Applications of the CPAL language to model, simulate and program Cyber-Physical Systems

Loïc FEJOZ
RealTime-at-Work (RTaW), France
loic.fejoz@realtimeatwork.com

Nicolas NAVET, Sakthivel SUNDHARAM, Sebastian ALTMEYER
University of Luxembourg, Luxembourg
firstname.lastname@uni.lu

***Abstract*— CPAL is a new language to model, simulate, verify and program Cyber-Physical Systems (CPS). CPAL serves to describe both the functional behaviour of activities (i.e., 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 flows among the functions). CPAL is meant to support two use-cases. Firstly, CPAL is a development and design-space exploration environment for CPS with main features being the formal description, the editing, graphical representation and simulation of CPS models. Secondly, CPAL is a real-time execution platform. The vision behind CPAL is that a model is executed and verified in simulation mode on a workstation and the same model can be later run on an embedded board with a timing-equivalent run-time behaviour. The design and development of CPAL have been organized around a set of realistic case-studies that will be demonstrated during the demo session.**

I. MODEL AS THE CODE

CPAL has been initially inspired by the success of three interpretation-based runtime environments, successfully certified at the highest criticality levels and deployed at large scale in railway interlocking systems over the last 20 years at SNCF and RATP in France, and in UK and other countries through the Westlock interlocking system from Westingshouse. Surprisingly to the best of our knowledge, except above applications and some industrial automation (PLCs) *model interpretation* has not been widely explored, albeit it possesses a number of key advantages such as: the model can be directly uploaded on the target, there is no difference between the model and the code, the total software size is greatly reduced both off-line and on the target, hardware independence is ensured, etc.

CPAL supports two types of model interpretation: the direct interpretation of the design models on an interpretation engine running on top of the hardware, called "bare-metal model interpretation" (BMMI), and the interpretation on top of an OS. The latter is less predictable from a timing point of view but more convenient for development and experimentations.

CPAL and its associated tools are jointly developed by our research group at the University of Luxembourg and the company RTaW since 2012. The CPAL documentation, graphical editor and execution engine for Windows, Linux, embedded Linux, and Raspberry Pi are freely available for all uses at http://www.designcps.com. A BMMI port of CPAL is available for Freescale FRDM-K64F boards.

II. CPAL: PROVIDING HIGH-LEVEL ABSTRACTIONS FOR EMBEDDED SYSTEMS

Figure 1, shows that Model-Driven Development is an enabling technology to fill the programming languages gaps. But still existing languages lack the high-level abstractions and automation features that would make them more productive. In addition, the design and development of embedded systems, especially ones with dependability constraints, necessitates the use of many best practices. None of the programming languages we are aware of are well suited to make the development of safe and provably correct embedded systems as quick and easy as possible.

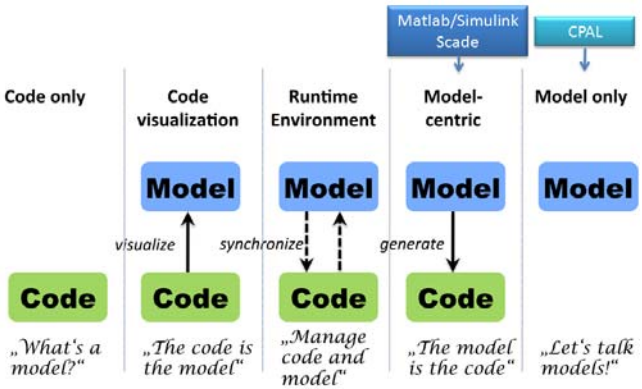

Figure 1: Spectrum of Model-Based Design approaches (core of the figure from [Br04]).

The main requirement when designing CPAL was to natively provide the high-level abstractions which are (i) familiar in the domain of embedded systems, and (ii) needed to express in an unambiguous and concise manner domain specific patterns of functional behaviors as well as non-functional properties. A *process* denotes the core language entity to implement a recurrent activity with its own dynamic. A process is automatically activated at a specified rate, with the optional requirement that a specific logical condition is fulfilled to execute (this is called guarded execution). CPAL processes are classically referred to as tasks, runnables or threads in other contexts.

CPAL provides the programmers with high-level abstractions well suited for the domain of CPS such as

- Real-time scheduling mechanisms: processes are activated with a user-defined period, possibly with an

offset relationship with each other, and additional execution conditions such as for instance the occurrence of some external events.

- Finite State Machines (FSM): the logic of a process can be defined as a Finite State Machine (FSM) based on Mode-Automata.
- Communication channels to support data flow exchanges between processes, and reading/writing to hardware ports.
- Introspection mechanisms that enable processes to query at run-time their execution characteristics such as their activation rate and activation jitters.

A key objective behind CPAL is to let designer state the permissible timing behavior of the system in a declarative manner while a system synthesis step involving both analysis and optimization then generates a scheduling solution which at run-time is enforced by the execution environment.

III. Demonstration of CPAL use-cases

CPAL [Na16] supports several use-cases discussed below and that will be demonstrated during the demonstration session.

A. *High-level programming language for network simulation environments*

CPAL can serve to describe the functional behavior of applications and high-level protocol layers. A CPAL model is for instance used in [Se15] to simulate the SOME/IP Service Discovery protocol in a Daimler Car's prototype network. The model hands over the frames once created to the simulation kernel of RTaW-Pegase, a communication architecture performance analysis tool from RTaW. Interestingly, the same CPAL simulation model can be executed with no changes on an embedded target or a workstation to experiment on a test-bed later in the design process.

This use-case will be briefly demonstrated through a CPAL model that transmits video streams with different coding standards (raw video, H.263) with segmentation an automotive Ethernet network.

B. *Modeling and simulation language for Design Space Exploration*

CPAL is meant to support the formal description, the editing, graphical representation and simulation of cyber-physical systems. It can be used in its own development environment, like done for the FMTV Challenge [Al15], or within Matlab/Simulink to implement the controller, as done for the landing gear case-study [Bo14]. The simulation models can be executed in real-time (i.e., activation periods are respected) or as fast as possible in simulation mode.

This use-case will be briefly illustrated on the FMTV Challenge 2015, highlighting also the limits of what can be automated.

C. *Real-time execution engine*

The intention of CPAL is to provide not only a modeling language, but also an interpreter which ensures equivalence between the simulated behavior of the model and the behavior on the execution platform, both in the functional and the temporal domain.

As in Figure 2, this use-case will be demonstrated on the "smart parachute", a remote termination add-on component improving safety of UAS [Ci16] developed in partnership with the company Alérion. The parachute is controlled by a CPAL program, on top of the bare-metal interpreter, executing on a Freescale FRDM-K64F board.

Figure 2: From simulation to field test

D. *CPAL for teaching and research*

CPAL has been used for teaching Model-Based Design (MBD) for embedded systems since 2012 at the University of Luxembourg at the 3rd year Bachelor level. Practicals include programming a capsule coffee machine, a simplified programmable floor robot and elevator control system, etc. Our experience has been positive in terms of how fast students have been able to work autonomously on the development of the system. Indeed, most students are able to master the language within a few hours. In addition to the simplicity of the language, the free availability of the tools, the on-line examples and the CPAL-Playground facilitate the learning process. CPAL is also used in the experiments of the research conducted at the University of Luxembourg on timing-aware Model-Driven Engineering. We will present small case-studies used in teaching and research outcomes based on CPAL.

References

[Al15a] S. Altmeyer, N. Navet, L. Fejoz, "Using CPAL to model and validate the timing behaviour of embedded systems", WATERS Workshop, July 2015.

[Al15b] S. Altmeyer, N. Navet, "Towards a declarative modeling and execution framework for real-time systems", First IEEE Workshop on Declarative Programming for Real-Time and Cyber-Physical Systems, San-Antonio, USA, December 1, 2015

[Bo14] F. Boniol, V. Wiels, "The landing gear system case study", pp1-18, Proc. ABZ 2014, 2014.

[Br04] A. Brown, "An Introduction to Model Driven Architecture – Part1: MDA and today's systems", IBM technical library, 2004.

[Ci16] 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", to appear at ERTSS2016, Toulouse, January 2016.

[Na16] N. Navet, L. Fejoz, L. Havet, S. Altmeyer, "Lean Model-Driven Development through Model-Interpretation: the CPAL design flow", Embedded Real-Time Software and Systems (ERTS 2016), Toulouse, France, January 27-29, 2016.

[Se15] J. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, "Formal Analysis of the Startup Delay of SOME/IP Service Discovery", DATE 2015, Grenoble, France, March 9-13, 2015.