Axel Poigné, Willem-Paul de Roever,
Gérard Berry, and Amir Pnueli (editors):

Synchronous Languages

Dagstuhl-Seminar-Report; 104
28.11.-2.12.94 (9448)

Report on the Dagstuhl Seminar on

Synchronous Languages

organized by

Axel Poigné (GMD, Sankt Augustin)
Willem-Paul de Roever (Universität Kiel)
Gérard Berry (Ecole des Mines, Sophia Antipolis)
Amir Pnueli (Weizmann Institute, Rehovot)

November, 28th - December, 2nd, 1994

Contents

A Historical Upshot

Synchronous Languages such as LUSTRE, ESTEREL and SIGNAL were conceived in the first half of the eighties by mainly French researchers. Independently, Harel & Pnueli worked on an almost synchronous language, STATECHARTS, as part of the STATEMATE system for the specification of real-time embedded systems (mainly concerning software for the aircraft industry). Independently, Ward & Mellor published in 1985 their 3-volume approach "Structured Development of Real-time Systems", containing important synchronous elements.
The second half of the 1980's was used to:

- Obtain efficient implementations for these languages and to convince users that the resulting systems make sense for specifying real-time embedded systems.

- To resolve semantic problems which are a consequence of the synchronous approach (which states in essence that a system responds in zero time to environmental requests - Berry's "synchrony hypothesis").

  On the latter foundational research by Berry & co-workers on the semantics of ESTEREL, Halbwachs & co-workers on LUSTRE, Le Guernic & Benveniste on SIGNAL, Pnueli, Huizing et al on the semantics of STATECHARTS, stands out.

At the beginning of the nineties the picture changes. Rather than investigating what's there, it is investigated and formulated what should be there, i.e., improvements and alternatives are suggested. E.g., Florence Maraninchi (at Grenoble) proposes a semantics for a new version of STATECHARTS called ARGOS (and implements this) which yields a truly modular notion of refinement of modes (or states) and does away with a number of "paradoxes" due to the synchronous nature of such languages (which require complicated semantical notions for their solution). However, on the other hand the following becomes clear:

- There is no way in these languages to specify that an implementation should satisfy real-time constraints (e.g. in STATECHARTS it is impossible to express that an implementation of a state should run in less than 10 nano seconds).

- Traditional data structures are not integrated, and neither is their refinement, or any notion of refinement, suitably described for those languages.

- There is no notion of object-orientation for such languages. Solutions are proposed (by Rumbaugh et al) and projects applied for to find these (Pnueli & Harel).

Also it becomes clear that ESTEREL/LUSTRE on the one hand, and STATEMATE on the other, serve different purposes. E/L aims at specifying the minimal timed automaton to control a component of a real-time embedded system and become therefore equipped/connected with silicon compilers, while STATEMATE aims at specifying an overall distributed system with many components which are geometrically spread (through their notion of Activity charts). Rather than competing, as done in the period 1985-1990, their complementary roles get accepted and their general relevance to industrial purposes becomes known in wider circles than only aircraft manufactures (DST/Kiel, Siemens/München, Ahlstom-Alcatel/Paris).

Joseph Sifakis gets allotted a complete laboratory - VERIMAG - (about 15 engineers in addition to his own equipe) to build a future specification system for the AIRBUS software in which ARGOS/STATEMATE at the high level of specification where truly distributed processes count are combined with LUSTRE/ESTEREL on a lower level of specification where single chips are specified. Model checkers are combined with the structured automata generated by these systems, e.g. by Werner Damm and co-workers at Oldenburg. Siemens starts collaborating with the SIGNAL group (Benveniste). (Siemens builds the fastest BDD model checker at present and may want to couple it with such systems).

The (mixed) description of combinations of such formalisms becomes a recognized topic of research (e.g. by Poigné & co-workers who have started to combine all the synchronous languages in a modular way). Proof systems and (decidable) assertion languages for the underlying formalisms are developed (de Roever et al., Damm et al.), versions are formulated to truly specify real-time (Huizing, Pnueli).

In Germany independent groups working as these issues meet for the first time jointly (GI Fachgespräch, June 16/17th, 1994, Kiel). DST/Kiel starts funding a synchronous languages based project in combination with a funding agency (Technologiestiftung Schleswig-Holstein) to obtain a tool supporting the design of safety critical systems.

It is in this context of emerging industrial and academic interest within Germany in the specification, prototyping, verification and testing capabilities of real-time embedded systems through the use of synchronous languages and their coupling with advanced tools that we have applied for a Dagstuhl seminar on this topic to bring academia and industry together in Germany and abroad in order to exchange ideas, to discuss their research and to foster collaboration.

This Dagstuhl meeting was a first effort to get together researchers concerned with the different synchronous formalisms to provide a state-of-the-art overview of the field.

Since then the industrial impact of synchronous formalisms has increased. Several industrial applications have been developed being based on the synchronous paradigm and more and more industries have spent considerable time and efforts in evaluating and applying the synchronous approach, e.g.:

- Dassault-Aviation is using extensively ESTEREL for their Rafale airplane.
- Cadence uses ESTEREL as a front-end for their POLIS experimental codesign system.
- Daimler-Benz is making rather extensive experiments with ESTEREL.
- AT&T R&D Indian Hill lab has made successful experiments with ESTEREL and has built a verification system for ESTEREL programs.
- The new generation of Airbus airplanes is programmed in LUSTRE.
- The Hong-Kong subway has been re-engineered in LUSTRE.
- Part of the French nuclear plant safety system is programmed in LUSTRE.
- DASA has developed an environment, called HOSTESS, based on the synchronous data-flow model; they are evaluating the commercial version of LUSTRE, SAO+/LUSTRE, which is also a strong candidate to be used for the development of the software of the Airbus A3XX.
- SNECMA, one of the world's most important manufacturers of jet engines, considers using SILDEX (the professional SIGNAL environment) for next generation development of engine control systems.
- Electricité de France has carried out extensive experiments with SIGNAL.

Data-flow Synchronous Languages[1]

Albert Benveniste, Paul Caspi,
Paul Le Guernic, and Nicolas Halbwachs

In this paper, we present a theory of synchronous data-flow languages. Our theory is supported by both some heuristic analysis of applications and some theoretical investigation of the data-flow paradigm. Our model covers both behavioural and operational aspects, and allows both synchronous and asynchronous styles of implementation for synchronous programs. This model served as a basis to establish the *GC common format for synchronous data-flow languages.* A full version of this paper can be found in LNCS 803 (A decade of Concurrency).

Dataflow Process Networks and their Relationship to Synchronous Languages

Edward A. Lee

Dataflow process networks are a special case of Kahn process networks made up of repeated firings of dataflow actors. This talk explores their use as a formal underpinning for dataflow languages. These languages are not synchronous, although specialized sublanguages have a great deal in common with synchronous languages. So called "synchronous dataflow" captures programs where the dataflow actors have totally predictable firing patterns. Boolean dataflow captures systems where firing patterns depend in predictable ways on Boolean signals in the system. Multidimensional dataflow extends the partial orders to multiple dimensions (as in Lucid). In all cases, the balance equations serve much the same purpose as the clock calculus in synchronous languages, but without imposing a total order on events.

[1]This work has been supported, through several contracts in the framework of C^2A–group, by Ministère de la Recherche et de l'Espace, Ministère de l'Industrie (SERICS), and Ministère de la Défense (DRET).

Synchronous data-flow and lazy functional programming

Paul Caspi

We first show how to translate data-flow programs in a lazy functional language like Haskell or LazyML. This raises the question: What is synchrony within this framework? Our answer is based on the striking analogy which exists between synchronous data-flow compilers, and the deforestation technique introduced by Philip Wadler in lazy evaluation.
We then show how the clock calculus of synchronous data-flow looks like a type system which aims at rejecting non deforestable data-flow networks. Now, expressing these networks in a currified language, and the clock calculus in terms of type checking, allows us to characterize synchronous recursively defined networks.
Since these synchronous networks are not bounded memory and bounded reaction time, this raises the question: what is reactivity here? An answer is: static and tail-recursive synchronous networks.
This leads us to the original concept of tail-recursive block diagrams, which, hopefully, provides a fully visual, well-structured extension to block-diagram programming.

Timing and Retiming Statecharts

Adriano Peron and Andrea Maggiolo-Schettini

We generalize Statecharts by associating delays and timeouts with transitions, and durations with communicated signals. Occurrences of transitions are related with a dense time domain. We consider how behaviour changes when temporal features are changed. We investigate the effects of temporal shift, slow-down and speed-up of the environment. We study the conditions under which non-discrete environments can be equivalently replaced by discrete environments (and viceversa). We investigate also conditions under which durations can be associated with instantaneous behaviours.

Towards a Process Algebra of Statecharts [2]

Andrew Uselton

Our goal is to develop a process algebra, SPA, for statecharts that is in strict compliance with the semantics of Pnueli and Shalev ("What is in a Step: On the Semantics of Statecharts," in Theoretical Aspects of Software, LNCS 526). In particular we would like to isolate all communication in a statechart-style broadcast merge operator. Our motivations are twofold. First, we believe that the notion of hierarchy in statecharts is useful and should be available independent of the statechart formalism. Second, it is well known that the various notions of communication have consequences in the semantics. Isolating communication in the merge operator allows for an easy and uniform way of comparing, and in particular experimenting with, alternative semantics.
Here we define a simple algebra SA with two operators. One is a k-place or-operator for constructing finite state automata-like terms, and the other is a binary (associative and commutative) and-operator. We formalize the step semantics as a mapping $\Psi : SA \rightarrow LTS$, where LTS is a set of labeled transition systems. Two statecharts A and A' are equal, written $A \hat{=} A'$ if $\Psi(A) \approx \Psi(A')$. It is well known that the step semantics is not compositional – i.e. $\hat{=}$ is not a congruence – and we present a example showing this. Next we introduce an alternative semantic mapping $\Psi_{>} : SA \rightarrow LTS_{>}$ for the "orderly step semantics." The semantic labels in $LTS_{>}$ are equipped with enough ordering information to make this semantics compositional, and again two statecharts A and A' are equal, written $A \cong A'$ if $\Psi_{>}(A) \approx \Psi_{>}(A')$. The careful choice of equivalence over labels gives the (strongly conjectured) result that $\cong$ is the largest $\hat{=}$ respecting congruence over SA (the proof is in preparation).
We conclude with a quick look at statecharts process algebra SPA and its relation to SA. SPA uses the operators *prefix*, *choice*, and *fix* in their usual way for specifying finite state automata. A new operator *state_refinement* is introduced to capture the statechart notion of hierarchy, and an appropriate definition of *merge* is introduced to capture the broadcast style of communication.

[2]This work is in collaboration with Scott Smolka.

A Comparison of Statecharts Variants

Michael von der Beeck

The Statecharts formalism supports the development of intuitive graphical specifications for reactive systems. Nevertheless, some serious problems became apparent in the original Statecharts formalism so that many different Statecharts variants were proposed to overcome them. These problems are thoroughly described and approaches for solving them are evaluated. Furthermore, a set of distinctive features is elaborated which is used for a detailed comparison of the Statecharts variants. Finally, the feature set is used to characterize a new hypothetical Statecharts variant.

Formal Semantics for Ward & Mellor's Transformation Schemas and its Application to Fault-Tolerant Systems

Carsta Petersohn, Jan Peleska,
Cornelis Huizing, and Willem-Paul de Roever

One of the Structured Analysis and Design methods is that of Ward & Mellor aiming at giving a specification of software which is independent of, and considerably more abstract than, the code eventually produced. Although widely used in industry, its description [WM85, War86] contains a number of inconsistencies. Yet W&M's method contains at least sufficient indications for us to try to *reconstruct* its intended meaning. We show that with the formal methods developed for the definition and analysis of so-called *synchronous* languages a consistent and precise semantics can be reconstructed for the W&M method. Incompleteness in description can be identified and removed. We give an example of the main flaws in W&M's definition of the semantics of transition diagrams, discuss our suggestions to resolve them and sketch a precise semantics for the Essential Model of W&M's method. A considerable complete formal semantics is given in [PHPdR94a]. Also a formal definition of W&M's semantics enables the development of a symbolic interpreter to animate TS. Moreover, we argue the need for a family of semantics for different application areas using a 'real-world' example from the field of fault tolerance as discussed in [PHPdR94b].

References

[PHPdR94a] C. Petersohn, C. Huizing, J. Peleska and W.-P. de Roever. Formal Semantics for Ward & Mellor's TRANSFORMATION SCHEMAS. In D. Till, editor, *Sixth Refinement Workshop* Springer Verlag, 1994.

[PHPdR94b] C. Petersohn, C. Huizing, J. Peleska and W.-P. de Roever. Formal Semantics for Ward & Mellor's TRANSFORMATION SCHEMAS and the Specification of Fault -Tolerant Systems. In K. Echtle and D. Hammer and D. Powell, editors, *Dependable Computing – EDCC-1 (First European Depenable Computing Conference)* , volume 852 of *LNCS*, pages 59–76,1994.

[War86] Paul T. Ward. The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing. *IEEE TSE*, SE-12(2):198–210, February 1986.

[WM85] Paul T. Ward and Stephen. J. Mellor. *Structured Development for Real-Time Systems, Vol.1 – Introduction & Tools.* Yourdon Press, Prentice-Hall, Englewood Cliffs, 1985.

[War86] Paul T. Ward. The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing. *IEEE TSE*, SE-12(2):198–210, February 1986.

[WM85] Paul T. Ward and Stephen. J. Mellor. *Structured Development for Real-Time Systems, Vol.1 – Introduction & Tools.* Yourdon Press, Prentice-Hall, Englewood Cliffs, 1985.

Exact Causalities for Esterel (part I): Electrical Causality Analysis

Gérard Berry

Esterel is a synchronous programming language where communication is done by instantaneously broadcasting signals and instantaneously testing for this values. This may lead to paradoxes like in

```
Present S else emit S
```

which should emit `S` if, and only if `S` is not received, contradicting the basic broadcasting assumption.

The key to rejecting such programs in Esterel is to transform Esterel programs into boolean circuits and to perform causality analysis on the resulting circuit that may have causality cycles. The causality analysis is done by replacing classical logic by a constructive logic, or equivalently by Scott Booleans. We give an electrical characterization of circuits that are causal in this sense, by showing that a circuit is causal iff it is delay insensitive. We give a BDD-based algorithm to compute formally whether a circuit is causal.

We finally analyze the Esterel VG$^+$ compiler that implements the notion of electrical causality.

Exact Causalities for Esterel (part II): Boolean Causality Analysis

Olivier Ploton

As described in the talk about electrical causality, several Esterel programs must be rejected as non causal. A basic choice is to decide whether or not to accept the following program: `output S;`

```
present S then emit S else emit S end
```

Electrical causality rejects this program; boolean causality accepts it. The boolean causality analysis consists in translating en Esterel term into a boolean function instead of a gate network (circuit). The causality condition for a local signal S is expressed as the functional property "The emission of S does not depend on its presence". The boolean causality analysis leads to a compositionnal semantics. It enjoys nice debugging properties: observation (debug tracing) does not disturb causality. Both electrical and boolean causalities are implemented using efficient, BDD based algorithms, in the experimental Esterel v5 compiler.

BAC: A Boolean Automaton Checker

Nicolas Halbwachs

BAC is a BDD-based symbolic verification tool devoted to "Boolean automata". A Boolean automaton is a finite state machine, where the next state is uniquely determined by the current state and the current values of inputs. A state is a vector of Boolean variables. The behavior of the automaton can be restricted by an "assertion", a Boolean formula which must be always true during any execution of the automaton. More precisely, the behaviors of the machine are exactly all the *infinite* behaviors of the automaton (without assertion) which continuously satisfy the assertion. An "invariant" can also be specified, which will be verified by the tool. The transition function, the assertion and the invariant are given as Boolean formulas involving current state variables, current input variables, and possibly local variables which must be defined in turn. The definitions of local variables may present loops.
Such automata appeared very convenient to model circuits and the control part of synchronous programs.
The current functionalities BAC include

- checking the consistency of looping definitions, and solving them, when possible;

- making the assertion "executable", i.e., strengthening the assertion in such a way that it is possible to check, in each state, which are the transitions which make possible a future infinite execution;
- computing the set of variable configurations which can be reached from a given set of initial states, taking into account the assertion;
- checking the satisfaction of the given invariant, by either forward or backward symbolic traversal of the reachable states. If the verification fails, BAC returns a diagnosis, which is the set of shortest executions leading to invariant violation.

The paper and the tool are available by public ftp at
`imag.fr:/pub/SPECTRE/LUSTRE/PAPERS/bac.ps.gz`
and
`imag.fr:/pub/SPECTRE/LUSTRE/BAC.tar.gz`.

Boolean Automata for Implementing Pure Esterel

Axel Poigné

A new compilation scheme is proposed for the synchronous language Esterel. The compilation is based on the new intermediate format of Boolean automata. Boolean automata have two kinds of statements:

- $s \Leftarrow \phi$: the signal s is emitted if the condition ϕ is satisfied, and
- $h \leftarrow \phi$: the control register is set for the next instant if ϕ is satisfied.

A Esterel program is represented by a set $\mathcal{P}$ of such 'equations'; the former compute the status of all signals in the system and are solved simultaneously on each tick of the clock; the latter are guarded assignments which reset the state of the automaton's finite control. The translation is algebraic by nature using 'system' signals for supporting particular constructs. E.g. a signal α is raised as a start signal, the signal ω denotes termination. Then sequential composition $\mathcal{P}; \mathcal{Q}$ of two Boolean automata $\mathcal{P}$ and $\mathcal{Q}$ is defined by $\mathcal{P} \lhd \{\omega \leftarrow \mathit{false}\} \vee \mathcal{Q}[\mathcal{P}.\omega/\alpha]$ where $\mathcal{P}.\omega$ denotes the termination condition for $\mathcal{P}$ which is substituted for the start signal α, i.e. $\mathcal{Q}$ starts if $\mathcal{P}$ terminates. $\mathcal{P} \lhd \{\omega \leftarrow \mathit{false}\}$ overrides the equations for the termination signal in $\mathcal{P}$ to $\omega \leftarrow \mathit{false}$, hence sequential composition terminates only if $\mathcal{Q}[\mathcal{P}.\omega/\alpha]$ terminates.
A full paper is available by web: `http://set.gmd.de/EES/Papers/E2BA.ps.gz`.

Boolean automata: a compact representation of Pure Esterel programs

Leszek Holenderski, Axel Poigné

A simple translation schema of Pure Esterel to Boolean automata, although useful for proving correctness of the translation, may lead to an automaton whose size is exponential w.r.t. the size of a source Esterel program. We show two optimizations to the translation schema which guarantee that the size of the resulting automaton is linear in size of the Esterel program, provided that the program does not use the Esterel's hiding statement 'signal S in P end'. Unfortunately, for the programs which use the hiding statement we still may get automata whose size grows exponentially, in the worst case, with the number of nested 'signal' statements. Fortunately, such worst cases should not occur in practice.

A Compositional Proof System for Statecharts Based on Symbolic Timing Diagrams

Johannes Helbig

We present a proof system for statecharts that allows compositional reasoning about safety and liveness properties expressed in terms of symbolic timing diagrams. Since the latter can be considered as a visual dialect of linear-time temporal logic, the assertion language is, by contrast to existing approaches, decidable. Compositionality allows to reduce the verification task to subtasks manageable by symbolic model checking. The particular compositional structure we present supports an incremental design style; this, and the visual nature of specification languages and proofs enhances industrial applicability of the verification method.

Statecharts, Transition Structures and Transformations

Adriano Peron

Statecharts are state-transition machines endowed with hierarchy on states and parallelism on transitions. The relationship between structure over states and behaviour is investigated. It is shown that a statechart is described by a pair of relations over transitions (a transition structure), the former describing causality and the other describing a notion of asymmetric independence. A statechart can be effectively constructed from its transition structure. Transition structures corresponding to a subclass of Statecharts are characterized. Natural notions of morphisms among transition structures allow to define classes of statechart transformations which preserve behaviour.

A Comparison of Ward & Mellor's TRANSFORMATION SCHEMA with STATE- & ACTIVITYCHARTS

Jan Peleska[3]

A comparison is made between Structured Methods, as represented by the Essential Model of Ward&Mellor's Transformation Schemas, and the Statemate specification language consisting of State- and Activitycharts. The comparison is based on the languages' semantic properties. An example from the field of fault-tolerant systems serves as a "benchmark problem" to investigate the practical applicability of both Transformation Schemas and Statemate for meaningful "real-world" systems. It is motivated that a family of semantics should be introduced for Structured Methods and similar CASE methods. This family will allow the designer to adjust the specification language used in an optimal way to the specific semantic properties of the target system.

[3]In collaboration with: Kees Huizing, Carsta Petersohn

A Compositional Semantics of ESTEREL in Durational Calculus

R.K. Shyamasundar[4]

In this talk, a compositional semantics of pure ESTEREL is presented using a variant of durational calculus called Mean Value Calculus (MVC). MVC provides an axiomatization of ESTEREL and the rules can be used to prove properties of the programs. In the talk, it is shown how the semantics can be used to study the expressive power of the operators of the language, and in particular it is shown that the recently added operator, *suspend* does indeed increase the power at the behavioural level. The efforts towards obtaining a completeness-notion for the operators of the language and extensions of the semantics to the recent asynchronous extensions of ESTEREL namely, CRP, are also discussed.

Implementation of Communicating Reactive Processes.

S. Ramesh

Communicating Reactive Processes (CRP) is a new paradigm proposed to combine the capabilities of both synchronous and asynchronous programs. Synchronous programs are useful for embedded reactive applications while asynchronous programs have been useful in distributed systems. CRP, exhibiting both synchronous and asynchronous behaviors, has a wider applicability. In this talk, the issues behind the implementation of CRP are discussed and a new efficient implementation is suggested. The implementation is a simple extension of a protocol, proposed by the author, for implementing CSP. The correctness of the implementation follows from that of the CSP implementation.

[4]Joint work with Paritosh Pandya and Y.S. Ramakrishna

Synchronous Automaton Compositions

Florence Maraninchi

We propose to study various synchronous communication mechanisms (ranging from rendez-vous to the chain-reaction mechanism of Statecharts) by expressing their semantics in a general framework where *maximal information models* are associated to programs. The bisimulation of such models is guaranteed to be a congruence for the language operators. The models are labeled transition systems with structured labels. We show how to define a *program congruence*, coarser than the abovementioned bisimulation, out of a *label congruence*. Finally, we propose to define label congruences out of *label abstractions*.
A label abstraction which induces a program congruence clearly shows which part of the maximal-information is relevant for the communication mechanism under study. In other words, it shows which amount of information has to be kept in what we call the model of a program, for the semantics to be compositional.
This is related to the pragmatic point of view that, for a language to be usable in practise, its semantics should be compositional, and the model of a program should be significantly more abstract than the program itself.

An Intermediate Model for Reactive Languages

Cornelis Huizing, Rob Gerth

We wanted to give a denotational semantics for Statext. What's Statext? Well, this is the textual version of Statecharts (proposed by Amir Pnueli & Yonit Kesten, 1992), you could say. But it's more: there is a powerful timing operator and the semantics of the macro step is highly nondeterministic.
Anyway, when we defined this denotational semantics we discovered the same patterns that we'd seen before in other semantics, so we tried to identify an intermediate model. Translation ot this model (or algrebra, if you like) should capture most of the pecularities of the language, whereas translation from this model to a history semantics is straightforward. For Statext, at least, this works. The semantics of the macro step (the step in which the computation takes place in the form of micro steps, during which no time passes) became quite clear, even to some in the audience.
Hallmarks of the intermediate model are:

- one operator for parallelism, non-deterministic choice, and pre-emption!
- a consistency operator, which makes the macro steps globally consistent in the sense that you can consider every micro step as being taken under the same set of current events (in one macro step).

At the end, the lecturer managed to pop in a stick about "true interleaving", which could be considered as a pun, but also as a statement that interleaving is a misnomer.

A Calculus of Stochastic Systems for the Specification, Simulation, and Hidden State Estimation of Mixed Stochastic/Non-stochastic Systems

Albert Benveniste, Bernard C. Levy,
Eric Fabre, and Paul Le Guernic[5]

This is the abstract and introduction of a full paper. This full paper is available, either as an IRISA research report through ftp at ftp.irisa.fr in techreports/1994/PI-837.ps.Z, or in final version, as INRIA res. rep. 2465, accessible in ftp.inria.fr. The final version will appear in Theoretical Computer Science, 1995.

In this paper, we consider *mixed systems* containing both stochastic and non-stochastic[6] components. To compose such systems, we introduce a general combinator which allows the specification of an arbitrary mixed system in terms of elementary components of only two types. Thus, systems are obtained hierarchically, by composing subsystems, where each subsystem can be viewed as an "increment" in the decomposition of the full system. The resulting mixed stochastic system specifications are generally not "executable", since they do not necessarily permit the incremental simulation of the system variables. Such a simulation requires compiling the dependency relations existing between the system variables. Another issue involves finding the most likely internal states of a stochastic system from a set of observations. We provide a small set of primitives for transforming mixed systems, which allows the solution of the two problems of incremental simulation and estimation of stochastic systems within a common framework. The complete model is called CSS (*a Calculus of Stochastic Systems*), and is implemented by the SIG language, derived from the SIGNAL synchronous language. Our results are applicable to pattern recognition problems formulated in terms of Markov random fields or hidden Markov models (HMMs), and to the automatic generation of diagnostic systems for industrial plants starting from their risk analysis.

[5]Eric Fabre is with IRISA-INRIA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France, Fabre@irisa.fr. Bernard C. Levy is with the Dept. of of Electrical and Computer Engineering, Univ. of California, Davis, CA 95616, USA, levy@ece.ucdavis.edu.

[6]Throughout this paper, we use the word "non-stochastic" to refer to systems which have no random part. In control science or statistics, such systems would be called "deterministic" as opposed to "stochastic" ; however this name would be misleading in computer science, where "deterministic" vs. "nondeterministic" has a totally different meaning. This is why we decided to use the word "non-stochastic" here.

Object-oriented Statecharts

Amir Pnueli

This paper reports on an effort to develop an integrated set of diagrammatic languages for modelling object-oriented systems, and to construct a supporting tool. The main goal is for models to be intuitive and well-structured, yet fully executable and analyzable, enabling automatic synthesis of usable and efficient code in languages such as $C^{+}+$. At the heart of the modeling method is the language auf statecharts for specifying object behavior, and a hierarchical OMT-like language for describing the structure of classes and their inter-relationships, that we call O-charts. Objects can interact by event generation, or by direct invocation of operations. In the interest of keeping the exposition manageable, we concentrate here on a rather simple framework, that we feel adequately represents the issues fundamental to the entire effort. It leaves out several technically involved topics, such as inheritance, multiple-thread concurrency and active objects, which will be described elsewhere.

SIGNAL: Compilation, Composition & Parallel Implementation

Olivier Maffeis[7]

We focus on the composition of SIGNAL programs to detect deadlocks at the specification level as well as to prevent implementation inference from creating them. First we introduce the abstract representation of SIGNAL programs which is constituted of a system of boolean equations connected to a dependence graph. This abstract representation is a generalization of the notion of Directed Acyclic Graphs. Then an abstraction of this representation is defined to enable the detection of deadlocks by composition. Over this abstraction we study the composition of processes and define a congruence. Furthermore we define the new notion of *fully deadlock consistent scheduling* which preserves the composition capabilities at the implementation level and thereby enables the separate implementation of SIGNAL programs. We end by using the notion of fully deadlock consistent scheduling to cluster nodes; this constitutes a key step towards the parallel implementation of non regular (i.e., complex controlled) applications specified in SIGNAL.

[7]Joint work with Paul Le Guernic

The SL Synchronous Language

Frederic Boussinot

We present a new synchronous programming language named SL based on Esterel, in which hypothesis about signal presences or absences are not allowed. Thus, one can decide that a signal is absent during one instant only at the end of this instant, and so reaction to this absence is delayed to the next instant. Esterel "causality problems" are avoided at the price of replacing strong preemptions by weak ones. An operational semantics based on rewriting rules has been built and an implementation is described which allows either to directly execute programs, or to produce automata.

Mixing Reactive Synchronous and Asynchronous Languages: a step towards the Hybrid Systems

Olivier Roux and Martin Richard

We present a proposal for the conjunction of the asynchronous and the synchronous paradigms. This is achieved through the ELECTRE language [CR95].
The motivation is to cope with reactive programs with lengthing actions. These actions can be interrupted by some memorizable occurrences of events, and then possibly restarted at the beginning or resumed at the interruption point.
Indeed, such programs are featured both by the discrete event switches (instantaneous actions which stand for the synchronous part) and by continuous evolutions (lengthing actions which stand for the asynchronous part). The combination of these two approaches (namely the synchronous one and the asynchronous one) makes it possible to compile the so-called *ambisynchronous* programs which are composed of ESTEREL code joined to ELECTRE code according to the parallel operator [RR94a].
Moreover, temporal requirements as well as behavioral properties can be checked upon these reactive *hybrid* (discrete/continuous) programs. We show that the ELECTRE programs can also be compiled together with temporal specifications in order to produce stopwatch automata [RR94b] which are a kind of linear hybrid automata where bounded response and bounded reachability are decidable.
As a matter of fact, we assert that the ambisynchronism approach is a step towards the specification of hybrid (discrete/continuous) systems.

References

[CR95] Franck Cassez and Olivier Roux. Compilation of the ELECTRE reactive language into finite transition systems. *Theoretical Computer Science*, 144, june 1995. to appear.

[RR94a] Martin Richard and Olivier Roux. Conjonction de codes réactifs synchrones et asynchrones. In *Actes de la conférence Rencontres du parallélisme RenPar'6*, ENS Lyon, june 1994.

[RR94b] Olivier Roux and Vlad Rusu. Verifying time-bounded properties for electre reactive programs with stopwatch automata. In *Workshop on Hybrid Systems and Autonomous Control (WHSAC)*, Cornell University, Ithaca (New-York, USA), october 1994. to appear in LNCS.

On Designing A Synchronous Object-Oriented Language

Reinhard Budde

The systems we consider are real-time systems to be embedded into larger applications. Typically these systems are reactive: They are stimulated by signals from the environment and they give feedback to the environment. The reaction time is dictated by the environment. The systems have to be reliable. Proofs or at least a sound argumentation that specified reaction times can be met must be possible.
A preferred strategy in industry is to take a standard micro-controller with lots of peripheral devices integrated on the same chip and to customize it by software. Far the most microprocessors produced are used in applications of this kind. Now software is dominant to glue components together and supply flexible functionality.
Object-orientation is needed for such real-time systems - to hide hardware/software design decisions, - to master the development of variants, - to achieve maintainable, flexible systems architectures, - to lay a foundation for proving properties of the system. The basic properties we need are - encapsulation, - inheritance, - polymorphism, The object-oriented basis of our design language and framework under development is similar to the concepts of the programming language Eiffel.
SYSTEM DYNAMICS
It is well-established to use object-orientation to describe a systems (static) architecture. Many methods have been developed and are established now. But there is a on-going debate of how the dynamics of an object-oriented systems have to be described. Finite state machines (Booch) and Statechart-like description (Rumbaugh et.al.) are proposed. Explicitly or implicitly these proposals are based on a synchronous metaphor and a broadcast of

signals. At a first glance this seems not to fit to the client-supplier-model underlying object-oriented architectures.
Means for describing the reactive behavior of an object-oriented system is a prerequisite for coping with real-time problems. Thus we added to the class definition a reactive part (such classes are called reactive classes). The textual variant of this reactive part is based on Esterel, the graphical on Argos.
Objects, that are direct instances of a reactive class are called reactive objects. A system executes these objects in parallel. They communicate exclusively by signals. Signals are broadcasted. The notion of causality in the synchronous setting is adatpted to this framework, and a notion of consistency (absence of time races) added.
THE LANGUAGE eE (embedded Eifel) ¡– one F only!
Our framework is a smooth integration of the paradigm of perfect synchronization and the object-oriented paradigm. Its focus is on non-preemptive descriptions, but allows non-critical parts of the system being preempted.
Reaction times are derived from the systems by worst-case bottom-up calculations. These values are used for prescheduling the time-critical parts. The time-uncritical parts are scheduled at run-time.
The compiler under development is based on - compilation of the reactive part to Boolean Automata - dataflow analysis in the data parts Model checking (done with SMV and auto) is integrated to prove assertions about classes, e.g. the exclusivity of two signals in any instant at compile time to avoid run-time checks.

An Object Execution Model for Synchronous Models

Frederic Boulanger, Guy Vidal-Naquet

We present an execution model that allows synchronous modules to be represented by objects that may communicate synchronously or asynchronously. This model is intended to be used in the object-oriented design of complex applications that mix synchronous parts with asynchronous ones, the communication between these parts being asynchronous. Furthermore, the synchronous parts may be developped in several synchronous languages, or even implemented in hardware. Our model allows synchronous communication between those separatly compiled parts, assuming there are no dependency loop.
Object orientation allows dynamic creation and destruction of objects. This is supported by our model, along with dynamic changes in the synchronous communication network. This leads to the notion of reaction of a clock (set of synchronous objects that communicate synchronously). The clock reacts

to inputs signals at instant t by producing outputs and determining its state at instant t+1 (new set of objects, new communication network).
The nature of asynchronous communications is intentionnaly not imposed because it depends on the application domain. However, we propose three mecanisms: communication by last value (when the full history of events is not meaningful), communication through unbounded buffer, where overflow of the physical buffer is an error (when the loss of the least event is critical), and communication through bounded buffers, where overflow leads to the loss of the oldest values (when the behaviour is not critically impaired by the loss of some samples).
This execution model is actually implemented by two tools and a C++ class library. The first tool, occ++, translates oc code (automaton description) into a C++ class. The second tool, mdlc, compiles descriptions of composite modules written in the Module Description Language. The libSync library provides the excution machine for synchronous objects. This implementation works on any Unix machine, and it has been succesfully ported to VxWorks to drive physical devices in a real-time environment.

Sequential Function Charts (Grafcet): A Synchronous Semantics

Charles ANDRE

Sequential Function Charts (SFC) are widely used in industrial applications. The IEC 848 Standard (1987) specifies this model. The semantics given in the Standard is an informal one.
SFC are synchronous models very convenient for industrial applications. Synchronous languages are emergent solutions to reactive system programming. Our goal is to combine SFC and the imperative synchronous language ESTEREL and apply them to control system programming.
In a first part, we exhibit some ambiguities underlying the "classical" definition of SFC. We then introduce a formal model for SFC and the principle of a semantics compatible with ESTEREL's semantics.
Problems of instantaneous cycle, non determinism and causality are analysed. A "micro-step" semantics is proposed.
Endowed with this semantics, SFC and ESTEREL programs can be mixed. SFC can be, now, integrated in the software environment for synchronous programming.

A Synchronous Data-Flow Proof Method for the Grafcet Language

Lionel Marce

The Grafcet is a graphical programming language used mainly in the field of automatisms. Its powerful control structures can express particularly parallelism. This last notion is often awkward to handle. So it is necessary to be able to prove some properties on systems programmed in Grafcet.
We define Grafcet semantics without or with stability research (WSR) in using the synchronous language SIGNAL. We associate an equation SIGNALwith each step of the Grafcet program. This equation defines the conditions of activation and desactivation of the step according to its environment. These equations are put together into one process. In the case of WSR an other process determines if the reached situation is stable or not.
The validation is brought through a polynomial dynamical system obtained directly from Signal equations defined on the field F3. The proofs are realized on boolean signals. These equations are put away in 3 classes: one for the initial conditions, one other for the constraints , and the last for the evolution. We use some basic operators of the tool Sigali to verify properties directly on the polynomial dynamical equations or we use the observer principle.

An example will be given, and some results on the limits due to the combinatorial explosion.

2-adic Numbers and Circuits

Gérard Berry

We use the 2-adic number theory as a semantic tool for describing the semantics of sequential circuits. A 2-adic number is a bitstream interpreted as a number written low-order bits first. We present the bolean algebra and ultrametric structure of 2-adic numbers, and we use this structure to build new arithmetic circuits. Finally, we show a normal form theorem for sequential circuits: any bitwise function can be computed by a sequential circuit in normal form.

Verification of linear hybrid systems by means of convex approximations

Nicolas Halbwachs, Yann-Eric Proy, and Pascal Raymond[8]

We present a new application of the abstract interpretation by means of convex polyhedra, to a class of hybrid systems, i.e., systems involving both discrete and continuous variables. The result is an efficient automatic tool for approximate, but conservative, verification of reachability properties of these systems.
This paper has been published in SAS'94, International Symposium on Static Analysis, LNCS 864, Springer Verlag.

[8]Miniparc-Zirst, 38330 - Montbonnot, France e-mail: {Pascal.Raymond,Yann-Eric.Proy}@imag. fr

Dagstuhl Seminar 9448 Participants

Charles **André**
Laboratoire I 3S
41 boulevard Napoleon III
F-06041 Nice
France
email: andre@cma.cma.fr
tel.: +33-93.21.79.57

Albert **Benveniste**
INRIA-IRISA
Campus de Beaulieu
Avenue du Général Leclerc
F-35042 Rennes Cedex
France
email: benveniste@irisa.fr
tel.: +33-99-84-72-35
fax: +33-99-84-71-71

Gérard **Berry**
Ecole des Mines de Paris
Centre de Mathématique Appliquées
Place Sophie Lafitte
F-06560 Valbonne
France
email: berry@cma.cma.fr
tel.: +33-93.95.74.68

Reinhard **Budde**
GMD - Forschungszentrum
Informationstechnik GmbH
SET - EES
D - 53754 St. Augustin
Germany
email: budde@gmd.de
tel.: +49-2241-14 24 17

Frédéric **Boulanger**
Supelec
Plateau de Moulon
F-91192 Gif sur Yvette
France
email: frederic.boulanger@supelec.fr
tel.: +33-1-69.85.14.91

Frederic **Boussinot**
Ecole Nationale Superieure des Mines
Centre de Mathématiques Appliquées
Place Sophie Lafitte
F-06904 Sophia Antipolis Cedex
France
email: fb@cma.cma.fr
tel.: +33-1-93.95.74.75

Paul**Caspi**
Miniparc - Zirst
Laboratoire Verimag
Rue Lavoisier
F-38330 Montbonnot St Martin
France
email: paul.caspi@imag.fr
tel.: +33-76.90.96.33

Werner **Damm**
Universität Oldenburg
FB 10 - Informatik
Ammerländer Heerstr. 114
D-26111 Oldenburg
Germany
email:
werner.damm@informatik.uni-oldenburg.de
tel.: +49-441-798-45 02

Willem P. **de Roever**
Christian-Albrecht-Universität Kiel
Inst. für Informatik und Prakt. Mathematik Haus II
Preusserstrasse 1-9
D-24105 Kiel
Germany
email: wpr@informatik.uni-kiel.de
tel.: +49-431 56 04 71 / 74

Nicolas **Halbwachs**
Miniparc - Zirst
Laboratoire Verimag
Rue Lavoisier
F-38330 Montbonnot St Martin
France
email: Nicolas.Halbwachs@imag.fr
tel.: +33-76.90.96.37

Johannes **Helbig**
Universität Oldenburg
FB 10 - Informatik

Ammerländer Heerstr. 114
D-26111 Oldenburg
Germany

Leszek **Holenderski**
GMD - Forschungszentrum
Informationstechnik GmbH
SET - EES
D - 53754 St. Augustin
Germany
email: holenderski@gmd.de
tel.: +49-2241-14 24 12

Cornelis **Huizing**
Eindhoven University of Technology
Dept.of Mathematics and Computing Science
P.O. Box 513
NL-5600 MB Eindhoven
The Netherlands
email: keesh@win.tue.nl
tel.: +31-40-47 41 20

Paul **Le Guernic**
Université de Rennes
INRIA- IRISA
Campus de Beaulieu
Avenue du Général Leclerc
F-35042 Rennes Cedex
France
email: leguernic@irisa.fr
tel.: +33-99.84.72.42

Edward A. **Lee**
University of California at Berkeley
Dept. of Electrical Engineering
and Computer Science
507 Cory Hall
Berkeley CA 94720
USA
email: eal@eecs.berkeley.edu

Thomas **Lindner**
FZI - Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe
Germany
email: lindner@fzi.de
tel.: +49-721-9654-616

Olivier **Maffeis**
GMD - Forschungszentrum
Informationstechnik GmbH
SET - EES
D - 53754 St. Augustin
Germany
email: maffeis@gmd.de
tel.: +49-2241-14 22 67
Andrea **Maggiolo-Schettini**
Università di Pisa
Dipartimento di Informatica
Corso Italia 40
I-56125 Pisa
Italy
email: maggiolo@di.unipi.it
tel.: +39-50-887-259

Florence **Maraninchi**
Miniparc - Zirst
Laboratoire Verimag
Rue Lavoisier
F-38330 Montbonnot St Martin
France
email: Florence.Maraninchi@imag.fr
tel.: +33-76.90.96.42

Lionel **Marce**
UBO UFR Sciences
Dept. Informatique
6 avenue Victor Le Gorgeu
F-29285 Brest
France
email: marce@univ-brest.fr
tel.: +33-98.01.63.89

Erich **Mikk**
Christian-Albrecht-Universität Kiel
Inst. für Informatik und Prakt. Mathematik Haus II
Preusserstrasse 1-9
D-24105 Kiel
Germany

Matthew **Morley**
GMD - Forschungszentrum
Informationstechnik GmbH
SET - EES
D - 53754 St. Augustin
Germany
email: morley@gmd.de
tel.: +49-2241-14-22 67

Klaus **Nökel**
SIEMENS AG
ZFE T SE 1
D-81730 München

Germany
email: noekel@zfe.siemens.de
tel.: +49-89-636-4 36 86

Jan **Peleska**
Christian-Albrecht-Universität Kiel
Inst. für Informatik
und Prakt. Mathematik Haus II
Preusserstrasse 1-9
D-24105 Kiel
Germany
email: jap@informatik.uni-kiel.de
tel.: +49-431 55 28 82

Adriano **Peron**
Università di Udine
Dip. di Matem. e Informatica
via Zanon 6
I-33100 Udine UD
Italy
email: peron@dimi.uniud.it
tel.: +39-432-272 208

Carsta **Petersohn**
Christian-Albrecht-Universität Kiel
Inst. für Informatik und Prakt. Mathematik Haus II
Preusserstrasse 1-9
D-24105 Kiel
Germany
email: cp@informatik.uni-kiel.d400.de
tel.: +49-431-56 04 79

Olivier **Ploton**
Ecole Nationale Superieure des Mines
Centre de Mathématiques Appliquées
Place Sophie Lafitte
F-06904 Sophia Antipolis Cedex
France
email: ploton@cma.cma.fr
tel.: +33-1-93-95-74-93

Amir **Pnueli**
Weizmann Institute
Departement of Appl. Mathematics
P.O. Box 26
76100 Rehovot
Israel
email: amir@wisdom.weizmann.ac.il
tel.: +972-8-343434

Axel **Poigné**
GMD - Forschungszentrum
Informationstechnik GmbH
SET - EES
D - 53754 St. Augustin
Germany
email: poigne@gmd.de
tel.: +49-2241-142440

S. **Ramesh**
Indian Institute of Technology
Department of Computer Science
& Engineering
Bombay 400 076
India
email: ramesh@cse.iitb.ernet.in
tel.: +91-22-578-25 45 x27 01

Martin **Richard**
Ecole des Mines de Nantes
Dept. d'Informatique
4 Rue Alfred Kastler
F-44070 Nantes Cedex 03
France
email: richard@info.emn.fr
tel.: +33.51.85.82.16

Olivier **Roux**
Laboratoire d'Automatique de Nantes
Ecole Centrale de Nantes
1 Rue de la Noe
F-44072 Nantes Cedex 3
France
email: roux@lan.ec-nantes.fr

Michael **Schenke**
Universität Oldenburg
FB 10 - TheoretischeInformatik
Ammerländer Heerstraše 114-118
D-26111 Oldenburg
Germany
email:
schenke@informatik.uni-oldenburg.de
tel.: +49-441-798-3133

R.K. **Shymasundar**
Tata Institute of Fundamental Research
Homi Bhabha Road
Bombay 400 005
India
email: shyam@turing.tifr.res.in
tel.: +91-22-215-29 71 X 2288

Karl-Heinz **Sylla**
GMD - Forschungszentrum
Informationstechnik GmbH
SET - EES
D - 53754 St. Augustin
Germany
email: karl-heinz.sylla@gmd.de
tel.: +49-2241-14 22 60

Shmuel **Tyszberowicz**
Tel AvivUniversity
Dept. of Computer Science
69978 Tel Aviv
Israel
email: tyshbe@math.tau.ac.il
tel.: +972-3-640-80 35

Andrew **Uselton**
SUNY at Stony Brook
Department of Computer Science
Stony Brook NY 11794-4400
USA
email: uselton@cs.sunysb.edu
tel.: +1-516-331-39 78

Guy **Vidal-Naquet**
Supelec
Plateau de Moulon
F-91192 Gif sur Yvette
France
email: vidalnaq@supelec.fr
tel.: +33-1-69.85.14.75

Michael **von der Beeck**
RWTH Aachen
Lehrstuhl für Informatik III
Ahornstr. 55
52074 Aachen
Germany
email:
beeck@i3.informatik.rwth-aachen.de
tel.: +49-241-80-2 13 16

Klaus **Winkelmann**
SIEMENS AG
Zentralabt. Forschung und Entwicklung
ZFE T SE1
Otto-Hahn-Ring 6
D-81730 München
Germany
email: klaus.winkelmann@zfe.siemens.de
tel.: +49-89-636-4 11 25