MARTE/CCSL+TimeSquare+K-Passa:

A design platform using formal MoCCs for

embedded Model-based engineering

C. André, J. Boucaron, A. Coadou, J. DeAntoni, B. Ferrero, F. Mallet, R. de Simone
EPI Aoste
INRIA Méditerranée & UMR CNRS I3S & Université de Nice Sophia Antipolis
Sophia Antipolis, FRANCE
{firstname.surname}@sophia.inria.fr

***Abstract*—This extended abstract discusses our on-going work to build a design platform for embedded systems. To cover the whole design flow we start with a UML description annotated with MARTE. TimeSquare can edit, analyse and execute these UML models according to the operational semantics of CCSL. Then, when restricting to specific models of computations, we can use efficient static analysis techniques implemented in K-Passa to optimize these models according to several criteria such as latency, throughput or interconnect buffer sizes. The analysis results are transformed back into the UML model.**

Keywords-Formal MoCC; Process network; MARTE, MDE

I. CONTEXT

We are acknowledging the fact that, in the past, most modelling environments in the field of embedded and distributed systems have been based on a few conceptual diagrammatic representations.

Dynamics/behavioural parts:

- **state-based formalisms** are encompassed in Process Algebras (CSP, CCS, …), Statecharts and numerous variants, and many sorts of concurrent automata networks, with synchronization handshake mechanisms;
- **activity-based formalisms** originate rather from Petri Nets and specific subclasses, leading to Data Flow Process Networks of various kinds (Kahn Process Networks [1]are worth noticing as a data-flow/activity-based modelling style with channel/place links, but also local control-flow state-based modelling).

Structural parts:

- **component formalisms** exposing ports for communication and construction of large systems by hierarchical combination of components, with required/offered interfaces and assume/guarantee properties.

The role of true data values and true computations are usually downplayed in these modelling frameworks, and deported to an external host language (C/C++) for encoding. Only the trace of data operations that may impact the control branching structure is meant to be preserved, through crude abstraction into discrete domains usually.

This is common ground, and despite many subtle differences in design expressivity, most embedded design and analysis formalisms are based, at least pictorially, on these types of modelling diagrams. Of course, more serious differences stem when dealing with precise operational semantics, the synchronous/asynchronous spectrum, and also timing/schedulability/performance analysis. Here we view two distinct trends:

Some theories (and the related methodologies and environments) are adopting a view of logical functional time. Then, the main time unit is the transition/reaction step, global or local. Synchronization primitives are used in the design to introduce constraints between the local activation time(s) of distinct components. Any legible execution of the global system must obey these constraints, thereby scheduling and synchronizing the components and their interactions/communications. In the context of Process Network conflict-free models (such as Marked Graphs or Synchronous Data Flow [2] models for instance), one can devise statically a regular global scheduling so that, in addition to their original asynchronous semantics, the very same models can be endowed with a fully synchronous semantics that optimize many parameters (throughput, channel buffer size…). In these lines of models one can also cite the synchronous reactive formalisms [3] (Esterel, Lustre, Signal...), StateCharts, and HDLs (Hardware Description Languages) at RTL level based on a single clock pulse;

Other theories are based on a more physical time, with extra functional interpretations that do not truly directly translate into syntactic instructions in the control flow, but rather as timeliness properties that should be reached at execution, usually on charge of implementation mechanisms and scheduling techniques that are beyond the scope of modelling representation, and can only be approximated or mimicked for early analysis at modelling stage.

II. MARTE, ITS TIME MODEL AND CCSL

We have been active participants and promoters of the recent official OMG UML profile for Modelling and Analysis of Real-Time Embedded systems (MARTE) [4]. We concentrated in the definition of a Time Model subprofile [5], which allows a rich-but-well-defined variety of time notions (logical/physical, discrete/dense…) to be applied as time bases and clocks onto the otherwise classical main UML diagram views. In a profile annex, we proposed a Language for expressing Clock Constraint Specifications (CCSL) [6].

With this consistent set of constraints one can state for instance that a clock is subsampling another one on a given regular pattern (see example Figure 1), that two clocks have bounded jitters, and so on. Most importantly, the constraints have precise formal semantics, so that the solution space of possible legible schedules can be searched algorithmically or interactively.

The ultimate goal of CCSL is to provide a means to define formally timed Models of Computations and Communications (MoCCs), by providing a syntactic way to describe semantic relations between tied behaviours. This is in part inspired from the theory of tagged systems of Lee and Sangiovanni-Vincentelli [7], but also of previous works on Structural Operational Semantics (SOS) definition rules typed structural elements and relations amongst them are expressed as CCSL constraints. These very same clocks are then applied to behavioural elements (via MARTE stereotypes) and define their activation and terminating conditions or any kind of time constraints. Instants of CCSL clocks represent an action starting or finishing, a message being sent or received, a behaviour being called or terminating, entering or exiting a state, … Instants, whether linked to physical time or not, also serve as references to apply duration or time constraints. Like synchronous languages, CCSL deals with multi-form time and these durations or dates are expressed in number of ticks of a given clock.

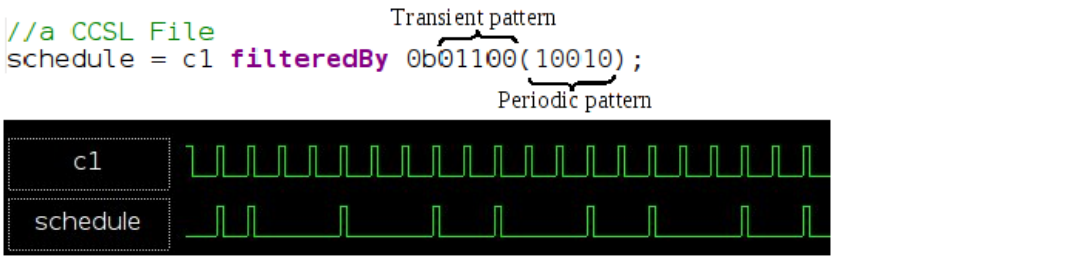

Figure 1: A CCSL subsampling specification on a specific pattern and its simulated timing diagram in TimeSquare

III. TIMESQUARE AND K-PASSA

TimeSquare [8] is our software environment dedicated to the resolution of CCSL constraints and computation of partial solutions. TimeSquare has four main features: 1) definition/modelling of CCSL user-defined libraries that encapsulate the MoCCs, 2) specification/modelling of a CCSL model and its application to a specific UML-based or DSL model, 3) simulation of MoCCs and generation of a corresponding trace model, 4) based on a trace model, displaying and exploring the augmented timing diagram, animating UML-based model and storing the scheduling result inside the model and sequence diagrams. TimeSquare is provided as a set of Eclipse plug-ins. A detailed description of TimeSquare features, examples, and video demonstrations are available at its website.

We have developed a second tool, named **K-Passa** [9], which computes efficient static schedules in the case of specific Process Network MoCCs. It currently handles Marked Graphs, Synchronous Data Flow Process Networks, Latency-Insensitive Designs and K-Periodically Routed Graphs as input MoCCs.

The *efficient* schedules are computed according to several criteria such as latency, throughput or interconnect buffer sizes considered for optimization. K-Passa also provides useful information computed on the models, such as correctness checks of safety and liveness for instance. The tool is available as both a standalone application and a library. The standalone application allows creating and editing diagrams using a full featured GUI, to conduct experiments to ensure correctness, to run optimizations and simulations. The library allows building co-simulation models for SystemC/C++ and Java.

TimeSquare and K-Passa can be coupled in two ways: first the UML editing of MoCCs with adequate stereotype could allow to specify input models for K-Passa analysis in the TimeSquare environment (work-in-progress); second, the schedule solutions computed in K-Passa can (already) be re-injected into CCSL format and used in TimeSquare (augmented timing diagrams, UML model animation, etc).

[1] Gilles Kahn. “The semantics of a simple language for parallel programming”. Information Processing, 471-475, 1974.

[2] E. A. Lee and D.G. Messerschmitt: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Computers 36(1) 24–35, 1987.

[3] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, Robert de Simone: “The synchronous languages 12 years later”. Proc. of the IEEE 91(1): 64-83, 2003.

[4] The ProMARTE Consortium: “UML Profile for MARTE”, beta 2. Object Management Group. number: ptc/08-06-08, 2008.

[5] Charles André, Frédéric Mallet and Robert de Simone: “Modeling time(s)”. MoDELS. Volume 4735 of LNCS., Springer, 559–573, 2007.

[6] Frédéric Mallet, Charles André, Robert de Simone. “CCSL: specifying clock constraints with UML/Marte”. ISSE, 4(3):309-314, 2008.

[7] E. A. Lee and A. L. Sangiovanni-Vincentelli. “A framework for comparing models of computation”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17(12):1217–1229, December 1998.

[8] Charles André, Benoît Ferrero, Frédéric Mallet, Robert de Simone. “TimeSquare: a software environment for timed systems”. Date’09, University Booth, April 2009.

[9] Julien Boucaron, Benoît Ferrero, Jean-Vivien Millo, Robert de Simone. “Statically Scheduled Process networks”. INRIA RR-6289, 2007