Synchronous Programming—SYNCHRON '04 Dagstuhl Seminar Executive Summary*

Stephen A. Edwards
Columbia University, US

Nicholas Halbwachs
VERIMAG–IMAG, FR

Reinhard von Hanxleden
Universität Kiel, DE

Thomas Stauner
BMW Car IT, DE

November 28–December 3, 2004

This seminar was the 11th in a series of semi-annual workshops on the Synchronous Languages (Esterel, Lustre, and Signal). These languages were invented in the early 1980's to make the programming of reactive systems easier. Benveniste et al. [3] presents a recent survey of the field.

1 The Seminar

The goal of the seminar was to bring together researchers and practitioners of synchronous programming, and furthermore to reach out to relevant related areas and industrial users. With a record participation in this year's SYNCHRON workshop and a broad range of topics discussed, the aims seem to have been well-met. The program of the seminar was composed of around thirty presentations, all of which included extensive technical discussions. The fields covered included synchronous semantics, modeling languages, verification, heterogeneous and distributed systems, hardware/software integration, reactive processing, timing analyses, application experience reports, and industrial requirements.

Particularly successful this year were presentations from the automotive industry. Stefan-Alexander Schneider and Thomas Stauner both discussed issues with real-time software development at BMW. Matthias Hoffmann represented DaimlerChrysler.

*Alain Girault wrote the first draft of this summary.

Dagstuhl Seminar Proceedings 04491
Synchronous Programming - SYNCHRON'04
http://drops.dagstuhl.de/opus/volltexte/2005/195

2 Reactive Systems

Any automatic control software is classified as a reactive system [9]. Indeed, such software must react continuously to their environment. They differ from interactive systems (e.g., operating systems) because their reaction speed is imposed by the environment—because the environment cannot wait. Examples of such software include nuclear power plant controllers, aircraft flight systems, automotive controls, and so forth.

The essential characteristics of reactive systems are

- Criticality: They are highly critical (e.g., time-critical), just like the systems they control are critical.
- Parallelism: At least the parallelism between the system and its environment must be taken into account during the specification. Moreover, it is often convenient for the designer to conceive of the system as a set of parallel components cooperating in order to achieve the desired behaviour.
- Determinism: A deterministic system determines a sequence of output signals from a sequence of input signals in an unique way. This property makes its design, analysis, and debugging much easier. For critical systems, this choice is obvious.

3 The Classical Approach

Classical programming tools are not well-suited to reactive systems programming. Automata-based systems lack high-level parallel programming primitives while asynchronous languages do not respect the intrinsic determinism of reactive systems. Asynchronous language inherit from the field of operating systems and time sharing. In particular, asynchronous parallelism amounts to interleaving. The construct `a||b` (run `a` in parallel with `b`) is either implemented as `a;b` (execute `a` then `b`) or as `b;a`, thus introducing an unwanted non-determinism. This is the case of well-known languages like Occam and Ada, which use the rendezvous-based mechanism inspired by CSP. It is also the case of SDL which uses waiting queues inspired by Petri Nets.

4 Synchronous Abstraction

Synchronous languages are instead based on the simultaneity principle: The construct `a||b` is implemented as the unit `ab`, leaving to the compiler any choice of detailed scheduling. Another way of viewing a synchronous program consists of saying that all the parallel processes evolve simultaneously, sharing a common discrete time scale. This is known as the logical time abstraction: all processes compute one discrete time step simultaneously. This is the approach taken by the synchronous language Esterel [5].

Another approach is to view a synchronous program as a dynamic system, specified as a system of dynamic equations. The job of the synchronous compiler consists, then, of solving this system of equations. This is the approach taken by the synchronous languages Lustre [6, 7] and Signal [10].

5 Advantages of the Synchronous Approach

There are numerous advantages to the synchronous approach. The main one is that the temporal semantics is simplified, thanks to the aforementioned logical time abstraction. This leads to clear temporal constructs and easier time reasoning. Just like ML and Pascal are high-level sequential programming languages in the sense that they are typed and structured, synchronous languages are high-level parallel languages in the sense that they are temporally typed and structured. Programming with ML reduces functional bugs; programming with synchronous languages reduces temporal bugs.

Another key advantage is the reduction of the state-space explosion problem from the discrete logical time abstraction. Synchronous systems evolves in a sequence of discrete steps, and nothing occurs between two successive steps (this is as opposed to models with interleaving concurrent semantics, such as concurrent Java). This makes program debugging, testing, and validating easier. In particular, formal verification of synchronous programs is possible with techniques like model checking. Another consequence is that synchronous language compilers are able to generate automatically embeddable code with performances that can be measured precisely. Hence the reaction time of the software can be known at compile time and can be compared with the desired sampling period. Control engineers can specify and tune their automatic control algorithm with synchronous languages and then rely on the compiler to generate automatically embeddable code, therefore avoiding the tedious and error-prone task of actually implementing the code corresponding to their algorithm.

6 Synchronous Languages

Historically, the first synchronous language is Esterel [4, 5], developped at the Centre de Mathématiques Appliquées (CMA) of École des Mines de Paris, in Sophia-Antipolis, France, and later joined by people from INRIA. It is an imperative language that was originally inspired by CCS and SCCS. Esterel introduces constructs like preemption and communication by synchronous broadcast. It is devoted to the programming of discrete event systems. Esterel Technologies now markets an industrial version of the Esterel compiler. There exists several other synchronous languages. This is just a selection, presented in chronological order:

- Lustre [6] is a data-flow declarative functional language also inspired by Lucid. The Scade tool, initially developed by Verilog and Aerospatiale is based on Lustre. Scade is now marketed by Esterel Technologies.

- Signal [10] is also a data-flow declarative language, but it is relational instead of functional like Lustre. In this sense, it is more general than Lustre. Polychrony is the public domain Signal compiler, while Sildex is the commercial tool developed by TNI-Valiosys.

- Argos [11] is a purely synchronous version of the well known Statecharts formalism [8], which yields a number of advantages. In particular, Argos has a compositional semantics. SyncCharts [1] and Mode Automata are both inspired from Argos.

- Polis [2] is a graphical tool for implementing Codesign Finite State Machines (CFSM). The model of computation behind CFSMs is a set of synchronous FSMs communicating asynchronously; It is therefore known as Globally Asynchronous Locally Synchronous (GALS). The Cierto VCC tool developed by Cadence is based on Polis.

- SL, the Synchronous Language, is a variant of Esterel where hypotheses about signal presence or absence are not allowed. Whether a given signal is present or absent can only be decided at the end of a synchronous instant, hence reaction to a signal is delayed until the next instant. The main advantage is that causality problems are avoided. SL was the starting point of many other synchronous languages such as Sugar Cubes and Junior.

While Esterel, Argos, and SL are more suited to discrete event systems, Lustre, Signal and Polis are very close to the specification formalisms used by automatic control engineers: block diagrams, differential equations, data flow networks, automata, and so on.

7 Industrial Impact

Synchronous languages have recently seen a tremendous interest from leading companies developing automatic control software for critical applications, such as Schneider, Dassault, Aerospatiale, Snecma, Cadence, Texas, and Thomson. For instance, Lustre is used to develop the control software for nuclear plants and Airbus planes. Esterel is used to develop DSP chips for mobile phones, to design and verify DVD chips, and to program the flight control software of Rafale fighters. And Signal is used to develop digital controllers for airplane engines. The key advantage pointed by these companies is that the synchronous approach has a rigorous mathematical semantics which allows the programmers to develop critical software faster and better.

8 Summary

In summary, synchronous programming is an interesting approach for designing and programming automatic control software. Synchronous languages have

a well-founded mathematical semantics that allow ideal temporal constructs as well as formal verification of the programs and automatic code generation. We believe they are ideally suited to programming automatic control software because they are close to the classic specification formalisms used by control engineers, and also because they offer code generation tools that avoid the tedious and error-prone task of implementing the control algorithm after having specified it. These nice features have been confirmed by their recent successes in the automatic control industry.

Participants

This year's participation in the SYNCHRON workshop was the highest ever, which we attribute largely to the excellent Dagstuhl facilities.

Joaquin Aguado , Universität Bamberg
Albert Benveniste, IRISA/INRIA Rennes
Gérard Berry, Esterel Technologies - Villeneuve
Reinhard Budde, Fraunhofer Inst. - St. Augustin
Zbigniew Chamski, Philips Research - Eindhoven
Jean-Louis Colaco, Esterel Technologies - Toulouse
Gwenáél Delaval, INRIA Rhône-Alpes
Stephen A. Edwards, Columbia University
Harald Fecher, Universität Kiel
Abdoulaye Gamatié, Université de Rennes
Peter Gammie, Chalmers UT - Göteborg
Alain Girault, INRIA Rhône-Alpes
Gregor Goessler, INRIA Rhône-Alpes
Claude Helmstetter, VERIMAG - IMPG
Matthias Hoffmann, DaimlerChrysler Research - Berlin
Leszek Holenderski, Philips Research - Eindhoven
Ralf Huuck, National ICT Australia - Eveleigh
Erwan Jahier, VERIMAG - IMPG
Bertrand Jeannet, IRISA/INRIA Rennes
Chiheb Kossentini, VERIMAG - IMPG
Marcel Kyas, Universität Kiel
Ouassila Labbani, Université de Lille
Xin Li, Universität Kiel
Jan Lukoschus, Universität Kiel
Gerald Lüttgen, University of York
Louis Mandel, Université Paris VI
Florence Maraninchi, VERIMAG - IMPG
Christophe Mauras, Université de Nantes
Michael Mendler, Universität Bamberg
Jan Mikac, VERIMAG - IMPG
Matthieu Moy, VERIMAG - IMPG

Barry Norton, University of Sheffield
Gordon Pace, University of Malta
Mihaly Petreczky, CWI - Amsterdam
Dumitru Potop-Butucaru, Université de Rennes
Marc Pouzet, Université Paris VI
Steffen H. Prochnow, Universität Kiel
Pascal Raymond, VERIMAG - IMPG
Martin Richard, Ecole des Mines de Nantes
Jan Romberg, TU München
Eric Rutten, INRIA Rhône-Alpes
Norman R. Scaife, VERIMAG - IMPG
Klaus Schneider, TU Kaiserslautern
Stefan-Alexander Schneider, BMW AG - München
Satnam Singh, Microsoft - Seattle
Cristian Soviani, Columbia University
Thomas Stauner, BMW Car IT
Walid Taha, Rice University
Jean-Pierre Talpin, INRIA Rennes
Olivier Tardieu, INRIA - Sophia Antipolis
Stephan Thesing, Universität Saarbrücken
Christine Vella, University of Malta
David White, University of York
Reinhard Wilhelm, Universität Saarbrücken
Willem-Paul de Roever, Universität Kiel
Robert de Simone, INRIA - Sophia Antipolis
Reinhard v. Hanxleden, Universität Kiel
Jan H. van Schuppen, CWI - Amsterdam

References

[1] Charles André. Representation and analysis of reactive behaviors: A synchronous approach. In *Proceedings of Computational Engineering in Systems Applications (CESA)*, pages 19–29, Lille, France, July 1996.

[2] Felice Balarin, Paolo Giusto, Attila Jurecska, Claudio Passerone, Ellen Sentovich, Bassam Tabbara, Massimiliano Chiodo, Harry Hsieh, Luciano Lavagno, Alberto Sangiovanni-Vincentelli, and Kei Suzuki. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer, Boston, Massachusetts, 1997.

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

[4] Gérard Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In S. D. Brooks, A. W.

Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 389–448. Springer-Verlag, Heidelberg, Germany, 1984.

[5] Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[6] Paul Caspi, Daniel Pilaud, Nicholas Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *ACM Symposium on Principles of Programming Languages (POPL)*, Munich, January 1987. Association for Computing Machinery.

[7] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[8] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[9] David Harel and Amir Pnueli. *On the Development of Reactive Systems*, volume 13 of *NATO ASI Series. Series F, Computer and Systems Sciences*, pages 477–498. Springer-Verlag, 1985.

[10] Paul Le Guernic, Thierry Gautier, Michel Le Borgne, and Claude Le Maire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, September 1991.

[11] F. Maraninchi. The Argos language: Graphical representation of automata and description of reactive systems. In *Proceedings of the IEEE Workshop on Visual Languages*, Kobe, Japan, October 1991.