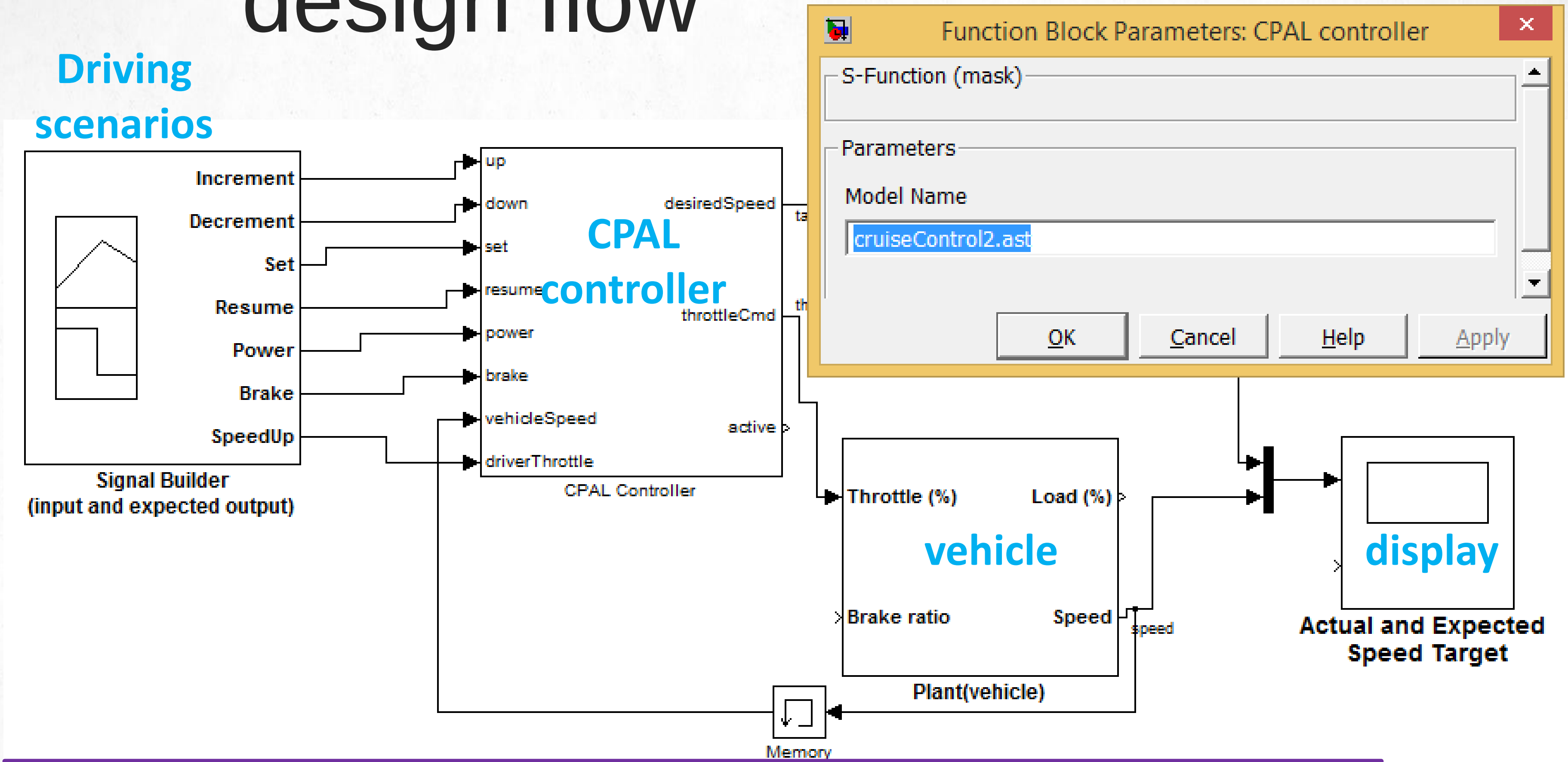
Model-based fault-injection



Time for the parachute to deploy (in seconds) and satisfaction of requirement R4 versus network quality ratio [11]

UC#3

Towards a timing augmented design flow



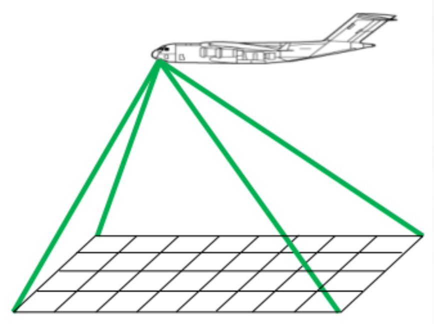
Ongoing research

- ✓ Timing accurate simulation & delays injected in the simulation
- ✓ Execution on target is timing-equivalent to simulation

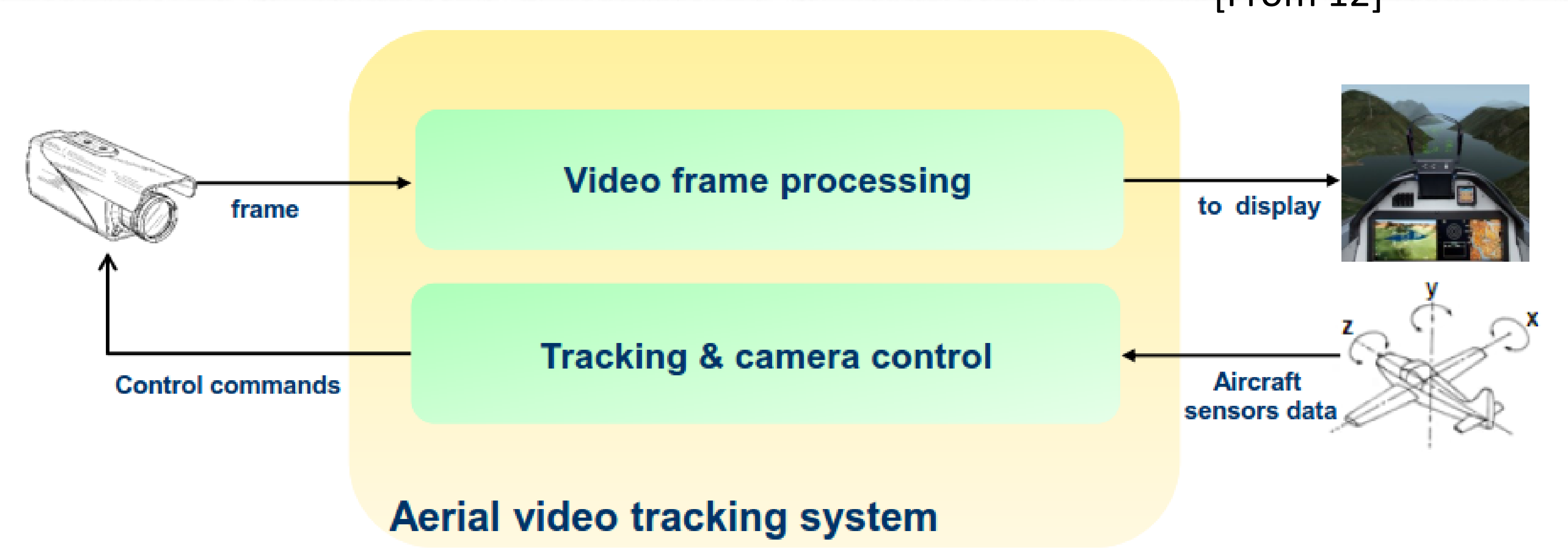
UC#4

Thales FMTV challenge [12,13]

Aerial video system to detect and track a moving object, e.g. a vehicle on a roadway
Challenge timing analysis community

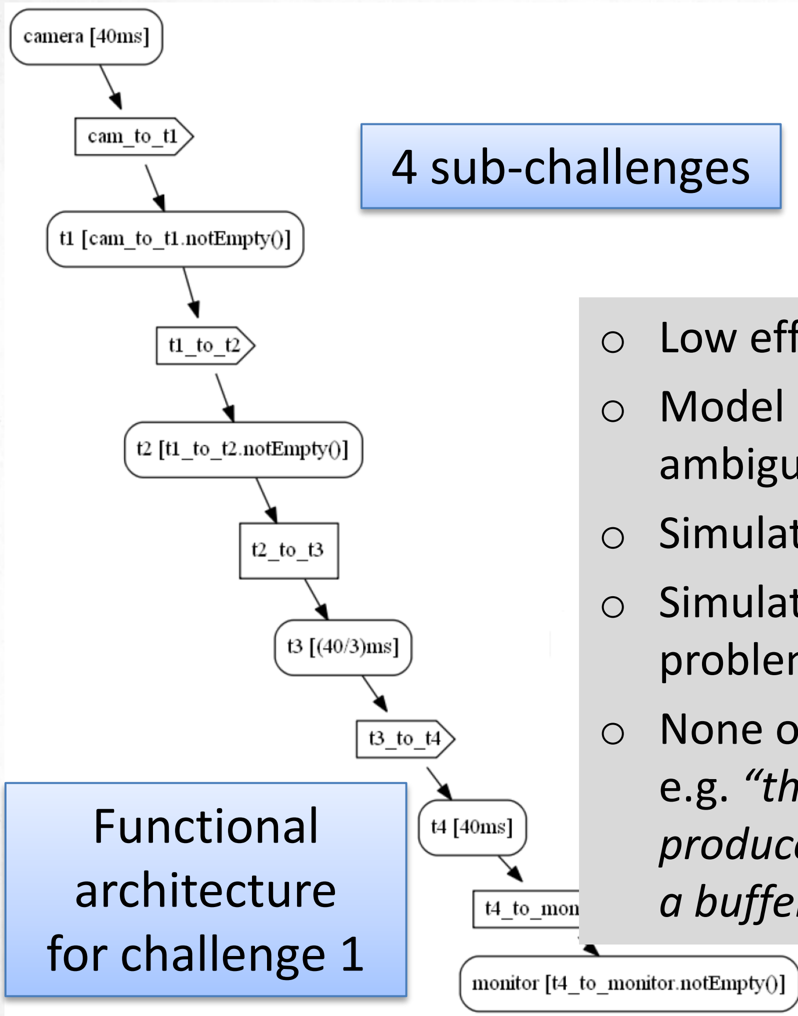


[From 12]



[From 12]

FMTV challenge in CPAL [13]



Functional architecture for challenge 1

4 sub-challenges

"Pen and paper"

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

- Low effort to model vs automata-based formalisms
- Model and graphical representation helped to highlight ambiguities
- Simulation helped to find errors in the analysis
- Simulation biased towards worst-case helped -> open problem
- None of the schedulability questions could be automated, e.g. *"the minimum time distance between two frames produced by the camera that will not reach the display, for a buffer size n = 3"*

Conclusion & future work

- Positive feedback about CPAL through use-cases
- Ongoing dev: annotation language to map I/Os to variables
- Quality of the tool chain and documentation will be key
- Development of a commercial offering
- Time-domain verification is low-risk, value-domain is open
- Timing equivalence between models in simulation and execution

Envisioned use-cases:

- ✓ **HW independence & scalable dependability**
- ✓ **Real-time IoT**
- ✓ **Adaptive and resilient CPS**

Thank you for your attention!

Want to give it a try? Binaries, code examples and playground at https://designcps.com

References

1. **N. Navet N., L. Fejoz L., L. Havet , S. Altmeyer, "[Lean Model-Driven Development through Model-Interpretation: the CPAL design flow](#)", Technical report from the University of Luxembourg, October 2015.**
2. A. Brown, "An Introduction to Model Driven Architecture – Part1: MDA and today's systems", IBM technical library, 2004.
3. T. Trew, "Creating Embedded Platforms with MDA: Where's the Sweet Spot", slides presented at ECMDA-FA, 2009.
4. T. A. Henzinger, "Two challenges in embedded systems design: predictability and robustness", Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 366(1881):3727–3736, 2008.
5. M. Antoni, "Formal validation method and tools for computerized interlocking system", 18th International Symposium on Formal Methods (FM 2012), Industry day, August 27-31, 2012.
6. S. Altmeyer, N. Navet, "Towards a declarative modeling and execution framework for real-time systems", To appear in the First IEEE Workshop on Declarative Programming for Real-Time and Cyber-Physical Systems, December 2015.
7. S. Altmeyer, N. Navet, "The case for FIFO scheduling", technical report from the University of Luxembourg, to appear, November 2015.
8. J. Seyler, N. Navet, L. Fejoz, "Insights on the Configuration and Performances of SOME/IP Service Discovery", in SAE International Journal of Passenger Cars- Electronic and Electrical Systems, 8(1), 124-129, 2015.

References Continued

9. J. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, "Formal Analysis of the Startup Delay of SOME/IP Service Discovery", Design, Automation and Test in Europe (DATE2015), Grenoble, France, March 13-15, 2015.
10. L. Ciarletta, L. Fejoz, A. Guenard, N. Navet, "Development of a safe CPS component: the hybrid parachute, a remote termination add-on improving safety of UAS",submitted to Embedded Real-Time Software and Systems (ERTS 2016), Toulouse, France, January 27-29, 2016.
11. F. Boniol, V. Wiels, "The landing gear system case study", pp1-18, Proc. ABZ 2014, 2014.
12. R. Henia, L. RIOUX, "Formal Methods for Timing Verification - The 2015 FMTV Challenge", 2014. https://waters2015.inria.fr/files/2014/11/FMTV-2015-Challenge.pdf
13. S. Altmeyer, N. Navet, L. Fejoz, "Using CPAL to model and validate the timing behaviour of embedded systems", 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Lund, Sweden, July 7, 2015.