ABSTRACT
Hybrid systems modeling languages are widely used in the
development of embedded systems. Two representatives are
Simulink/Stateflow, which combines Ordinary Differential
Equations (ODEs), data-flow and difference equations, hier-
erarchical automata à la StateCharts [13], and imperative
features; and the Modelica language [17] based on DAEs
with features for modeling discrete components. Ptolemy
II is another example in which several models of computa-
tion 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 view-
point 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 imple-
mentation, to producing reliable simulation runs efficiently,
and also to the generation of provably equivalent embed-
ded 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 require-
ments. 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 func-
tions, which is in turn transformed into C code [5]. This ap-
proach, initiated in Lucid Synchrone [18], is implemented
in the SCADE Suite KCG code generator of SCADE 6, 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 al-
 lows 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 de-
velopment processes where target code must be trustworth-
y. 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 exist-
ing compiler infrastructure. We introduced an ideal syn-
chronous semantics based on non standard analysis [4] for a
Lustre-like language with ODEs [3], and extended the ker-
nel language with hierarchical automata [2] and a modular
causality analysis [1]. These results form the foundation of
ZÉLUS [7] 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 regu-
lar synchronous functions are compiled as before—the same
synchronous code is used both for simulation and for ex-

1http://mathworks.org/simulink
2https://www.modelica.org
3http://ptolemy.eecs.berkeley.edu/ptolemyII/
4http://www.esterel-technologies.com/products/scade-suite/
5zelus.di.ens.fr
execution on target platforms. It also shows the versatility of the KCG infrastructure based on successive reworkings. 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, Benoît Caillaud (INRIA, Rennes), Timothy Bourke (INRIA, Paris-Rocquencourt), Jean-Louis Colaco, 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


