Building a Hybrid Systems Modeler from Synchronous Language Principles

(Invited Talk)

Marc Pouzet

DI, École normale supérieure,
45 rue d'Ulm, 75230 Paris cedex 05

Marc.Pouzet@ens.fr

ABSTRACT

Hybrid systems modeling languages are widely used in the development of embedded systems. Two representatives are Simulink/Stateflow,[1] which combines Ordinary Differential Equations (ODEs), data-flow and difference equations, hierarchical automata *à la* StateCharts [13], and imperative features; and the Modelica language [17][2] based on DAEs with features for modeling discrete components. Ptolemy II[3] is another example in which several models of computation are combined [14].

While the formal verification of abstract hybrid systems has been studied extensively [8], many language related issues remain to be addressed. In this regard, we share the viewpoint of Lee and Zheng that hybrid modeling languages are best viewed as *programming languages that happen to have a hybrid systems semantics* [15, 16]. This raises important questions related to language design, semantics, and implementation, to producing reliable simulation runs efficiently, and also to the generation of provably equivalent embedded target code. While sequential code generation in hybrid modeling tools is routinely used for efficient simulation, it is used infrequently or not at all to produce target embedded code in critical applications subject to strong safety requirements. This results in a break in the development chain: parts of applications must be rewritten into sequential code and properties verified of the source model must essentially be reverified of the target code.

Sequential code generation from a synchronous language like Lustre [11] has been studied in detail [12]. It can be formalized as a series of source-to-source transformations—that eliminate high level constructs like hierarchical automata [10]—into a generic intermediate representation for transition functions, which is in turn transformed into C code [5]. This approach, initiated in Lucid Synchrone [18], is implemented in the Scade Suite KCG code generator of Scade 6[4], which is used for developing various critical applications.

Yet synchronous languages only manipulate discrete-time signals. Their expressiveness is limited to ensure important safety properties like determinacy, execution in bounded time and space, and simple, traceable code generation. Their cyclic execution requires minimal run-time support and does not suffer from the complications that accompany numerical solvers of ODEs. Conversely, a hybrid modeling language allows discrete and continuous time behaviors to interact. But this interaction is not constrained enough nor specified with adequate precision in tools like Simulink/Stateflow which results in semantic pitfalls and bugs [9, 4, 1]. A precise description of all the compilation steps, that is, the actual implemented semantics, is mandatory in safety critical development processes where target code must be trustworthy. Our goal, in short, is to increase the expressiveness of synchronous languages without sacrificing any confidence in their compilation.

In previous work, we introduced a novel approach for the design and implementation of a hybrid modeling language that reuses synchronous language principles and an existing compiler infrastructure. We introduced an ideal synchronous semantics based on non standard analysis [4] for a Lustre-like language with ODEs [3], and extended the kernel language with hierarchical automata [2] and a modular causality analysis [1]. These results form the foundation of Zélus [7] [5] and were validated inside the industrial Scade Suite KCG code generator (Release 6.4, 2014) developed at Esterel-Technologies/ANSYS [6].

In this talk, I summarize the ongoing work on Zélus and the way it has been applied to the Scade Suite KCG code generator. In the latter, it was possible to reuse the existing infrastructure entirely—namely, static typing, causality and initialization analyses, intermediate languages, and various compiler optimizations—with minimal modifications. The proposed language extension is conservative in that regular synchronous functions are compiled as before—the same synchronous code is used both for simulation and for ex-

[1] http://mathworks.org/simulink

[2] https://www.modelica.org

[3] http://ptolemy.eecs.berkeley.edu/ptolemyII/

[4] http://www.esterel-technologies.com/products/scade-suite/

[5] zelus.di.ens.fr

ecution on target platforms. It also shows the versatility of the KCG infrastructure based on successive rewritings. The precise definition of all compilation steps, built on the proven compiler infrastructure of a synchronous language avoids the rewriting of control software and may also increase confidence in simulation results.

This work results from a collaboration with Albert Benveniste, Benoit Caillaud (INRIA, Rennes), Timothy Bourke (INRIA, Paris-Rocquencourt), Jean-Louis Colaço, Bruno Pagano and Cédric Pasteur (Esterel-Technologies/ANSYS).

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; D.3.2 [**Language classifications**]: Data-flow languages; D.3.4 [**Processors**]: Code generation, Compilers; I.6.2 [**Simulation and Modeling**]: Simulation Languages

General Terms

Algorithms, Languages, Theory

Keywords

Real-time systems; Hybrid systems; Synchronous languages; Block diagrams; Compilation; Semantics; Type systems

1. REFERENCES

[1] Albert Benveniste, Timothy Bourke, Benoit Caillaud, Bruno Pagano, and Marc Pouzet. A Type-based Analysis of Causality Loops in Hybrid Systems Modelers. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, Berlin, Germany, April 15–17 2014. ACM.

[2] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. A Hybrid Synchronous Language with Hierarchical Automata: Static Typing and Translation to Synchronous Code. In *ACM SIGPLAN/SIGBED Conference on Embedded Software (EMSOFT'11)*, Taipei, Taiwan, October 2011.

[3] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Divide and recycle: types and compilation for a hybrid synchronous language. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES'11)*, Chicago, USA, April 2011.

[4] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Non-Standard Semantics of Hybrid Systems Modelers. *Journal of Computer and System Sciences (JCSS)*, 78(3):877–910, May 2012. Special issue in honor of Amir Pnueli.

[5] Darek Biernacki, Jean-Louis Colaco, Grégoire Hamon, and Marc Pouzet. Clock-directed Modular Code Generation of Synchronous Data-flow Languages. In *ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, Tucson, Arizona, June 2008.

[6] Timothy Bourke, Jean-Louis Colaço, Bruno Pagano, Cédric Pasteur, and Marc Pouzet. A Synchronous-based Code Generator For Explicit Hybrid Systems Languages. In *International Conference on Compiler Construction (CC)*, LNCS, London, UK, April 11-18 2015.

[7] Timothy Bourke and Marc Pouzet. Zélus, a Synchronous Language with ODEs. In *International Conference on Hybrid Systems: Computation and Control (HSCC 2013)*, Philadelphia, USA, April 8–11 2013. ACM.

[8] Luca Carloni, Maria D. Di Benedetto, Alessandro Pinto, and Alberto Sangiovanni-Vincentelli. Modeling Techniques, Programming Languages, Design Toolsets and Interchange Formats for Hybrid Systems. Technical report, IST-2001-38314 WPHS, Columbus Project, March 19 2004.

[9] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating Discrete-Time Simulink to Lustre. *ACM Transactions on Embedded Computing Systems*, 2005. Special Issue on Embedded Software.

[10] Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. A Conservative Extension of Synchronous Data-flow with State Machines. In *ACM International Conference on Embedded Software (EMSOFT'05)*, Jersey city, New Jersey, USA, September 2005.

[11] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[12] N. Halbwachs, P. Raymond, and C. Ratel. Generating efficient code from data-flow programs. In *Third International Symposium on Programming Language Implementation and Logic Programming*, Passau (Germany), August 1991.

[13] D. Harel. StateCharts: a Visual Approach to Complex Systems. *Science of Computer Programming*, 8-3:231–275, 1987.

[14] Edward A. Lee and Alberto Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on CAD*, 17(12), December 1998.

[15] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)*, volume 3414, Zurich, Switzerland, March, 9-11 2005. LNCS.

[16] Edward A. Lee and Haiyang Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT*, Salzburg, Austria, September 30-October 3 2007.

[17] Modelica. `http://www.modelica.org/`, 2015.

[18] Marc Pouzet. *Lucid Synchrone, version 3. Tutorial and reference manual.* Université Paris-Sud, LRI, April 2006.