The CPAL programming language
Design, Simulate, Execute
Embedded Systems

Lean Model-Driven Development through
Model-Interpretation

Nicolas Navet and Sebastian Altmeyer, University of Luxembourg
Loïc Fejoz and Lionel Havet, RealTime-at-Work

Embedded Real-Time Software and Systems (ERTS 2016)
Toulouse, France, January 28, 2016
Software has become the key to innovation

Amount of software is growing exponentially – what about productivity gains in software development?

Innovation increasingly relies on software

Software is disrupting complete industries

Every company has to learn to become a software company

Model-Driven Development is certainly a powerful enabler but ..

Programming environments still lack

✔ the high-level concepts: embedded system specific language abstractions

✔ automation features ("state the what, not the how") that would make them more productive

[inspired from posts at http://www.theenterprisearchitect.eu/]
CPAL is an embedded systems specific language

A. Model and program
   functional and non-functional concerns

B. Simulate
   possibly embedded within external tools such as RTaW-Pegase™ and Matlab/Simulink™

C. Execute
   bare metal or hosted by an OS - prototypes or real systems

A joint project of RealTime-at-Work and University of Luxembourg since 2012
5-steps of MBD

- Code only
- Code visualization
- Runtime Environment
- Model-centric
- Model only

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

Figure from [2] and [3]
Why a new programming language?

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

- Conceived to facilitate 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 brings benefits: monitoring at run-time, security, no distortion between model and code, WORA, etc.

Our view: major productivity and quality improvements still ahead of us through better programming languages and environments
A glance at the state-of-the-art

- With respect to **synchronous languages**?
  - Less demanding programming model: syntax close to mainstream languages, multiple I/Os per execution
  - 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
  - Would benefit from rich data-flow language of Prelude

- A large number of related (many discontinued) languages since the mid-80s: Pearl, Real-Time Euclid, C-extensions (real-time concurrent C, PRET-C, mbeddr), Labview RT module, RT and safety-critical Java, SCCharts, Papyrus-RT, etc → most are imperative (and not declarative like CPAL) in the non-functional domain
Outline

A. Selected highlights of the language

B. Processes are recurrent Finite State Machines

C. CPAL scheduling and task activation model

D. Timing-augmented design flow

E. Use-cases: automotive Ethernet simulation, Thales FMTV challenge, UAV programming
A few highlights of the language
Hello, world

processdef HelloWorld()
{
    state Main {
        println("Hello, world");
    }
}

process HelloWorld: aTask[100ms]();
Aim: intuitive and productive

Hello, world

```c
processdef MonitorProc(in uint8: aPort, out bool: alarm)
{
    const uint8: threshold = 30;
    state NormalMode{
        /* ... */
        on (aPort > threshold) {
            alarm = true;
        } to AlarmMode;
    }
    state AlarmMode {
        /* ... */
    }
    after (2s) if (aPort < threshold) to NormalMode;
}
```

FSM embedded in the process

NormalMode

on (aPort > threshold) after (2s) if (aPort < threshold)

AlarmMode
Processes: recurring activities whose logic is described as Finite State Machine
Finite-state Machines to describe the logic of a process

Code both in states and transitions

Timed transition and condition

Timed transition

Boolean condition
A process is periodically activated

One “step” of execution of the FSM

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

A transition can be fired?

Yes

Execute transition code
Move to next state

No

Stay in current state

Execute common code

Execute state-specific code

Wait until period has elapsed

_activation condition met or none?

No

Yes

Activate condition met or none?
Process introspection

```plaintext
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 and detect abnormal events at run-time.
CPAL scheduling and task activation model
CPAL’s 2 Execution Modes

**Simulation mode**
- Execution is as fast as possible (e.g. periods are not respected)
- Code executed in zero time – except if stated otherwise with timing annotations
- CPAL interpreter is hosted by an OS
- No access to real I/Os

**Real-Time mode**
- Real-time execution
- Code (instructions, read/write I/Os) takes time to execute – depends on the platform
- CPAL can be executed on bare hardware or hosted by an OS

**Overhead data on Freescale FRDM-K64F:**
- max. activation jitter: 40us
- timer interrupt: 0.6us
- context switch overhead: 2us
Vision behind CPAL

Timing equivalence needed depends on the application, can be e.g. 1) full determinism 2) order-preserving for observable events, or 3) deadline constraints met.

In CPAL current release, execution order of processes remains the same in simulation and in real-time mode.
Simulating execution times

Timing annotations can be derived by built-in monitoring facilities and are respected by the simulator
Activation conditions (aka “guarded executions”) are for implementing functioning modes and executing event-triggered activities.
CPAL scheduling model

- The choice of **non-preemptive scheduling**:
  - No context-switch + no cache related preemption delays (CRPD) on the WCET + less memory usage
  - No shared resources, easier to validate, less timing variability
  - But .. reduced ability to meet tight deadline constraints

- Currently **FIFO** policy is available:
  - Enforce **event-order determinism**
  - Work-conserving unlike static cyclic scheduling

- Built-in support for WCET measurements at run-time

- Planed to support partitioned multi-processor scheduling
Declaring timing correctness: designer states the “what”, not the “how”, environment does the rest

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

Allocate the models to the processing units

“Scheduler synthesis”

Ideas discussed in [6], implementation ongoing
Use-Cases
**Simulation: Some/IP SD**

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

Max analysis: 4.005ms
Max simulation: 3.98ms
Developing CPS: a smart parachute for UAV [10]

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

Communication

- uplink
- rcp_xbeeTask [50ms]
  - rcp_emergencyCommand
  - downlink
  - rcp_modeTask [50ms]
  - rcp_emergencyActivated
    - rcp_uiTask [200ms]
      - rcp_inEmergencyLED
      - rcp_powerLED
    - rcp_hwTask [20ms]
      - rcp_ic
      - rcp_powerSwitch
      - rcp_servo

On-board module

UI

HW control
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]
Towards a timing augmented design flow

Driving scenarios

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

Ongoing research

UC#3
Thales FMTV challenge \cite{12,13}

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

Aerial video tracking system

- Video frame processing
- Tracking & camera control

\[\text{Control commands} \rightarrow \text{Video frame processing} \rightarrow \text{Tracking & camera control} \rightarrow \text{to display} \]

\[\text{Aircraft sensors data} \]

\[\text{frame} \]

\[\text{[From 12]}\]
FMTV challenge in CPAL [13]

4 sub-challenges

- 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 $n = 3$”

“Pen and paper”

<table>
<thead>
<tr>
<th>Description</th>
<th>Simulation</th>
<th>Scheduling Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>1B</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>2A</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>2B</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>

Functional architecture for challenge 1

www.designcps.com
Conclusion & ongoing work

- CPAL: an interpreted language on a **time-triggered execution engine** - imperative programming in the functional domain - declarative programming in the non-functional domain
- Positive feedback about CPAL through industrial use-cases and teaching
- Code generation feasible for higher performance - hook to native code too
- **Objectives:** timing equivalence between models in simulation and execution / SILx for the execution engine

**Envisioned use-cases for the execution engine:**
- ✔ UAV and robotics
- ✔ Real-time IoT
- ✔ Adaptive and resilient CPS

*CPAL is free to use for academics (research works and industrial projects), Extensions to the language and toolset are welcome*
Thank you for your attention!

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


