

# Synchronous Programming

Edited by

Stephen A. Edwards<sup>1</sup>, Alain Girault<sup>2</sup>, and Klaus Schneider<sup>3</sup>

1 Columbia University, US, [sedwards@cs.columbia.edu](mailto:sedwards@cs.columbia.edu)

2 INRIA, FR, [alain.girault@inria.fr](mailto:alain.girault@inria.fr)

3 TU Kaiserslautern, DE, [klaus.schneider@informatik.uni-kl.de](mailto:klaus.schneider@informatik.uni-kl.de)

---

## Abstract

Synchronous programming languages are programming languages with an abstract (logical) notion of time: The execution of such programs is divided into discrete reaction steps, and in each of these reactions steps, the program reads new inputs and reacts by computing corresponding outputs of the considered reaction step. The programs are called synchronous because all outputs are computed together in zero time within a step and because parallel components synchronize their reaction steps by the semantics of the languages. For this reason, the synchronous composition is deterministic, which is a great advantage concerning predictability, verification of system design, and embedded code generation. Starting with the definition of the classic synchronous languages Esterel, Lustre and Signal in the late 1980's, the research during the past 20 years was very fruitful and led to new languages, compilation techniques, software and hardware architectures, as well as extensions, transformations, and interfaces to other models of computations, in particular to asynchronous and hybrid systems.

This report is a summary of the Dagstuhl Seminar 13471 "Synchronous Programming", which took place during November 18-22, 2013, and which was the 20<sup>th</sup> edition of the yearly workshop of the synchronous programming community. The report contains the abstracts of the presentations given during the seminar in addition to the documents provided by the participants on the web pages of the seminar<sup>1</sup>.

**Seminar** 18.–22. November, 2013 – [www.dagstuhl.de/13471](http://www.dagstuhl.de/13471)

**1998 ACM Subject Classification** C.4 Performance of Systems, D.1.3 Concurrent Programming, D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification, D.3.3 Language Constructs and Features, D.4.7 Organization and Design, F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages

**Keywords and phrases** Synchronous Languages, Hybrid Systems, Formal Verification, Models of Computation, WCET Analysis, Embedded Systems

**Digital Object Identifier** 10.4230/DagRep.3.11.117

**Edited in cooperation with** Manuel Gesell

---

<sup>1</sup> See <http://www.dagstuhl.de/13471> for more information.



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Synchronous Programming, *Dagstuhl Reports*, Vol. 3, Issue 11, pp. 117–143

Editors: Stephen A. Edwards, Alain Girault, and Klaus Schneider



DAGSTUHL  
REPORTS

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Executive Summary

*Stephen A. Edwards*

*Alain Girault*

*Klaus Schneider*

License  Creative Commons BY 3.0 Unported license  
© Stephen A. Edwards, Alain Girault, and Klaus Schneider

### Model-based Design of Embedded Systems

In general, the *development of embedded systems* is a challenging task: Concerning the hardware platforms, developers have to cope with tight resource constraints, *heterogeneous and application-specific hardware architectures*, virtual prototypes, and many other difficulties during the design phases. Concerning the software side, several concurrent tasks are executed on the available hardware, either with or without the help of special operating systems, sometimes statically or dynamically scheduled to the available *hardware platforms*, and sometimes tightly coupled with the hardware platforms themselves (implementing memory barriers etc). Finally, many non-functional aspects have to be considered as well like the energy consumption, the *reliability*, and most important the *prediction of the worst-case computation times*. As many embedded systems are *real-time systems*, it is not sufficient to perform the right computations; in addition, the results have to be available at the right point of time to achieve the desired functionality. Besides, the direct interaction with other systems that often have a continuous behavior requires to consider *cyber-physical systems*. Since many embedded systems are used in safety-critical applications, incorrect or delayed behaviors are unacceptable, so that *formal verification* is often applied. Since, moreover, the development costs have to be minimized, new design flows that allow the development of safe and flexible embedded systems are of high interest.

For these reasons, *model-based design flows* became popular where one starts with an abstract model of the embedded system. Many languages are discussed for such model-based approaches, but most of them are based on only a few models of computation. A *model of computation* thereby defines *which, when and why an action* of the system takes place taking into account the *timeliness*, the *causality*, and the *concurrency* of the computations. Classic models of computation are *dataflow process networks*, where computations can take place as soon as sufficient input data is available, *synchronous systems*, which are triggered by clocks, *discrete-event based systems*, where each process is sensitive to the occurrence of a set of certain events, and *cyber-physical systems* whose behavior consists of discrete and continuous transitions (the latter are determined by differential equations).

It is not surprising that all models of computation have their advantages and disadvantages. For example, dataflow process networks can be naturally mapped to distributed systems and have a robust form of asynchronous concurrency provided that the nodes implement continuous functions (as required for Kahn networks). Synchronous systems are the perfect choice for implementing deterministic systems with predictable real-time behaviors on platforms having a local control (like clocks in digital hardware circuits). Discrete-event based systems are ideal for efficiently simulating systems, since the events directly trigger the next actions.

Many years of research were necessary to understand the above mentioned models of computation in depth to be able to develop corresponding programming languages, compilers and verification techniques. The synchronous programming community made substantial progress in this area: Today, the synchronous programming languages have precise formal

semantics which are supported by efficient compiler techniques. Moreover, synchronous languages provide high-level descriptions of real-time embedded systems so that all relevant requirements for a model-based design flow are fulfilled. There are also graphical versions of these textual languages, notably Safe State Machines (developed from Argos and SyncCharts), and there are commercial versions like SCADE. The SCADE tool provides a code generator certified against DO 178-B, which makes it particularly attractive for the aircraft sector.

Quoting Benveniste et al.: *Today, synchronous languages have been established as a technology of choice for modeling, specifying, validating, and implementing real-time embedded applications. The paradigm of synchrony has emerged as an engineer-friendly design method based on mathematically sound tools* [Proceedings of the IEEE, January 2003].

## Open Problems

Despite the incredible progress made in the past, even the combination of the classic synchronous languages Esterel, Lustre, and Signal is not yet fully understood. All these languages are based on the abstraction of physical time to a logical time, where each logical step of time may consist of finitely many executions of actions that are – at least in the programming model – executed in zero time. Such a logical step of the computation matches naturally with an interaction of a reactive system with its environment. However, looking at the details, one can observe that the semantics differ: for example, Lustre and Signal are not based on a single clock like Esterel, and while Esterel's and Lustre's semantics are operational and can therefore be defined by least fixpoints, Signal is rather declarative and requires a more complicated analysis before code generation.

Since different models of computation have different advantages and disadvantages, their *combination* becomes more and more important. This does also imply the *translation and communication between models of computations*. For example, so-called globally asynchronous, locally synchronous (GALS) systems have been developed, mixing both asynchronous and synchronous computations. For model-based designs starting from synchronous languages, special forms of synchronous systems have been defined in terms of the (weakly) endochronous systems. Intuitively, endochronous systems are synchronous systems that can determine from which input ports the values are expected for the next reaction step (and therefore they can derive the absence of other inputs, and they do not need the explicit knowledge of absence). For this reason, one can integrate endochronous systems in an asynchronous environment without destroying their synchronous behaviors.

Similar techniques are used for generating distributed systems from high-level descriptions (like synchronous programs) which lead, e.g., also to first approaches to multithreaded code generation from synchronous languages, which becomes more important due to the advent of multicore processors in embedded system design. More progress is needed and will likely be available in the near future in combining these different forms of discrete models of computations.

The combination of synchronous, endochronous, or asynchronous discrete systems with continuous behaviors to describe cyber-physical systems is still in its infancies. Of course, there are many languages for modeling, simulating, and even formally verifying these systems, but most of these languages lack of a formal semantics, and essentially none of them lends itself for a model-based design like synchronous languages. The generalization of the concepts of synchronous systems to polychronous systems, and even further to cyber-physical systems will be a challenge for future research.

## Results of the Seminar

The major goal of the seminar was therefore to allow researchers and practitioners in the field of models of computation and model-based design to discuss their different approaches. Desired results are new combinations of these techniques to form new language concepts and design flows that are able to choose the best suited language for particular components and that allow engineers the sound integration of synchronous and asynchronous, discrete and continuous, or event- and time-triggered systems. Besides this, still more research is required for further developing compilation techniques for future manycore processors, and even to develop special processors like the PRET architectures to obtain better estimated time bounds for the execution of programs.

The seminar proposed here aims at addressing all of these questions, building on a strong and active community and expanding its scope into relevant related fields, by inviting researchers prominent in model-based design, embedded real-time systems, mixed system modeling, models of computation, and distributed systems. The seminar was held in the tradition of the Synchronous Programming (SYNCHRON) workshops that are used as the yearly meeting place for the community in this exciting field. The SYNCHRON workshops started in 1994 at Schloss Dagstuhl, and we were proud to celebrate the 20<sup>th</sup> edition of the workshop from November 18–22, 2013 again in Schloss Dagstuhl. Previous editions of the SYNCHRON workshop were organized at the following locations:

- 2012:** Le Croisic, France – <http://synchron2012.inria.fr>
- 2011:** Dammarie-les-Lys – <http://synchron2011.di.ens.fr>
- 2010:** Fréjus, France – <http://www.artist-embedded.org/artist/Synchron-2010,2206.html>
- 2009:** Dagstuhl, Germany – <http://www.dagstuhl.de/09481>
- 2008:** Aussois, France – <http://synchron2008.lri.fr>
- 2007:** Bamberg, Germany
- 2006:** L’Alpe d’Huez, France – <http://www.inrialpes.fr/Synchron06/>
- 2005:** Qwara, Malta – <http://www.cs.um.edu.mt/~synchron05/>
- 2004:** Dagstuhl, Germany – <http://www.dagstuhl.de/04491>
- 2003:** Luminy, France – <http://www.verimag.imag.fr/PEOPLE/Nicolas.Halbwachs/SYNCHRON03/>
- 2002:** La Londe les Maures, France – <http://www-sop.inria.fr/tick/Synchron2002.html>
- 2001:** Dagstuhl, Germany – <http://www.dagstuhl.de/01491>
- 2000:** Saint-Nazaire, France
- 1999:** Hyères, France – <http://www-sop.inria.fr/meije/synchron99/location.htm>
- 1998:** Gandia, Spain
- 1997:** Roscoff, France
- 1996:** Dagstuhl, Germany – <http://www.dagstuhl.de/9650>
- 1995:** Luminy, France
- 1994:** Dagstuhl, Germany – <http://www.dagstuhl.de/9448>

During its 20 years of existence, the workshop has significantly evolved: its scope has grown to expand to many languages and techniques that are not classically synchronous, but have been substantially influenced by the synchronous languages’ attention to timing, mathematical rigor, and parallelism. Also, while many of the most senior synchronous language researchers are still active, many younger researchers have also entered the fray and have taken the field in new directions. We carefully selected the potential persons to be invited in that senior and junior researchers of the different branches mentioned above will participate the seminar.

This year, we had 44 participants where 23 came from France, 10 from Germany, 5 from the USA, 2 from Sweden, 2 from UK, one from Portugal and one even from Australia. The seminar had 33 presentations of about 45 minutes length with very active discussions<sup>2</sup>. The presentations can be clustered in typical research areas around synchronous languages like

- synchronous and asynchronous models of computation
- hybrid systems
- causality and other program analyses
- compilation techniques
- predictable software and hardware architectures

It was a pleasure to see that the synchronous programming community is still very active in these research fields and that even after 20 years of research, there are still more and more interesting and fruitful results to be discovered. The following sections contains short abstracts of the presentations of the seminar, and further documents were provided by many participants on the seminar's webpage.

February 2014, Albert Benveniste, Stephen A. Edwards, Alain Girault, and Klaus Schneider

---

<sup>2</sup> See <http://www.dagstuhl.de/schedules/13471.pdf> for the schedule.

## 2 Table of Contents

### Executive Summary

*Stephen A. Edwards, Alain Girault, and Klaus Schneider* . . . . . 118

### Overview of Talks

Desynchronization of Synchronous Systems

*Yu Bai* . . . . . 124

Representing Spatially Moving Entities using Time-Variant Topologies

*Fernando Barros* . . . . . 124

From Quasi-Synchrony to LTTA

*Guillaume Baudart* . . . . . 125

BPDF: A Statically Analyzable DataFlow Model with Integer and Boolean Parameters

*Vagelis Bebelis* . . . . . 125

Towards Discrete Controller Synthesis for the Reactive Adaptation of Autonomic Systems

*Nicolas Berthier* . . . . . 126

A slow afternoon chez PARKAS and a very fast fly (a fun talk)

*Timothy Bourke* . . . . . 126

Modelyze: Embedding Equation-Based DSLs

*David Broman* . . . . . 127

Index theory for Hybrid DAE Systems

*Benoit Caillaud* . . . . . 127

Functioning Hardware from Functional Specifications

*Stephen A. Edwards* . . . . . 128

Debugging and Compiler Bootstrapping with an Equation-Based Language Compiler

*Peter A. Fritzson* . . . . . 128

Interactive Verification of Cyber-physical Systems

*Manuel Gesell* . . . . . 129

Behavioral Equivalence of Transducers under a Fixed Protocol

*Dan R. Ghica* . . . . . 130

Timing Through Types

*Dan R. Ghica* . . . . . 130

Precise Timing Analysis for Direct-Mapped Caches

*Alain Girault* . . . . . 131

When the decreasing sequence fails... Improving fixpoint approximation in program analysis

*Nicolas Halbwachs* . . . . . 131

Modal Interface Automata

*Gerald Luetzgen* . . . . . 132

Safety Issues in MARTE/CCSL Specifications

*Frederic Mallet* . . . . . 132

|  |     |
|--|-----|
| In Search of a Physical Semantics of Boussinot’s Reactive Model<br><i>Louis Mandel</i> . . . . .   | 133 |
| From Synchronous to Timed Programming<br><i>Eleftherios Matsikoudis</i> . . . . .  | 133 |
| Berry-Constructive Programs are Sequentially Constructive, or: Synchronous Programming from a Scheduling Perspective<br><i>Michael Mendler</i> . . . . . | 135 |
| SCCharts – Sequentially Constructive Charts<br><i>Christian Motika</i> . . . . .   | 136 |
| A Tale of Two Semantics: Clocked Dimensions in a Multidimensional Language<br><i>John Plaice</i> . . . . .   | 137 |
| Integrated WCET estimation of multicore applications<br><i>Dumitru Potop-Butucaru</i> . . . . .  | 137 |
| A Causality Analysis for Hybrid Modelers<br><i>Marc Pouzet</i> . . . . .   | 138 |
| Timing Analysis Enhancement for Synchronous Program<br><i>Pascal Raymond</i> . . . . .   | 138 |
| Towards a Formal Software Design Methodology for Predictable Embedded Multi-processor Applications<br><i>Ingo Sander</i> . . . . .                       | 139 |
| AstraKahn: Coordination programming by extension and refinement of the Kahn model<br><i>Alex Shafarenko</i> . . . . .                                    | 139 |
| The Coroutine Model of Computation<br><i>Chris Shaver</i> . . . . .  | 140 |
| libDGALS: A Library-based Approach to Design Dynamic GALS Systems<br><i>Wei-Tsun Sun</i> . . . . .   | 141 |
| Compiling SCCharts (and other Sequentially Constructive Programs) to Hardware and Software<br><i>Reinhard von Hanxleden</i> . . . . .                    | 141 |
| <b>Conclusions</b> . . . . .   | 142 |
| <b>Participants</b> . . . . .  | 143 |

### 3 Overview of Talks

#### 3.1 Desynchronization of Synchronous Systems

*Yu Bai (TU Kaiserslautern, DE)*

**License** © Creative Commons BY 3.0 Unported license  
© Yu Bai

**Main reference** Y. Bai, K. Schneider, “Isochronous Networks by Construction Design,” to appear in Proc. of the Design, Automation and Test in Europe Conf. (DATE’14).

In this presentation, the main threads of methodologies in desynchronization of synchronous systems are discussed.

In the introduction, the three methods: latency-insensitive design, elastic circuits and desynchronization of synchronous programs are covered briefly, followed by their pros and cons. Finally a model-based approach is proposed in order to cover different design properties.

The second part of the talk introduced the simulation of synchronous elastic circuits in SystemC as an application of the proposed model-based approach, where synthesis of elastic modules and elastic channels are presented.

The last part discussed the endo / isochronous systems. Related concepts are compared with examples. Finally a general theorem of correct desynchronization is introduced: if the synchronous system  $P = P_1 \parallel \dots \parallel P_n$  (the synchronous composition of  $n$  processes) is constructive and clock-consistent, and each process  $P_i$  is patient, then the process  $P$  can be correctly desynchronized to a GALS system.

#### 3.2 Representing Spatially Moving Entities using Time-Variant Topologies

*Fernando Barros (University of Coimbra, PT)*

**License** © Creative Commons BY 3.0 Unported license  
© Fernando Barros

The representation of spatially moving systems is a complex task since communication is unstructured, making it difficult to assess what are the entities currently communicating. Given that interaction is mainly governed by the physical location of the entities, the communication pattern changes over time requiring a dynamic topology. To solve this problem we use the Heterogeneous Flow Systems Specification (HFSS), a modular modeling formalism designed to represent hybrid systems with time-variant topologies. We exploit the ability to represent dynamic topologies as an alternative to a representation using publish/subscribe (pub/sub) communication. Additionally, we show that HFSS dynamic topologies can support a large variety of representations taking advantage of the characteristics of the application domains, enabling more expressive and more efficient descriptions of moving entities.



### 3.3 From Quasi-Synchrony to LTTA

*Guillaume Baudart (ENS – Paris, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Guillaume Baudart

**Joint work of** Baudart, Guillaume; Bourke, Timothy; Pouzet, Marc

A Quasi-periodic System is one where every process  $P$  is periodic with a nominal period and a jitter. The time between two ticks may thus vary between 'small margins' during an execution:

Signal values are sent across a bus to one-place buffers at a receiver, whence they are sampled periodically.

In his 'cooking book', Paul Caspi showed how to build abstractions for implementing discrete systems on top of this architecture. In later work, with Albert Benveniste and others, he proposed communication protocols for preserving the discrete semantics of signal flows.

We present a brief survey of this work. In particular, we explain the simple relations between the periods and jitters of real-time tasks, and overwriting and oversamplings of values between writers and readers (it's all a matter of fence posts). We generalize (slightly) the idea of quasi-synchronous traces. We also clarify one of the communication protocols by modelling it in the hybrid synchronous language Zelus.

### 3.4 BPDF: A Statically Analyzable DataFlow Model with Integer and Boolean Parameters

*Vagelis Bebelis (INRIA Grenoble – Rhône-Alpes, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Vagelis Bebelis

**Joint work of** Bebelis, Vagelis; Fradet, Pascal; Girault, Alain; Lavigueur, Bruno

**Main reference** V. Bebelis, P. Fradet, A. Girault, B. Lavigueur, "BPDF: A statically analyzable dataflow model with integer and boolean parameters," in Proc. of the Int'l Conf. on Embedded Software (EMSOFT'13), pp. 1–10, IEEE, 2013.

**URL** <http://dx.doi.org/10.1109/EMSOFT.2013.6658581>

Dataflow programming models are well-suited to program many-core streaming applications. However, many streaming applications have a dynamic behavior. To capture this behavior, parametric dataflow models have been introduced over the years. Still, such models do not allow the topology of the dataflow graph to change at runtime, a feature that is also required to program modern streaming applications. To overcome these restrictions, we propose a new model of computation, the *Boolean Parametric Data Flow* (BPDF) model which combines integer parameters (to express dynamic rates) and boolean parameters (to express the activation and deactivation of communication channels). High dynamicity is provided by integer parameters which can change at each basic iteration and boolean parameters which can even change within the iteration.

The major challenge with such dynamic models is to guarantee liveness and boundedness. We present static analyses which ensure statically the liveness and the boundedness of BPDF graphs. We also introduce a scheduling methodology to implement our model on highly parallel platforms and demonstrate our approach using a video decoder case study.

### 3.5 Towards Discrete Controller Synthesis for the Reactive Adaptation of Autonomic Systems

*Nicolas Berthier (INRIA Rennes – Bretagne Atlantique, FR)*

License  Creative Commons BY 3.0 Unported license  
© Nicolas Berthier

About a decade ago, and due to the ever growing complexity of computer systems, a trend appeared putting forward the automation of the difficult tasks of software systems administration. The software assigned to this work is usually called an Autonomic Management System (AMS). It is composed of software components that evaluate the dynamics of the system under management through measurements (e.g., workload, memory usage), take decisions, and act upon it so that it stays in a set of acceptable states. Some components ensure performance and availability of the system, while others manage the redundancy of its hardware constituents to deal with errors. However, the actual design of such software leads to inconsistencies in the taken decisions, and coordination issues.

First, to tackle this problem, we take a global view and underscore the reactive nature of AMSs. This point of view allows us to suggest a new approach for the design of AMS software, based on synchronous programming and discrete controller synthesis techniques (DCS). They provide us with high-level languages for the specification of the system to manage, as well as means for statically dealing with inconsistencies and coordination issues. We illustrate our approach by applying our design to a realistic multi-tier application, and present an evaluation of its practicality by using a prototype implementation.

We also exploit the preceding modeling use case to identify the needs for extending DCS algorithms to handle quantitative properties. Over a second phase, we introduce ReaX, a new tool currently under development allowing the synthesis of controllers for logico-numerical reactive programs.

### 3.6 A slow afternoon chez PARKAS and a very fast fly (a fun talk)

*Timothy Bourke (ENS – Paris, FR)*

License  Creative Commons BY 3.0 Unported license  
© Timothy Bourke

Joint work of Bourke, Timothy; Pouzet, Marc

We briefly present a problem posed to use by Rafel Cases and Jordi Cortadella during a lunch organised by Gerard Berry. We propose solutions in the Simulink tool<sup>3</sup> and our language Zélus<sup>4</sup>.

Imagine two cars. One starts at Barcelona and travels at 50 km/hr toward Girona—a distance of 100 km. The other starts at Girona and travels at 50 km/hr toward Barcelona. Between the two is a fly travelling at 80 km/hr, initially from Barcelona toward Girona, and changing direction instantaneously whenever it meets either car. There are two questions.

1. How many zig-zags does the fly do during the two hours of travel?
2. Where will the fly be when the two cars reach their destinations?

<sup>3</sup> <http://www.mathworks.com/products/simulink/>

<sup>4</sup> <http://zelus.di.ens.fr>

We first modelled this problem in Simulink. The number of zig-zags, to our great surprise and pleasure, was **42!** [1] (Using R2012a with the ODE45 solver and a relative tolerance of  $1 \times 10^{-3}$ .)

We then modelled the problem in Zélus. This gave an answer of **48**. (Using the Sundials CVODE solver and a custom implementation of the Illinois algorithm.)

Obviously neither answer is correct since the system is not well defined at the instant the cars pass each other. The important questions are whether we should, or even can, statically detect and reject such cases? or stop with an error at runtime?

### References

- 1 D. Adams. *The Hitchhiker's Guide to the Galaxy*. Pan Books, 1979.

## 3.7 Modelyze: Embedding Equation-Based DSLs

*David Broman (University of California – Berkeley, US)*

**License** © Creative Commons BY 3.0 Unported license  
© David Broman

**Joint work of** Broman, David; Siek, Jeremy

**Main reference** D. Broman, J. G. Siek, “Modelyze: a Gradually Typed Host Language for Embedding Equation-Based Modeling Languages,” Technical report, EECS Department, University of California, Berkeley, UCB/EECS-2012-173, June, 2012.

**URL** <http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-173.html>

Cyber-physical systems combine computations, networks, and physical processes. Modeling and analysis of such systems are vital engineering techniques to manage complexity and enable rapid prototyping. In particular, complex cyber-physical systems are heterogeneous, requiring various model of computations. A key challenge is to provide both expressive modeling capabilities and mechanisms for analyzing these heterogeneous systems. This talk explores a solution to this challenge based on domain-specific embedded languages. We introduce a host language, named Modelyze, in which various domain-specific modeling languages may be embedded. The key features of Modelyze are first-class functions, which provide a mechanism to abstract components of a model, and typed symbolic expressions, to represent and manipulate equations and expressions. The type system for symbolic expressions supports model-level static error checking and provides an automatic lifting translation to provide seamless integration between the host language and the equations represented by symbolic expressions. The type system is based on gradual typing, enabling early static checking for model engineers while providing expressiveness for domain experts.

## 3.8 Index theory for Hybrid DAE Systems

*Benoit Caillaud (INRIA Rennes – Bretagne Atlantique, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Benoit Caillaud

Hybrid systems modelers exhibit a number of difficulties related to the mix of continuous and discrete dynamics and sensitivity to the discretization scheme. Modular modeling, where subsystems models can be simply assembled with no rework, calls for using Differential Algebraic Equations (DAE). In turn, DAE are strictly more difficult than ODE. They require sophisticated pre-processing using various notions of index before they can be submitted

to a solver. In this talk we discussed some fundamental issues raised by the modeling and simulation of hybrid systems involving DAEs. We focused on the following questions:

- What is the proper notion of index for a hybrid DAE system?
- What are the primitive statements needed for a DAE hybrid systems modeler?

The differentiation index for DAE explicitly relies on everything being differentiable. Therefore, generalizations to hybrid systems must be done with caution. We proposed relying on non-standard analysis for this. Non-standard analysis formalizes differential equations as discrete step transition systems with infinitesimal time basis. We could thus bring hybrid DAE systems to their non-standard form, where the notion of difference index can be firmly used.

### 3.9 Functioning Hardware from Functional Specifications

*Stephen A. Edwards (Columbia University – New York, US)*

**License** © Creative Commons BY 3.0 Unported license  
 © Stephen A. Edwards  
**URL** <http://www.cs.columbia.edu/~sedwards>

For performance at low power, tomorrow’s chips will be mostly application-specific logic only powered when needed. I propose synthesizing it from the functional language Haskell. My approach – rewriting to a simple dialect that enables a syntax-directed translation – enables parallelization and distributed memory systems. Transformations include scheduling arithmetic operations, replacing recursion with iteration, and improving data locality by inlining recursive types. I am developing a compiler based on these principles.

### 3.10 Debugging and Compiler Bootstrapping with an Equation-Based Language Compiler

*Peter A. Fritzson (Linköping University, SE)*

**License** © Creative Commons BY 3.0 Unported license  
 © Peter A. Fritzson  
**Joint work of** Fritzson, Peter A.; Pop, Adrian; Sjoelund, Martin; Asghar, Adeel; Casella, Francesco  
**Main reference** A. Pop, M. Sjölund, A. Asghar, P. Fritzson, F. Casella, “Static and Dynamic Debugging of Modelica Models,” in Proc. of the 9th Int’l Modelica Conference (Modelica’12), pp. 443–454, Linköping Electronic Conference Proceedings, Linköping University Electronic Press, 2012.  
**URL** <http://dx.doi.org/10.3384/ecp12076443>

The high abstraction level of equation-based object-oriented languages (EOL) such as Modelica has the drawback that programming and modeling errors are often hard to find. In this paper we present static and dynamic debugging methods for Modelica models and a debugger prototype that addresses several of those problems. The goal is an integrated debugging framework that combines classical debugging techniques with special techniques for equation-based languages partly based on graph visualization and interaction. The static transformational debugging functionality addresses the problem that model compilers are optimized so heavily that it is hard to tell the origin of an equation during runtime. This work proposes and implements a prototype of a method that is efficient with less than one percent overhead, yet manages to keep track of all the transformations/operations that the compiler performs on the model. Modelica models often contain functions and

algorithm sections with algorithmic code. The fraction of algorithmic code is increasing since Modelica, in addition to equation-based modeling, is also used for embedded system control code as well as symbolic model transformations in applications using the MetaModelica language extension. Our earlier work in debuggers for the algorithmic subset of Modelica caused instrumentation-based techniques which are portable but turned out to have too much overhead for large applications. The new debugger is the first Modelica debugger that can operate without run-time information from instrumented code. Instead it communicates with a low-level C-language symbolic debugger to directly extract information from a running executable, set and remove break-points, etc. This is made possible by the new bootstrapped OpenModelica compiler which keeps track of a detailed mapping from the high level Modelica code down to the generated C code compiled to machine code. The debugger is operational, supports both standard Modelica data structures and tree/list data structures, and operates efficiently on large applications such as the OpenModelica compiler with more than 100 000 lines of code. Moreover, an integrated debugging approach is proposed that combines static and dynamic debugging. To our knowledge, this is the first Modelica debugger that supports transformational debugging and algorithmic code debugging. This presentation also reports on the first bootstrapping (i.e., a compiler can compile itself) of a full-scale EOO (Equation-based Object-Oriented) modeling language such as Modelica. The Modelica language has been modeled/implemented in the OpenModelica compiler (OMC) using an extended version of Modelica called MetaModelica. OMC models the MetaModelica language and is now compiling itself with good performance. Benefits include a more extensible maintainable compiler, also making it easier to add functionality such as the above mentioned debugging support.

## References

- 1 Martin Sjölund and Peter Fritzson. Debugging Symbolic Transformations in Equation Systems. In Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, (EOOLT'2011), Zürich, Switzerland, Sept 5, 2011. Published by Linköping University Electronic Press, [http://www.ep.liu.se/ecp\\_home/index.en.aspx?issue=056](http://www.ep.liu.se/ecp_home/index.en.aspx?issue=056), Sept 2011.
- 2 Adrian Pop, Martin Sjölund, Adeel Asghar, Peter Fritzson, Francesco Casella. Static and Dynamic Debugging of Modelica Models. In Proceedings of the 9th International Modelica Conference (Modelica'2012), Munich, Germany, Sept. 3–5, 2012.
- 3 Martin Sjölund, Peter Fritzson, and Adrian Pop. Bootstrapping a Modelica Compiler aiming at Modelica 4. In Proceedings of the 8th International Modelica Conference (Modelica'2011), Dresden, Germany, March. 20–22, 2011.

## 3.11 Interactive Verification of Cyber-physical Systems

*Manuel Gesell (TU Kaiserslautern, DE)*

**License** © Creative Commons BY 3.0 Unported license  
© Manuel Gesell

**Joint work of** Gesell, Manuel; Li, Xian; Schneider, Klaus;


Cyber-physical systems (CPS) are widely used in safety-critical applications, there is a crucial need for modeling, simulation, and verification. Numerous approaches and tools for CPS verification have already been proposed in the past. Most of them concentrate on model-checking of finite abstractions of restricted classes of CPS. Interactive verification is

an alternative approach that does not suffer from the high complexity of decision procedures, which is well-suited for CPS verification.

Recently, the synchronous Quartz language has been extended for modeling cyber-physical systems, and a corresponding interactive theorem prover AIFProver is currently in development. It combines both model checking and theorem proving ideas, and supports compositional verification. The prototypical version has already been proved to be applicable to large discrete systems and a well-known benchmark of cyber-physical systems. Here, we will demonstrate the capability of the interactive verification approach and tool worked out so far, together with the key techniques remain to be solved in the near future.

### 3.12 Behavioral Equivalence of Transducers under a Fixed Protocol

*Dan R. Ghica (University of Birmingham, GB)*

License  Creative Commons BY 3.0 Unported license  
© Dan R. Ghica

Joint work of Ghica, Dan R.; Fredriksson, Olle; Al-Zobaidi, Zaid

This talk gives an overview of the “Geometry of Synthesis” programme of research, concerning the synthesis of hardware descriptions from specifications written in higher-order, imperative, recursive, concurrent programming languages. In the context of hardware synthesis we present a new<sup>5</sup> technique for aggressive minimisation of state machines taking into account constrained environments, which we call “coherent optimisation”. The main properties of the technique (soundness and compositionality) are proved formally using the proof assistant Agda.

### 3.13 Timing Through Types

*Dan R. Ghica (University of Birmingham, GB)*

License  Creative Commons BY 3.0 Unported license  
© Dan R. Ghica

Joint work of Ghica, Dan R.; Smith, Alex

We introduce a new general notion of resource based on Bounded Linear Logic (BLL) which has the algebraic structure of a semiring. For timing we use a semiring of schedules which are multisets of contractive linear affine transformation. In order to prove the coherence of the type system we describe a categorical model framework. We present, in the concrete case of timing, a simple type inference algorithm based on generating systems of constraints solvable by an SMT such as Zr.

---

<sup>5</sup> As far as we know.

### 3.14 Precise Timing Analysis for Direct-Mapped Caches

*Alain Girault (INRIA Grenoble – Rhône-Alpes, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Alain Girault

**Joint work of** Andalam, Sidharta; Sinha, Roopak; Roop, Partha; Girault, Alain; Reineke, Jan

**Main reference** S. Andalam, R. Sinha, P. S. Roop, A. Girault, J. Reineke, “Precise timing analysis for direct-mapped caches,” in Proc. of the 50th Annual Design Automation Conf. (DAC’13), pp. 148:1–148:10, ACM, 2013; available as pre-print at HAL.

**URL** <http://dx.doi.org/10.1145/2463209.2488917>

**URL** <http://hal.archives-ouvertes.fr/hal-00842368>

Safety-critical systems require guarantees on their worst-case execution times. This requires modelling of speculative hardware features such as caches that are tailored to improve the average-case performance, while ignoring the worst case, which complicates the Worst Case Execution Time (WCET) analysis problem. Existing approaches that precisely compute WCET suffer from state-space explosion. In this paper, we present a novel cache analysis technique for direct-mapped instruction caches with the same precision as the most precise techniques, while improving analysis time by up to 240 times. This improvement is achieved by analysing individual control points separately, and carrying out optimisations that are not possible with existing techniques.

### 3.15 When the decreasing sequence fails... Improving fixpoint approximation in program analysis

*Nicolas Halbwachs (VERIMAG – Grenoble, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Nicolas Halbwachs

**Joint work of** Halbwachs, Nicolas; Henry, Julien

**Main reference** N. Halbwachs, J. Henry, “When the decreasing sequence fails,” in Proc. of the 19th Int’l Symp. on Static Analysis (SAS’12), LNCS, Vol. 7460, pp. 198–213, Springer, 2012; available as pre-print at HAL.

**URL** [http://dx.doi.org/10.1007/978-3-642-33125-1\\_15](http://dx.doi.org/10.1007/978-3-642-33125-1_15)

**URL** <http://hal.archives-ouvertes.fr/hal-00734340>

The classical method for program analysis by abstract interpretation consists in computing a increasing sequence with widening, which converges towards a correct solution, then computing a decreasing sequence of correct solutions without widening. It is generally admitted that, when the decreasing sequence reaches a fixpoint, it cannot be improved further. As a consequence, all efforts for improving the precision of an analysis have been devoted to improving the limit of the increasing sequence. In this paper, we propose a method to improve a fixpoint after its computation. The method consists in projecting the solution onto well-chosen components and to start again increasing and decreasing sequences from the result of the projection.

### 3.16 Modal Interface Automata

*Gerald Luettgen (Universität Bamberg, DE)*

**License**  Creative Commons BY 3.0 Unported license  
© Gerald Luettgen

**Joint work of** Luettgen, Gerald; Vogler, Walter

Several modern interface theories for formally modelling and reasoning about component-based, concurrent systems have been built at the crossroads of de Alfaro and Henzinger’s Interface Automata (IA) and Larsen’s Modal Transition Systems (MTS). Two established examples are Nyman et al.s IOMTS and Bauer et al.s MIO, which differ in their view of component compatibility: IOMTS adopts an optimistic view leading to a more permissive parallel composition operator than MIO’s, but has technical shortcomings regarding (non-)monotonicity of refinement and the treatment of internal computation. In addition, both approaches neither consider conjunction on interfaces nor do they allow extending alphabets when refining system components, which are practically desired properties that enable one to specify and design systems incrementally.

This talk presents the novel interface theory Modal Interface Automata (MIA), which addresses the above shortcomings, and discusses MIA’s design decisions, trade-offs and limitations. The reported research is joint work with Walter Vogler of the University of Augsburg, Germany.

### 3.17 Safety Issues in MARTE/CCSL Specifications

*Frederic Mallet (INRIA Sophia Antipolis – Méditerranée, FR)*

**License**  Creative Commons BY 3.0 Unported license  
© Frederic Mallet

**Joint work of** Mallet, Frederic; Millo, Jean-Vivien

The Clock Constraint Specification Language (CCSL) proposes a rich polychronous time model dedicated to the specification of constraints on logical clocks: i.e., sequences of event occurrences. A priori independent clocks are progressively constrained through a set of clock operators that define when an event may occur or not. These operators can be described as labeled transition systems that can potentially have an infinite number of states. A CCSL specification can be scheduled by performing the synchronized product of the transition systems for each operator. Even when some of the composed transition systems are infinite, the number of reachable states in the product may still be finite: the specification is safe. The purpose of this paper is to propose a sufficient condition to detect that the product is actually safe. This is done by abstracting each CCSL constraint (relation and expression) as a marked graph. Detecting that some specific places, called counters, in the resulting marked graph are safe is sufficient to guarantee that the composition is safe.



### 3.18 In Search of a Physical Semantics of Boussinot's Reactive Model

*Louis Mandel (College de France, Paris, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Louis Mandel

**Joint work of** Aguado, Joaquin; Barros, Fernando; Baudart, Guillaume; Berry, Gérard; Bourke, Timothy; Boussinot, Frédéric; Caillaud, Benoit; de Simone, Robert; Delaval, Gwenaël; Edwards, Stephen; Ghica, Dan; Girault, Alain; Guatto, Adrien; Caillaud, Benoit; Lüttgen, Gerald; Maraninchi, Florence; Mendler, Michael; Pasteur, Cédric; Potop-Butucaru, Dumitru; Pouzet, Marc; Schneider, Klaus; Vuillemin, Jean

Gerard Berry showed earlier in the morning that the semantics of Esterel is given by electricity in circuits. This talk describes the search for a physical semantics of Boussinot's Reactive Model.

The first part of the talk presents the reactive model through the implementation in ReactiveML of the artwork “Carres Noir et Blanc” of Roger Vilder<sup>6</sup>.

The second part presents the five points of the reactive model which guarantee that all programs are causal by construction:

1. add a delay to the reaction to absence,
2. no strong abort,
3. handler of a weak preemption is executed with a delay,
4. add a delay to read the value of a signal,
5. always favour absence of signals.

The last point shows that the reactive model is not a subset of Esterel. For example the following program is not causal in Esterel but is correct in the reactive model: signal *s* in present *s* then emit *s* else ()

Therefore, electrical circuits do not provide a physical implementation of the reactive model. The assumption presented in this talk is that the reactive model can be implemented with circuits running with water instead of electricity.

Finally, a simulator of this kind of circuit implemented in ReactiveML is presented.

### 3.19 From Synchronous to Timed Programming

*Eleftherios Matsikoudis (University of California – Berkeley, US)*

**License** © Creative Commons BY 3.0 Unported license  
© Eleftherios Matsikoudis

High-level programming languages have allowed the programmer to ignore the specifics of the underlying execution platform, and focus just on the logic of the intended computational process, effectively decoupling programs from systems. Programs have become models of the systems that execute them. And conditioned on the absence of faults, any two systems executing the same program will have the same behaviour. This is true for sequential programs, and to some extent, for concurrent programs as well. But what about real-time programs?

In a real-time program, the programmer will typically specify the intended timing properties by direct access to the hardware, or use of available drivers specific to the targeted execution platform. The program becomes part of the system, and different programs

<sup>6</sup> [http://www.rogervilder.com/projets/carre\\_16.html](http://www.rogervilder.com/projets/carre_16.html)

```

actor Watchdog {
  interface {
    initialize : input channel (& channel unit)
    reset      : output channel unit
  }

  clear : & channel unit

  thread() {
    wait initialize? in [time , ...)
      clear = initialize

    wait *clear? in (time, time + 1.0)
      skip
    else
      reset = {}
  }
}

```

■ **Figure 1** The definition of a trivial watchdog actor in `act`.

are required to specify the same behaviour on different execution platforms. Real-time programming is still today low-level programming.

Our goal is a high-level programming language for timed systems. We use the term “timed” quite liberally to refer to any system that will determinately order its events relative to some physical or logical clock. We are interested in timed systems that are determinate and causal (see [2], [3]).

We present the basic features of a programming language that we call `act`. `act` is an actor-oriented timed programming language. An `act` program starts with the execution of the actor `main`. `main` can *create* other actors to form a dynamically evolving network of conceptually concurrent, memory isolated components that communicate solely through message passing. All actors in a program share a global notion of *logical* time, directly accessible in a program via the keyword `time`. The language allows for polymorphism in the type of `time`. Logical time advances through the use of *temporal* statements, such as `wait`. Non-temporal statements execute in *zero* logical time.

Figure 1 shows the definition of an actor that implements the functionality of a rather trivial watchdog in `act`. The actor consists of

1. a block of channel definitions, making up the interface of the actor,
2. an uninitialized variable definition local to the actor, representing the state of the actor, and
3. the actor’s thread of control, specifying the behaviour of the actor.

Once created, the Watchdog actor will wait until there is an event at the “initialize” channel, including the time instance at which the actor was created. It will then wait until there is an event at the channel whose address the actor was initialized with, and send a reset signal if there is no such event within 1.0 units of time from the time of initialization.

The watchdog example is interesting because it represents a determinate, causal component that does not preserve the prefix relation on discrete-event signals, and thus, cannot be

implemented as a data flow actor.

The theoretical basis for the design of `act`, and specifically, the choice of the temporal statement `wait`, is its completeness over all *synchronous* causal functions on discrete-event signals.

`act` adopts the zero-execution-time hypothesis common to all synchronous programming languages, and many of its constructs are inspired by Esterel. But it allows for time to be dense, and unlike Esterel, treats conditionals as sequential statements in the resolution of causal loops.

By relating logical to physical time, according to the PTIDES paradigm (see [5], [1]), `act` can be used to program real-time systems. The algorithmic approach presented in [4] can be extended to suitably chosen fragments of the language to perform the schedulability analysis necessary for hard real-time applications.

### References

- 1 John C. Eidson, Edward A. Lee, Slobodan Matic, Sanjit A. Seshia, and Jia Zou. Distributed real-time software for cyber-physical systems. *Proceedings of the IEEE*, 100(1):45–59, January 2012.
- 2 Eleftherios Matsikoudis and Edward A. Lee. The fixed-point theory of strictly causal functions. Technical Report UCB/EECS-2013-122, EECS Department, University of California, Berkeley, Jun 2013.
- 3 Eleftherios Matsikoudis and Edward A. Lee. On fixed points of strictly causal functions. Víctor Braberman and Laurent Fribourg, editors, *Formal Modeling and Analysis of Timed Systems*, volume 8053 of *Lecture Notes in Computer Science*, pages 183–197. Springer Berlin Heidelberg, 2013.
- 4 Eleftherios Matsikoudis, Christos Stergiou, and Edward A. Lee. On the schedulability of real-time discrete-event systems. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–15, 2013.
- 5 Yang Zhao, Jie Liu, and Edward A. Lee. A programming model for time-synchronized distributed real-time systems. In *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS '07. 13th IEEE*, pages 259–268, April 2007.

## 3.20 Berry-Constructive Programs are Sequentially Constructive, or: Synchronous Programming from a Scheduling Perspective

Michael Mendler (Universität Bamberg, DE)

License © Creative Commons BY 3.0 Unported license  
© Michael Mendler

Joint work of Aguado, Joaquín; von Hanxleden, Reinhard; Fuhrmann, Insa

Main reference J. Aguado, M. Mendler, R. von Hanxleden, I. Fuhrmann, “Grounding Synchronous Deterministic Concurrency in Sequential Programming,” to appear in Proc. of European Symposium on Programming (ESOP’14), Grenoble, April 2014.

We introduce an abstract value domain  $I(D)$  and associated fixed point semantics for reasoning about concurrent and sequential variable valuations within a synchronous cycle-based model of computation. We use this domain for a new behavioural definition of Berry’s causality analysis for Esterel in terms of approximation intervals. This gives a compact and more uniform understanding of causality and generalises to other data-types. We also prove that Esterel’s ternary domain and its semantics is conservatively extended by the recently proposed sequentially constructive (SC) model of computation. This opens the door to a direct mapping of Esterel’s signal mechanism into boolean variables that can be set and reset

arbitrarily within a tick. We illustrate the practical usefulness of this mapping by discussing how signal reincarnation is handled efficiently by this transformation, which is of complexity that is linear in program size, in contrast to earlier techniques that had, at best, potentially quadratic overhead.

### 3.21 SCCharts – Sequentially Constructive Charts

*Christian Motika (Universität Kiel, DE)*

**License** © Creative Commons BY 3.0 Unported license  
© Christian Motika

**Joint work of** von Hanxleden, Reinhard; Duderstadt, Bjoern; Motika, Christian; Smyth, Steven; Mendler, Michael; Aguado, Joaquin; Mercer, Stephen; O'Brien, Owen

**Main reference** R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, O. O'Brien, "SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications," Technical Report 1311, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Dec. 2013.

**URL** [http://www.informatik.uni-kiel.de/uploads/tx\\_publication/tr-1311-bericht.pdf](http://www.informatik.uni-kiel.de/uploads/tx_publication/tr-1311-bericht.pdf)

We present a new visual language, SCCharts, designed for specifying safety-critical reactive systems. SCCharts uses a new statechart notation similar to Harel Statecharts [3] and provides deterministic concurrency based on a synchronous model of computation (MoC), without restrictions common to previous synchronous MoCs like the Esterel constructive semantics [2]. Specifically, we lift earlier limitations on sequential accesses to shared variables, by leveraging the sequentially constructive MoC [4]. Thus SCCharts in short are SyncCharts [1] syntax plus Sequentially Constructive semantics.

The key features of SCCharts are defined by a very small set of elements, the Core SCCharts, consisting of state machines plus fork/join concurrency.

Conversely, Extended SCCharts contain a rich set of advanced features, such as different abort types, signals, history transitions, etc., all of which can be reduced via semantics preserving model-to-model (M2M) transformations into Core SCCharts. Extended SCCharts features are syntactic sugar because they can be expressed by a combination of Core SCCharts features.

On the one hand this eases the compilation and makes it more robust because it reduces its complexity. On the other hand, using Extended SCCharts features, a modeler is able to abstract away complexity of his or her SCCharts model which increases robustness and readability of a model. This approach enables a simple yet efficient compilation strategy and aids verification and certification.

#### References

- 1 C. Andre. *Semantics of SyncCharts*. Technical Report ISRN I3S/RR-2003-24-FR, I3S Laboratory, Sophia-Antipolis, France, April 2003.
- 2 G. Berry. *The foundations of Esterel*. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, pages 425-454, Cambridge, MA, USA, 2000.
- 3 D. Harel. *Statecharts: A visual formalism for complex systems*. *Science of Computer Programming*, 8(3):231-274, June 1987.
- 4 R. von Hanxleden, M. Mendler, J. Aguado, B. Duderstadt, I. Fuhrmann, C. Motika, S. Mercer, and O. O'Brien. *Sequentially Constructive Concurrency – A conservative extension of the synchronous model of computation*. In *Proc. Design, Automation and Test in Europe Conference (DATE'13)*, Grenoble, France, March 2013.

### 3.22 A Tale of Two Semantics: Clocked Dimensions in a Multidimensional Language

*John Plaice (The University of New South Wales, AU)*

**License** © Creative Commons BY 3.0 Unported license  
© John Plaice

**Joint work of** Plaice, John; Beck, Jarryd; Mancilla, Blanca; Wadge, William

In 1975, William W. Wadge and Edward A. Ashcroft introduced the language Lucid, in which the value of a variable was a stream. The successors to Lucid took two paths.

The first path, taken by Lustre, was to restrict the language so that a stream could be provided with a timed semantics, where the  $i$ -th element of a stream appeared with the  $i$ -th tick of the stream's clock, itself a Boolean stream. Today, Lustre is at the core of the Scade software suite, the reference tool for avionics worldwide.

The second path was to generalize the language to include multidimensional streams and higher-order functions. The latest language along this path is TransLucid, a higher-order functional language in which variables define arbitrary-dimensional arrays, where any atomic value may be used as a dimension, and a multidimensional runtime context is used to index the variables.

This talk will show how the two paths are being brought back together, with the introduction of clocked dimensions to TransLucid, thereby allowing for synchronous, reactive programming to take place within the context of a full-fledged higher-order declarative language.

### 3.23 Integrated WCET estimation of multicore applications

*Dumitru Potop-Butucaru (INRIA – Siège, FR)*

**License** © Creative Commons BY 3.0 Unported license  
© Dumitru Potop-Butucaru

**Joint work of** Potop-Butucaru, Dumitru; Puaut, Isabelle

**Main reference** D. Potop-Butucaru, I. Puaut, "Integrated Worst-Case Execution Time Estimation of Multicore Applications," in Proc. of the 13th Int'l Workshop on Worst-Case Execution Time Analysis, OASlcs, Vol. 30, pp. 21–31, Schloss Dagstuhl/Dagstuhl Publishing, 2013.

**URL** <http://dx.doi.org/10.4230/OASlcs.WCET.2013.21>

Worst-case execution time (WCET) analysis has reached a high level of precision in the analysis of sequential programs executing on single-cores. In this paper we extend a state-of-the-art WCET analysis technique to compute *tight* WCETs estimates of parallel applications running on multi-cores. The proposed technique is termed *integrated* because it considers jointly the sequential code regions running on the cores and the communications between them. This allows to capture the hardware effects across code regions assigned to the same core, which significantly improves analysis precision. We demonstrate that our analysis produces tighter execution time bounds than classical techniques which first determine the WCET of sequential code regions and then compute the global response time by integrating communication costs. Comparison is done on two embedded control applications, where the gain is of 21% on average.

### 3.24 A Causality Analysis for Hybrid Modelers

*Marc Pouzet (ENS – Paris, FR)*

License © Creative Commons BY 3.0 Unported license  
© Marc Pouzet

Joint work of Pouzet, Marc; Benveniste, Albert; Bourke, Timothy; Caillaud, Benoit; Pagano, Bruno

Explicit hybrid systems modelers like Simulink/Stateflow allow for programming both discrete- and continuous-time behaviors with complex interactions between them. A key issue in their compilation is the static detection of algebraic or causality loops. Such loops can cause simulations to deadlock, are a source of compilation bugs and prevent the generation of statically scheduled code.

This paper addresses this issue for a hybrid modeling language that combines synchronous Lustre-like data-flow equations with Ordinary Differential Equations (ODEs). We introduce the operator  $\text{last}(x)$  for the left-limit of a signal  $x$ . This operator is used to break causality loops and permits a uniform treatment of discrete and continuous state variables. The semantics relies on non-standard analysis, defining an execution as a sequence of infinitesimally small steps. A signal is deemed causally correct when it can be computed sequentially and only progresses by infinitesimal steps outside of discrete events. The causality analysis takes the form of a simple type system. In well-typed programs, signals are proved continuous during integration and can be translated into sequential code for integration with off-the-shelf ODE solvers.

The effectiveness of this system is illustrated with several examples written in Zelus, a Lustre-like synchronous language extended with hierarchical automata and ODEs.

### 3.25 Timing Analysis Enhancement for Synchronous Program

*Pascal Raymond (VERIMAG – Grenoble, FR)*


License © Creative Commons BY 3.0 Unported license  
© Pascal Raymond

Joint work of Raymond, Pascal; Maiza, Claire

In real-time systems, an upper-bound on the execution time is mandatory to guarantee all timing constraints: a bound on the Worst-Case Execution Time (WCET). High-level synchronous approaches are usually used to design hard real-time systems and specifically critical ones. Timing analysis used for WCET estimates are based on the executable binary program. Thus, a large part of semantic information, known at the design level, is lost due to the compilation scheme (typically organized in two stages, from high-level model to C, and then binary code). In this paper, we aim at improving the estimated WCET by taking benefit from high-level information. We integrate an existing verification tool to check the feasibility of the worst-case path. Based on a realistic example, we show that there is a large possible improvement for a reasonable analysis time overhead.

### 3.26 Towards a Formal Software Design Methodology for Predictable Embedded Multiprocessor Applications

*Ingo Sander (KTH Royal Institute of Technology, SE)*


License  Creative Commons BY 3.0 Unported license  
© Ingo Sander

The presentation addresses the increasing complexity of software design for multiprocessor embedded systems by proposing a design methodology that combines a formal foundation based on the theory of models of computation (MoCs) and the industrial system design language SystemC. In particular a software synthesis flow is presented that starts with an executable system model and yields an implementation on a multiprocessor system-on-chip.

The ForSyDe (Formal System Design) methodology provides the designer with SystemC modeling libraries that lead to executable system models from which abstract analyzable models can be extracted. Using these abstract models, the design space exploration, mapping and synthesis process can make use of the rich set of existing MoC theory by for instance incorporating scheduling and buffer optimization techniques to yield an efficient implementation on a multiprocessor system-on-chip. The presentation will also discuss to what extent performance guarantees can be given provided a predictable architecture is used as target architecture.

### 3.27 AstraKahn: Coordination programming by extension and refinement of the Kahn model

*Alex Shafarenko (University of Hertfordshire, GB)*

License  Creative Commons BY 3.0 Unported license  
© Alex Shafarenko

This talk introduces some concepts of the coordination language AstraKahn which can be used for programming synchronous and asynchronous systems within the same framework. The talk dwells primarily on the bottom layer of the AstraKhan stack, which is called the Topology and Progress Layer. The concept of pressure-based progress control is explained and the coordination of pressure via state-machine based synchronisers is discussed. This is work in progress. The current definition is available in the form of Arxiv preprint [1].

#### References

- 1 Alex Shafarenko *AstraKahn: A Coordination Language for Streaming Networks*. arXiv.org arXiv:1306.6029v3 [cs.PL], 2013

### 3.28 The Coroutine Model of Computation

*Chris Shaver (University of California – Berkeley, US)*

**License** © Creative Commons BY 3.0 Unported license  
© Chris Shaver

**Joint work of** Shaver, Chris; Lee, Edward A.

**Main reference** C. Shaver, E. A. Lee. “The coroutine model of computation,” in R. B. France et al. (eds.) “Model Driven Engineering Languages and Systems”, LNCS, Vol. 7590, pp. 319–334, Springer, 2012.

**URL** [http://dx.doi.org/10.1007/978-3-642-33666-9\\_21](http://dx.doi.org/10.1007/978-3-642-33666-9_21)

The Coroutine Model of Computation, defined by Shaver and Lee [5], is a formalism that generalizes other control-oriented models such as state machines, modal models, and imperative programs into a denotational language. Specifically, this denotational language is expressed in terms of the Modular Actor Interfaces of Tripakis et al. [6]. The semantics of this model defines a general interface for Continuation Actors, Actors that in addition to the usual inputs, outputs, and state have control-oriented features: control entry points, control exit locations, and the ability to suspend, terminate, or resume in the context of their containing model. These Continuation Actors can be assembled into a transition system, forming a Coroutine Model.

As opposed to conventional formalisms for state machines, the decision whether or how to transition control, typically codified in a transition guard language, is instead formally part of the interface of each individual Continuation Actor, called its ‘enter’ function. This idea is derived from Andre’s semantics for SyncCharts[1, 2], where he makes a similar association of control decisions with Reactive Cells. Consequently, a simple denotational semantics is given for a Coroutine Model that traverses a sequence of Coroutine Actors, firing each to produce outputs, and deciding how to proceed, whether to suspend, or whether to terminate after executing each Coroutine Actor by using the ‘enter’ function in its interface.

The semantics of the Coroutine Model are additionally extended to accommodate a non-strict form of operation. Given partial information about inputs, in the form of a domain representation, the domain of the power set with inverse inclusion is used to represent partial information about the finite set of possible control decisions for each Continuation Actor. With this definition of partial control decisions at each Continuation Actor, non-strict operation can be defined on the level of the whole Coroutine Model. A simple denotational definition is given for this behavior, and it is proven under this definition that if each contained Continuation Actor defines its outputs and control decisions as monotonic functions of its inputs (in the domain-theoretic sense), the semantics of the whole model will define its outputs and its ultimate control decision as monotonic functions of its inputs.

This monotonicity property is important in the context of hierarchical and heterogeneous models as a form of compositionality. In particular, a collection of such monotonic Coroutine Models can be put together in a mutual constructive fixed-point computation over their connected inputs and outputs, such as that of the Synchronous Reactive model defined by Edwards [3]. This property allows the Coroutine Model of computation to give the semantic quotient of control-oriented synchronous languages over fixed-point computations, providing a theoretical framework for expressing models such as SyncCharts [1] as hierarchical compositions of Coroutine Models and Synchronous Reactive Models, as is done with the KlePto translation of Motika [4].

#### References

- 1 C. Andre. *SyncCharts: A visual representation of reactive behaviors*. Rapport de recherche tr95-52, Universite de Nice-Sophia Antipolis, France, 1995.



- 2 C. Andre. *Computing synccharts reactions*. Electronic Notes in Theoretical Computer Science, 88(33):3–19, 2004.
- 3 S.A. Edwards. *The Specification and Execution of Heterogeneous Synchronous Reactive Systems*. Technical report, University of California Berkeley, 1997.
- 4 C. Motika. *Semantics and Execution of Domain Specific Models—KlePto and an Execution Framework*. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, December 2009.
- 5 C. Shaver, and E. A. Lee. *The coroutine model of computation*. Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, 2012. 319-334.
- 6 S. Tripakis, C. Stergiou, C. Shaver, and E.A. Lee. *A modular formal semantics for Ptolemy*. Mathematical Structures in Computer Science, 23, pp 834-881. doi:10.1017/S0960129512000278.

### 3.29 libDGALS: A Library-based Approach to Design Dynamic GALS Systems

*Wei-Tsun Sun (INRIA Grenoble – Rhône-Alpes, FR)*

License © Creative Commons BY 3.0 Unported license  
© Wei-Tsun Sun

We tackle the problem of designing and programming dynamic and reactive systems with four objectives: being based on a formal model of computation, using different types of concurrency, being efficient, and tolerating failures. The challenge lies in the fact that good formal models with very high level of abstraction generally result in non-efficient implementations. We propose a C based library approach following the formal Dynamic Globally Asynchronous Locally Synchronous (DGALS) model of computation. We show how a DGALS system can be dynamically constructed from concurrent behaviors on distributed platforms thanks to the DGALS paradigm. Finally, our experimental results clearly indicate the large execution time and memory footprint gains compared to the current state of the art approaches.

### 3.30 Compiling SCCharts (and other Sequentially Constructive Programs) to Hardware and Software

*Reinhard von Hanxleden (Universität Kiel, DE)*

License © Creative Commons BY 3.0 Unported license  
© Reinhard von Hanxleden

SCCharts [3] extend SyncCharts [1] with sequential constructiveness (SC) [2] and other features. We developed a compilation chain that first, in a *high-level compilation* phase, performs a sequence of model-to-model transformations at the SCCharts-level [3] such that they can be mapped directly to SC Graphs (SCGs). Then two alternative *low-level compilation* approaches allow mapping to hardware and software; the circuit approach generates a netlist, the priority approach simulates concurrency with interleaved threads.

### References

- 1 C. Andre. *Semantics of SyncCharts*. Technical Report ISRN I3S/RR-2003-24-FR, I3S Laboratory, Sophia-Antipolis, France, April 2003.
- 2 R. von Hanxleden, M. Mendler, J. Aguado, B. Duderstadt, I. Fuhrmann, C. Motika, S. Mercer, and O. O'Brien. *Sequentially Constructive Concurrency – A conservative extension of the synchronous model of computation*. In Proc. Design, Automation and Test in Europe Conference (DATE'13), Grenoble, France, March 2013.
- 3 R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. O'Brien. *SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications*. Technical Report 1311, Christian-Albrechts-Universitaet zu Kiel, Department of Computer Science, Dec 2013.

## 4 Conclusions

The seminar had many high-quality presentations and even more important, many fruitful discussions afterwards until the late evening hours. During the seminar, several research groups discussed their work, and it is not surprising that, as in previous years, some common research has been initiated. For example, Reinhard von Hanxleden and Michael Mendler are currently working on a shared DFG project where they explore new forms of causality that is called sequential causality, which is closer to the traditional sequential programming languages. As another example, Klaus Schneider, Jean-Pierre Talpin and Sandeep Shukla have published several papers about the combination of polychronous and synchronous languages including aspects like clock consistency and causality which cross fertilized both areas. Many other projects benefited from contributions of experts in the area, and therefore we are sure that the Synchronous Programming workshop will celebrate also other anniversaries in the future.

## Participants

- Joaquin Aguado  
Universität Bamberg, DE
- Mihail Asavae  
VERIMAG – Grenoble, FR
- Yu Bai  
TU Kaiserslautern, DE
- Fernando Barros  
University of Coimbra, PT
- Sanjoy K. Baruah  
University of North Carolina –  
Chapel Hill, US
- Guillaume Baudart  
ENS – Paris, FR
- Vagelis Bebelis  
INRIA Grenoble –  
Rhône-Alpes, FR
- Gérard Berry  
INRIA Sophia Antipolis –  
Méditerranée, FR
- Nicolas Berthier  
INRIA Rennes – Bretagne  
Atlantique, FR
- Timothy Bourke  
ENS – Paris, FR
- David Broman  
University of California –  
Berkeley, US
- Benoit Caillaud  
INRIA Rennes – Bretagne  
Atlantique, FR
- Albert Cohen  
ENS – Paris, FR
- Willem-Paul de Roever  
Universität Kiel, DE
- Robert de Simone  
INRIA Sophia Antipolis –  
Méditerranée, FR
- Gwenaël Delaval  
University of Grenoble – LIG, FR
- Stephen A. Edwards  
Columbia Univ. – New York, US
- Peter A. Fritzon  
Linköping University, SE
- Manuel Gesell  
TU Kaiserslautern, DE
- Dan R. Ghica  
University of Birmingham, GB
- Alain Girault  
INRIA Grenoble –  
Rhône-Alpes, FR
- Adrien Guatto  
ENS – Paris, FR
- Nicolas Halbwachs  
VERIMAG – Grenoble, FR
- Jun Inoue  
ENS – Paris, FR
- Xian Li  
TU Kaiserslautern, DE
- Gerald Lüttgen  
Universität Bamberg, DE
- Antoine Madet  
ENS – Paris, FR
- Frédéric Mallet  
INRIA Sophia Antipolis –  
Méditerranée, FR
- Louis Mandel  
College de France, Paris, FR
- Florence Maraninchi  
VERIMAG – Grenoble, FR
- Eleftherios Matsikoudis  
University of California –  
Berkeley, US
- Michael Mendler  
Universität Bamberg, DE
- Christian Motika  
Universität Kiel, DE
- Valentin Perrelle  
IRT SystemX, FR
- John Plaice  
The University of New South  
Wales, AU
- Dumitru Potop-Butucaru  
INRIA – Paris, FR
- Marc Pouzet  
ENS – Paris, FR
- Pascal Raymond  
VERIMAG – Grenoble, FR
- Ingo Sander  
KTH Royal Institute of  
Technology, SE
- Klaus Schneider  
TU Kaiserslautern, DE
- Alex Shafarenko  
University of Hertfordshire, GB
- Chris Shaver  
University of California –  
Berkeley, US
- Wei-Tsun Sun  
INRIA Grenoble –  
Rhône-Alpes, FR
- Reinhard von Hanxleden  
Universität Kiel, DE

