INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Espresso

Modélisation de systèmes réactifs polychrones

Rennes - Bretagne Atlantique

THEME COM

Table of contents

1. Team

Team Leader
Jean-Pierre Talpin [Research Director, INRIA, HdR]

Administrative Assistant
Céline Ammoniaux [Secretary, INRIA]

INRIA Personnel
Thierry Gautier [Research Associate, INRIA]
Paul Le Guernic [Research Director, INRIA]

CNRS Personnel
Loïc Besnard [Research Engineer, CNRS]

Post-Doctorate Members
Christian Brunette [Expert Engineer, INRIA]
Fabien Fillion [Expert Engineer, INRI]
Lionel Morel [Post-Doctorate, INRIA, until Sept. 1.]
Julio Peralta [Expert Engineer, INRIA, starting Apr. 1]
Eric Vecchie [Post-Doctorate, INRIA, starting. Dec. 15.]

Doctorate Students
Yann Glouche [INRIA, EADS Fundation]
Yue Ma [INRIA, starting Oct. 1.]
Hugo Métivier [University of Rennes]
Julien Ouy [INRIA, Regional Council of Britany]

2. Overall Objectives

2.1. Introduction

The Espresso project-team proposes models, methods and tools for computer-aided design of embedded systems. The model considered by the project-team is polychrony [12]. It is based on the paradigm of the synchronous hypothesis and allow for the specification of multi-clocked systems. The methods considered by the project-team put this model to work for the refinement-based (top-down) and component-based (bottom-up) design of embedded systems using correctness-preserving model transformations. The project-team makes a continuous effort to develop the Polychrony toolbox, freely available at http://www.irisa.fr/espresso/Polychrony.

Polychrony is an integrated development environment and technology demonstrator consisting of a compiler, a visual editor and a model checker. It provides a unified model-driven environment to perform embedded system design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony.

The company TNI-Valiosys supplies its commercial implementation, RT-Builder, used for industrial scale projects by Snecma/Hispano-Suiza and EADS – Airbus Industries (see http://www.tni-valiosys.com). Past and present collaborators of project-team Espresso through European, French and bilateral collaborations include CS-SI, CEA-List, MBDA, AONIX, SILICOMP, THALES, EDF, AIRBUS, VERIMAG, CEA.

2.2. Context and motivations

High-level embedded system design has gained prominence in the face of rising technological complexity, increasing performance requirements and shortening time to market demands for electronic equipments. Today, the installed base of intellectual property (IP) further stresses the requirements for adapting existing components with new services within complex integrated architectures, calling for appropriate mathematical models and methodological approaches to that purpose.

Over the past decade, numerous programming models, languages, tools and frameworks have been proposed to design, simulate and validate heterogeneous systems within abstract and rigorously defined mathematical models. Formal design frameworks provide well-defined mathematical models that yield a rigorous methodological support for the trusted design, automatic validation, and systematic test-case generation of systems.

However, they are usually not amenable to direct engineering use nor seem to satisfy the present industrial demand. As a matter of fact, the attention of the industry tends to shift to modeling frameworks based on general-purpose programming language variants, in response to a growing industry demand for higher abstraction-levels in the system design process and an attempt to fill the so-called *productivity gap*.

At present, a possibility of widening divergences between formal methods and industrial practices is perceivable. It seems that any useful methodology cannot avoid the industrial trend of using emerging programming languages. This contrasted picture calls for an effort toward the convergence between the theory of formal methods and the industrial practice and trends in system design.

Project-team Espresso aims at this convergence by considering the formal modeling framework of the Polychrony toolbox to serve as pivot formalism to import, transform, validate and export heterogeneous formalisms and languages.

2.3. The polychronous approach

Despite overwhelming advances in embedded systems design, existing techniques and tools merely provide *ad-hoc* solutions to the challenging issue of the productivity gap. The pressing demand for design tools has sometimes hidden the need to lay mathematical foundations below design languages. Many illustrating examples can be found, e.g. the variety of very different formal semantics found in state-diagram formalisms. Even though these design languages benefit from decades of programming practice, they still give rise to some diverging interpretations of their semantics.

The need for higher abstraction-levels and the rise of stronger market constraints now make the need for unambiguous design models more obvious. This challenge requires models and methods to translate a high-level system specification into a distribution of purely sequential programs and to implement semantics-preserving transformations and high-level optimizations such as hierarchization (sequentialization) or desynchronization (protocol synthesis).

In this aim, system design based on the so-called “synchronous hypothesis” has focused the attention of many academic and industrial actors. The synchronous paradigm consists of abstracting the non-functional implementation details of a system and lets one benefit from a focused reasoning on the logics behind the instants at which the system functionalities should be secured.

With this point of view, synchronous design models and languages provide intuitive models for embedded systems [4]. This affinity explains the ease of generating systems and architectures and verify their functionalities using compilers and related tools that implement this approach.

In the relational mathematical model behind the design language Signal, the supportive data-flow notation of Polychrony, this affinity goes beyond the domain of purely sequential systems and synchronous circuits and embraces the context of complex architectures consisting of synchronous circuits and desynchronization protocols: globally asynchronous and locally synchronous architectures (GALS).

This unique feature is obtained thanks to the fundamental notion of *polychrony*: the capability to describe systems in which components obey to multiple clock rates. It provides a mathematical foundation to a notion of *refinement*: the ability to model a system from the early stages of its requirement specifications (relations, properties) to the late stages of its synthesis and deployment (functions, automata).

The notion of polychrony goes beyond the usual scope of a programming language, allowing for specifications and properties to be described. As a result, the Signal design methodology draws a continuum from synchrony to asynchrony, from specification to implementation, from abstraction to refinement, from interface to implementation. Signal gives the opportunity to seamlessly model embedded systems at multiple levels of abstraction while reasoning within a simple and formally defined mathematical model.

The inherent flexibility of the abstract notion of signal handled in Signal invites and favors the design of correct-by-construction systems by means of well-defined model transformations that preserve the intended semantics and stated properties of the architecture under design.

3. Scientific Foundations

3.1. Scientific Foundations

Embedded systems are not new, but their pervasive introduction in ordinary-life objects (cars, telephone, home appliances) brought a new focus onto design methods for such systems. New development techniques are needed to meet the challenges of productivity in a competitive environment. Synchronous languages rely on the *synchronous hypothesis*, which lets computations and behaviors be divided into a discrete sequence of *computation steps* which are equivalently called *reactions* or *execution instants*. In itself this assumption is rather common in practical embedded system design.

But the synchronous hypothesis adds to this the fact that, *inside each instant*, the behavioral propagation is well-behaved (causal), so that the status of every signal or variable is established and defined prior to being tested or used. This criterion, which may be seen at first as an isolated technical requirement, is in fact the key point of the approach. It ensures strong semantic soundness by allowing universally recognized mathematical models to be used as supporting foundations. In turn, these models give access to a large corpus of efficient optimization, compilation, and formal verification techniques. The synchronous hypothesis also guarantees full equivalence between various levels of representation, thereby avoiding altogether the pitfalls of non-synthesizability of other similar formalisms. In that sense the synchronous hypothesis is, in our view, a major contribution to the goal of *model-based design* of embedded systems.

We shall describe the synchronous hypothesis and its mathematical background, together with a range of design techniques enpowered by the approach. Declarative formalisms implementing the synchronous hypothesis can be cast into a model of computation [12] consisting of a *domain* of traces or behaviors and of semi-lattice structure that renders the synchronous hypothesis using a timing equivalence relation: clock equivalence. Asynchrony can be superimposed on this model by considering a flow equivalence relation as well as heterogeneous systems [30] by parameterizing composition with arbitrary timing relations.

3.1.1. A synchronous model of computation

We consider a partially-ordered set of tags t to denote instants seen as symbolic periods in time during which a reaction takes place. The relation $t_1 \leq t_2$ says that t_1 occurs before t_2. Its minimum is noted 0. A totally ordered set of tags C is called a *chain* and denotes the sampling of a possibly continuous or dense signal over a countable series of causally related tags. Events, signals, behaviors and processes are defined as follows:

- an *event* e is a pair consisting of a value v and a tag t,
- a *signal* s is a function from a *chain* of tags to a set of values,
- a *behavior* b is a function from a set of names x to signals,
- a *process* p is a set of behaviors that have the same domain.

In the remainder, we write $\mathrm{tags}(s)$ for the tags of a signal s, $\mathrm{vars}(b)$ for the domains of b, $b|_X$ for the projection of a behavior b on a set of names X and b/X for its complementary.

Figure 1 depicts a behavior b over three signals named x, y and z. Two frames depict timing domains formalized by chains of tags. Signals x and y belong to the same timing domain: x is a down-sampling of y. Its events are synchronous to odd occurrences of events along y and share the same tags, e.g. t_1. Even tags of y, e.g. t_2, are ordered along its chain, e.g. $t_1 < t_2$, but absent from x. Signal z belongs to a different timing domain. Its tags, e.g. t_3 are not ordered with respect to the chain of y, e.g. $t_1 \neg \leq t_3$ and $t_3 \neg \leq t_1$.

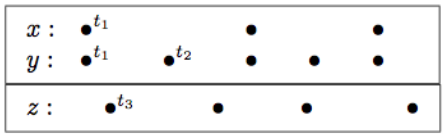


Figure 1. Behavior b over three signals x, y and z in two clock domains

3.1.1.1. Composition

Synchronous composition is noted $p \| q$ and defined by the union $b \cup c$ of all behaviors b (from p) and c (from q) which hold the same values at the same tags $b|_I = c|_I$ for all signal $x \in I = \text{vars}(b) \cap \text{vars}(c)$ they share. Figure 2 depicts the synchronous composition (Figure 2, right) of the behaviors b (Figure 2, left) and the behavior c (Figure 2, middle). The signal y, shared by b and c, carries the same tags and the same values in both b and c. Hence, $b \cup c$ defines the synchronous composition of b and c.

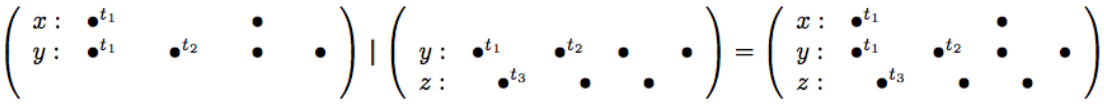


Figure 2. Synchronous composition of $b \in p$ and $c \in q$

3.1.1.2. Scheduling

A scheduling structure is defined to schedule the occurrence of events along signals during an instant t. A scheduling $\rightarrow$ is a pre-order relation between dates x_t where t represents the time and x the location of the event. Figure 3 depicts such a relation superimposed to the signals x and y of Figure 1. The relation $y_{t_1} \rightarrow x_{t_1}$, for instance, requires y to be calculated before x at the instant t_1. Naturally, scheduling is contained in time: if $t < t'$ then $x_t \rightarrow^b x_{t'}$ for any x and b and if $x_t \rightarrow^b x_{t'}$ then $t' \neg < t$.

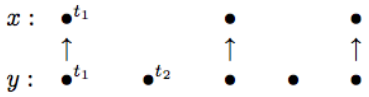


Figure 3. Scheduling relations between simultaneous events

3.1.1.3. Structure

A synchronous structure is defined by a semi-lattice structure to denote behaviors that have the same timing structure. The intuition behind this relation is depicted in Figure 4. It is to consider a signal as an elastic with ordered marks on it (tags). If the elastic is stretched, marks remain in the same relative (partial) order but have

more space (time) between each other. The same holds for a set of elastics: a behavior. If elastics are equally stretched, the order between marks is unchanged.

In Figure 4, the time scale of x and y changes but the partial timing and scheduling relations are preserved. Stretching is a partial-order relation which defines clock equivalence. Formally, a behavior c is a *stretching* of b of same domain, written $b \leq c$, iff there exists an increasing bijection on tags f that preserves the timing and scheduling relations. If so, c is the image of b by f. Last, the behaviors b and c are said *clock-equivalent*, written $b \sim c$, iff there exists a behavior d s.t. $d \leq b$ and $d \leq c$.

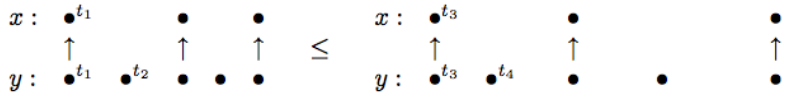


Figure 4. Relating synchronous behaviors by stretching.

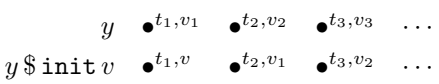
3.1.2. A declarative design languages

Signal [5] is a declarative design language expressed within the polychronous model of computation. In Signal, a process P is an infinite loop that consists of the synchronous composition $P \,|\, Q$ of simultaneous equations $x = y\ f\ z$ over signals named x, y, z. The restriction of a signal name x to a process P is noted P/x.

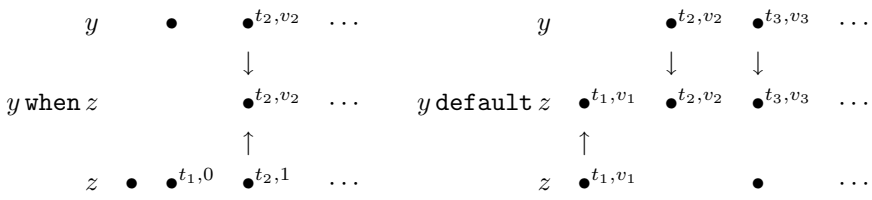
$$P, Q ::= x = y\ f\ z \mid P/x \mid P \,|\, Q$$

Equations $x = y\ f\ z$ in Signal more generally denote processes that define timing relations between input and output signals. There are four primitive combinators in Signal:

- delay $x = y \,\$\, \texttt{init}\ v$, initially defines the signal x by the value v and then by the previous value of the signal y. The signal y and its delayed copy $x = y \,\$\, \texttt{init}\ v$ are synchronous: they share the same set of tags $t_1, t_2, \cdots$. Initially, at t_1, the signal x takes the declared value v and then, at tag t_n, the value of y at tag t_{n-1}.



- sampling $x = y\ \texttt{when}\ z$, defines x by y when z is true (and both y and z are present); x is present with the value v_2 at t_2 only if y is present with v_2 at t_2 and if z is present at t_2 with the value true. When this is the case, one needs to schedule the calculation of y and z before x, as depicted by $y_{t_2} \longrightarrow x_{t_2} \longleftarrow z_{t_2}$.
- merge $x = y\ \texttt{default}\ z$, defines x by y when y is present and by z otherwise. If y is absent and z present with v_1 at t_1 then x holds (t_1, v_1). If y is present (at t_2 or t_3) then x holds its value whether z is present (at t_2) or not (at t_3).



The structuring element of a Signal specification is a process. A process accepts input signals originating from possibly different clock domains to produce output signals when needed. This allows, for instance, to specify a counter where the inputs `tick` and `reset` and the output `value` have independent clocks. The body of `counter` consists of one equation that defines the output signal `value`. Upon the event `reset`, it sets the count to 0. Otherwise, upon a `tick` event, it increments the count by referring to the previous value of `value` and adding 1 to it. Otherwise, if the count is solicited in the context of the counter process (meaning that its clock is active), the counter just returns the previous count without having to obtain a value from the `tick` and `reset` signals.

```
process counter = (? event tick, reset ! integer value)
    (| value := (0 when reset)
       default ((value$ init 0 + 1) when tick)
       default (value$ init 0)
    |);
```

A Signal process is a structuring element akin to a hierarchical block diagram. A process may structurally contain sub-processes. A process is a generic structuring element that can be specialized to the timing context of its call. For instance, the definition of a synchronized counter starting from the previous specification consists of its refinement with synchronization. The input tick and reset clocks expected by the process `counter` are sampled from the boolean input signals `tick` and `reset` by using the `when tick` and `when reset` expressions. The count is then synchronized to the inputs by the equation `reset ^= tick ^= count`.

```
process synccounter = (? boolean tick, reset ! integer value)
    (| value := counter (when tick, when reset)
     | reset ^= tick ^= value
     |);
```

3.1.3. Compilation of Signal

Sequential code generation starting from a Signal specification starts with an analysis of its implicit synchronization and scheduling relations. This analysis yields the control and data flow graphs that define the class of sequentially executable specifications and allow to generate code.

3.1.3.1. Synchronization and scheduling specifications

In Signal, the clock $\hat{}x$ of a signal x denotes the set of instants at which the signal x is present. It is represented by a signal that is true when x is present and that is absent otherwise. Clock expressions represent control. The clock $\texttt{when}\, x$ (resp. $\texttt{not}\, x$) represents the time tags at which a boolean signal x is present and true (resp. false).

The empty clock is written 0 and clocks expressions e combined using conjunction, disjunction and symmetric difference. Clocks equations E are Signal processes: the equation $e\hat{} = e'$ synchronizes the clocks e and e' while $e\hat{} < e'$ specifies the containment of e in e'. Explicit scheduling relations $x \rightarrow y\, \texttt{when}\, e$ allow to schedule the calculation of signals (e.g. x after y at the clock e).

$$
\begin{array}{rcll}
e & ::= & \hat{}x \mid \texttt{when}\, x \mid \texttt{not}\, x \mid e\hat{} + e' \mid e\hat{} - e' \mid e\hat{} * e' \mid 0 & \text{(clock expression)} \\
E & ::= & () \mid e\hat{} = e' \mid e\hat{} < e' \mid x \rightarrow y\, \texttt{when}\, e \mid E \,|\, E' \mid E/x & \text{(clock relations)}
\end{array}
$$

3.1.3.2. Synchronization and scheduling analysis

A Signal process P corresponds to a system of clock and scheduling relations E that denotes its timing structure. It can be defined by induction on the structure of P using the inference system $P : E$ of Figure 5.

```
x := y$ init v   : ^x ^= ^y
x := y when z    : ^x ^= ^y when z | y -> x when z
x := y default z : ^x ^= ^y default ^z | y -> x when ^y | z -> x when ^z ^- ^y
```

Figure 5. Clock inference system

3.1.3.3. Hierarchization

The clock and scheduling relations E of a process P define the control-flow and data-flow graphs that hold all necessary information to compile a Signal specification upon satisfaction of the property of *endochrony*. A process is said endochronous iff, given a set of input signals and flow-equivalent input behaviors, it has the capability to reconstruct a unique synchronous behavior up to clock-equivalence: the input and output signals are ordered in clock-equivalent ways.

To determine the order $x \preceq y$ in which signals are processed during the period of a reaction, clock relations E play an essential role. The process of determining this order is called hierarchization and consists of an insertion algorithm which hooks elementary control flow graphs (in the form of if-then-else structures) one to the others. Figure 6, right, let h3 be a clock computed using h1 and h2. Let h be the head of a tree from which h1 and h2 are computed (an if-then-else), h3 is computed after h1 and h2 and placed under h.

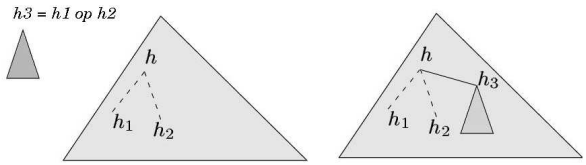


Figure 6. Hierarchization of clocks

3.1.3.4. Example

The implications of hierarchization for code generation can be outlined by considering the specification of a one-place buffer in Signal. Process `buffer` implements two functionalities. One is the process `alternate` which desynchronizes the signals `i` and `o` by synchronizing them to the true and false values of an alternating boolean signal `b`.

```
process buffer = (? i ! o)
  (| alternate (i, o)
   | o := current (i)
   |)
where
    process alternate = (? i, o ! )
      (| zb := b$1 init true
       | b := not zb
       | o ^= when not b
```

```
        | i ^= when b
        |) / b, zb;
    process current = (? i ! o)
      (| zo := i cell ^o init false
       | o  := zo when ^o
       |) / zo;
end;
```

The other functionality is the process `current`. It defines a `cell` in which values are stored at the input clock `^i` and loaded at the output clock `^o`. `cell` is a predefined Signal operation defined by:

$$x := y \texttt{ cell } z \texttt{ init } v =^{def} (m := x \$ \texttt{init}\, v \,|\, x := y \texttt{ default } m \,|\, \hat{x} = \hat{y} + \hat{z})\,/m$$

Clock inference applies the clock inference system of Figure 5 to the process `buffer` to determine three synchronization classes. We observe that `b, c_b, zb, zo` are synchronous and define the master clock synchronization class of `buffer`.

```
(| c_b ^= b
 | b   ^= zb
 | zb  ^= zo
 | c_i := when b
 | c_i ^= i
 | c_o := when not b
 | c_o ^= o
 | i -> zo when ^i
 | zb -> b
 | zo -> o when ^o
 |) / zb, zo, c_b, c_o, c_i, b;
```

There are two other synchronization classes, `c_i` and `c_o`, that corresponds to the true and false values of the boolean flip-flop variable `b`, respectively:

$$\texttt{b} \multimap \texttt{c\_b} \multimap \texttt{zb} \multimap \texttt{zo} \text{ and } \texttt{b} \preceq \texttt{c\_i} \multimap \texttt{i} \text{ and } \texttt{b} \preceq \texttt{c\_o} \multimap \texttt{o}$$

This defines three nodes in the control-flow graph of the generated code. At the main clock `c_b`, `b` and `c_o` are calculated from `zb`. At the sub-clock `b`, the input signal `i` is read. At the sub-clock `c_o` the output signal `o` is written. Finally, `zb` is determined. Notice that the sequence of instructions follows the scheduling relations determined during clock inference.

```
buffer_iterate () {
    b = !zb;
    c_o = !b;
    if (b) {
      if (!r_buffer_i(&i))
        return FALSE;
    };
    if (c_o) {
```

```
        o = i;
        w_buffer_o(o);
      };
      zb = b;
      return TRUE;
}
```

Whereas Signal uses a hierarchization algorithm to find a sequential execution path starting from a system of clock relations, Lustre leaves this task to engineers, which must provide a well-synchronized program: well-synchronized Lustre programs correspond to hierarchized Signal specifications.

3.1.3.5. Certification

The simplicity of the single-clocked model of Lustre eases program analysis and code generation and its commercial implementation, Scade by Esterel Technologies, provides a certified C code generator. Its combination to Sildex, the commercial implementation of Signal by TNI-Valiosys, as a front-end for architecture mapping and early requirement specification is the methodology advocated in the IST project Safeair (URL: http://www.safeair.org). The formal validation and certification of synchronous program properties has been the subject of numerous studies. In [40], a co-inductive axiomatization of Signal in the proof assistant Coq [36], based on the calculus of constructions [49], is proposed.

The application of this model is two-folds. It allows, first of all, for the exhaustive verification of formal properties of infinite-state systems. Two case studies have been developed. In [37], a faithful model of the steam-boiler problem was given in Signal and its properties proved with Signal's Coq model. It is applied to proving the correctness of real-time properties of a protocol for loosely time-triggered architectures, extending previous work proving the correctness of its finite-state approximation [38].

Another and important application of modeling Signal in the proof assistant Coq is being explored and consists of developing a reference compiler translating Signal programs into Coq assertion. This translation allows to represent model transformations performed by the Signal compiler as correctness preserving transformations of Coq assertions, yielding a costly yet correct-by-construction synthesis of target code.

Other approaches to the certification of generated code have been investigated. In [42], validation is achieved by checking a model of the C code generated by the Signal compiler in the theorem prover PVS with respect to a model of its source specification: translation validation.

4. Application Domains

4.1. Application Domains

The application domains covered by the Polychrony toolbox are engineering areas where a system design-flow requires high-level model transformations and verifications to be applied during the development-cycle.

The project-team has focused on developing such integrated design methods in the context of avionics applications, through the European IST projects Sacres, Syrf, Safeair. This research track is being continued in the submitted Espace (*avionics*) and Sea (*automotive*) projects.

In this context, Polychrony is seen as a platform on which the architecture of an embedded system can be specified from the earliest design stages until the late deployment stages through a number of formally verifiable design refinements.

Recent trends in *system-level design* show, in a far from unrelated way, the need for modeling systems on chips as globally asynchronous and locally synchronous systems. It is indeed manifest in the charter of the ACM-IEEE MEMOCODE conference. It is the subject of an ongoing collaboration of project-team Espresso with UC San Diego and Virginia Tech through INRIA associate-projects program.

5. Software

5.1. The Polychrony workbench

Participants: Loic Besnard, Thierry Gautier, Paul Le Guernic.

Polychrony is an integrated development environment and technology demonstrator consisting of a compiler, of a visual editor and of a model checker. It provides a unified model-driven environment to perform embedded system design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony.

Polychrony supports the synchronous, multi-clocked, data-flow specification language Signal. It is being extended by plugins to capture SystemC modules or real-time Java classes within the workbench. It allows to perform validation and verification tasks, e.g., with the integrated SIGALI model checker, the Coq theorem prover, or with the Spin model checker.

Polychrony is registered at the APP and is freely distributed from http://www.irisa.fr/espresso/Polychrony for non-commercial use. Based on the Signal language, it provides a formal framework:

1. to validate a design at different levels,
2. to refine descriptions in a top-down approach,
3. to abstract properties needed for black-box composition,
4. to assemble predefined components (bottom-up with COTS).

The company TNI-Valiosys supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects by Snecma/Hispano-Suiza and Airbus Industries (see http://www.tni-valiosys.com).

Polychrony is a set of tools composed of:

1. A Signal batch compiler providing a set of functionalities viewed as a set of services for, e.g., program transformations, optimizations, formal verification, abstraction, separate compilation, mapping, code generation, simulation, temporal profiling, etc.
2. A GUI with interactive access to compiling functionalities.
3. The SIGALI tool, an associated formal system for formal verification and controller synthesis, jointly developed with the Vertecs project-team (http://www.irisa.fr/vertecs).

Polychrony offers services for modeling application programs and architectures starting from high-level and heterogeneous input notations and formalisms. These models are imported in Polychrony using the data-flow notation Signal. Polychrony operates these models by performing global transformations and optimizations on them (hierarchization of control, desynchronization protocol synthesis, separate compilation, clustering, abstraction) in order to deploy them on mission specific target architectures. C, C++, multi-threaded and real-time Java and SynDex code generators are provided. The connection to the SynDEx distribution tool (http://www-rocq.inria.fr/syndex) has been developed in the context of the RNTL project Acotris.

5.2. Integrated Modular Avionics design using Polychrony

Participants: Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.

The Apex interface, defined in the ARINC standard [27], provides an avionics application software with the set of basic services to access the operating-system and other system-specific resources. Its definition relies on the Integrated Modular Avionics approach (IMA, [28]). A main feature in an IMA architecture is that several avionics applications (possibly with different critical levels) can be hosted on a single, shared computer system. Of course, a critical issue is to ensure safe allocation of shared computer resources in order to prevent fault propagations from one hosted application to another. This is addressed through a functional partitioning of the applications with respect to available time and memory resources. The allocation unit that results from this decomposition is the *partition*.

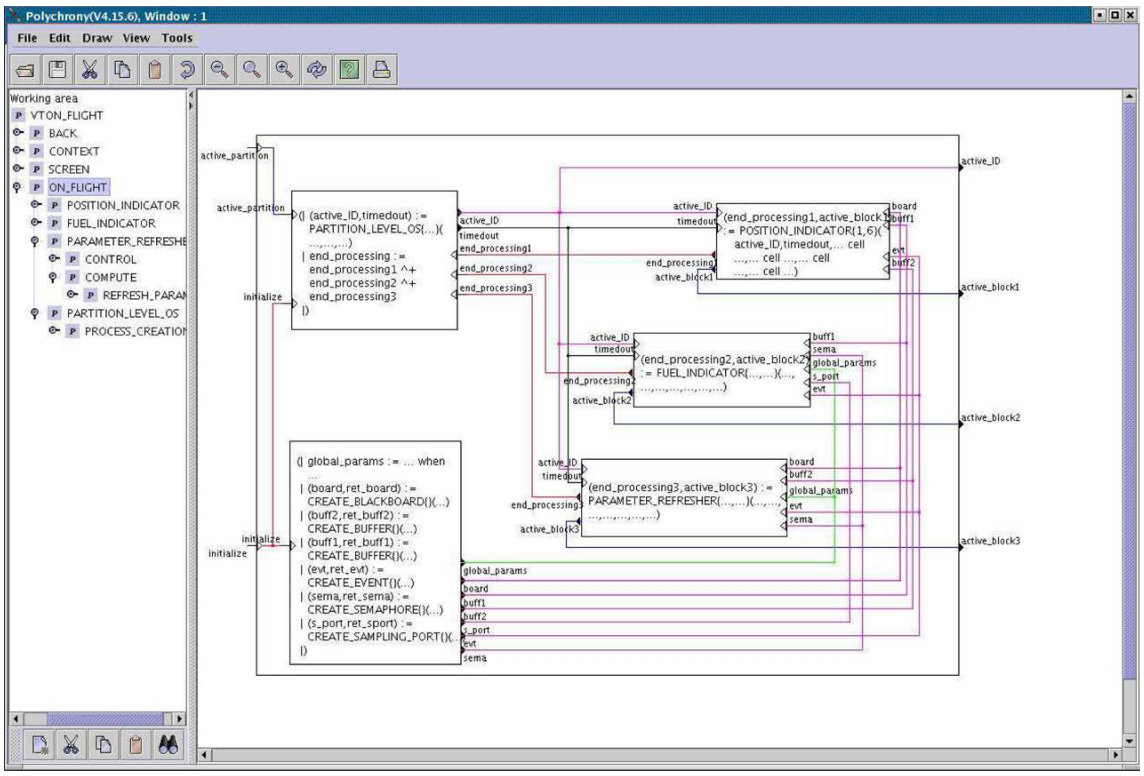


Figure 7. Avionics application modeling using the visual editor of the Polychrony workbench

A partition is composed of *processes* which represent the executive units (an ARINC partition/process is akin to a Unix process/task). When a partition is activated, its owned processes run concurrently to perform the functions associated with the partition. The process scheduling policy is priority preemptive.

Each partition is allocated to a processor for a fixed time window within a major time frame maintained by the operating system. Suitable mechanisms and devices are provided for communication and synchronization between processes (e.g. *buffer*, *event*, *semaphore*) and partitions (e.g. *ports* and *channels*).

The specification of the ARINC 651-653 services in Signal [6] is now part of the distribution Polychrony and offers a complete implementation of the Apex communication, synchronization, process management and partitioning services. Its Signal implementation consists of a library of generic, parameterizable Signal modules.

5.3. A model of Signal in Coq

Participant: Jean-Pierre Talpin.

The verification of a reactive system is usually done by elaborating a *discrete* model of the system specified in a dedicated formalism and then by checking a property against the model. The use of formal proof systems enables to prove *hybrid properties* about *infinite state systems*: the *correctness* and the *completeness* of a reactive system. To this aim, the Espresso project-team has developed a complete model of the Signal design language in Coq [40]. More precisely, we have defined a translation scheme of the trace semantics of Signal to the logical framework of Coq. We have conducted several case studies to demonstrate the applicability of the approach to resolve sophisticated verification problems: a complete model and proof of the well-known steam-boiler problem [37], the correctness of an implementation of a Signal protocol for loosely timed-triggered architectures [38]. Such a proof, of course, cannot always be done automatically: it requires human-interaction to direct the proof strategy. The prover can nonetheless automate its most tedious and mechanical parts. In general, formal proofs of programs are difficult and time-consuming. In the particular case of modeling a reactive system using Signal, experience however shows that this difficulty is significantly reduced thanks to the combined declarative style of programming and a relational style of modeling.

5.4. An Eclipse plugin for Polychrony

Participants: Christian Brunette, Loic Besnard, Jean-Pierre Talpin.

We have developed a metamodel and interactive editor of Polychrony in Eclipse. Signal-Meta is the metamodel of the SIGNAL language. It describes all syntactic elements specified in [32]: all SIGNAL operators (e.g. arithmetic, clock synchronization), model (e.g. process frame, module), and construction (e.g. iteration, type declaration).

Signal-Meta has been extended to allow the definition of mode automata, which were originally proposed by Maraninchi et al. [39] to extend the functionality-oriented data-flow paradigm with the capability to model transition systems easily and provide an additional imperative flavor.

MIMAD is also built as an extension of Signal-Meta and allows to design applications based on the *Integrated Modular Avionics* (IMA) architecture, which relies on the avionic standard APEX-ARINC [27], [28].

These metamodels aims at providing a user with a graphical framework allowing to model applications using a component-based approach. Application architectures can be easily described by just selecting these components via drag and drop, creating some connections between them and specifying their parameters as component attribute. To complete this framework, we have developed, for each of these metamodels, GME interpreters to transform the resulting graphical model to SIGNAL programs, and so to test and compile them in Polychrony.

Using the modeling facilities provided with the Topcased framework, we create a graphical environment for Polychrony (see figure 8).

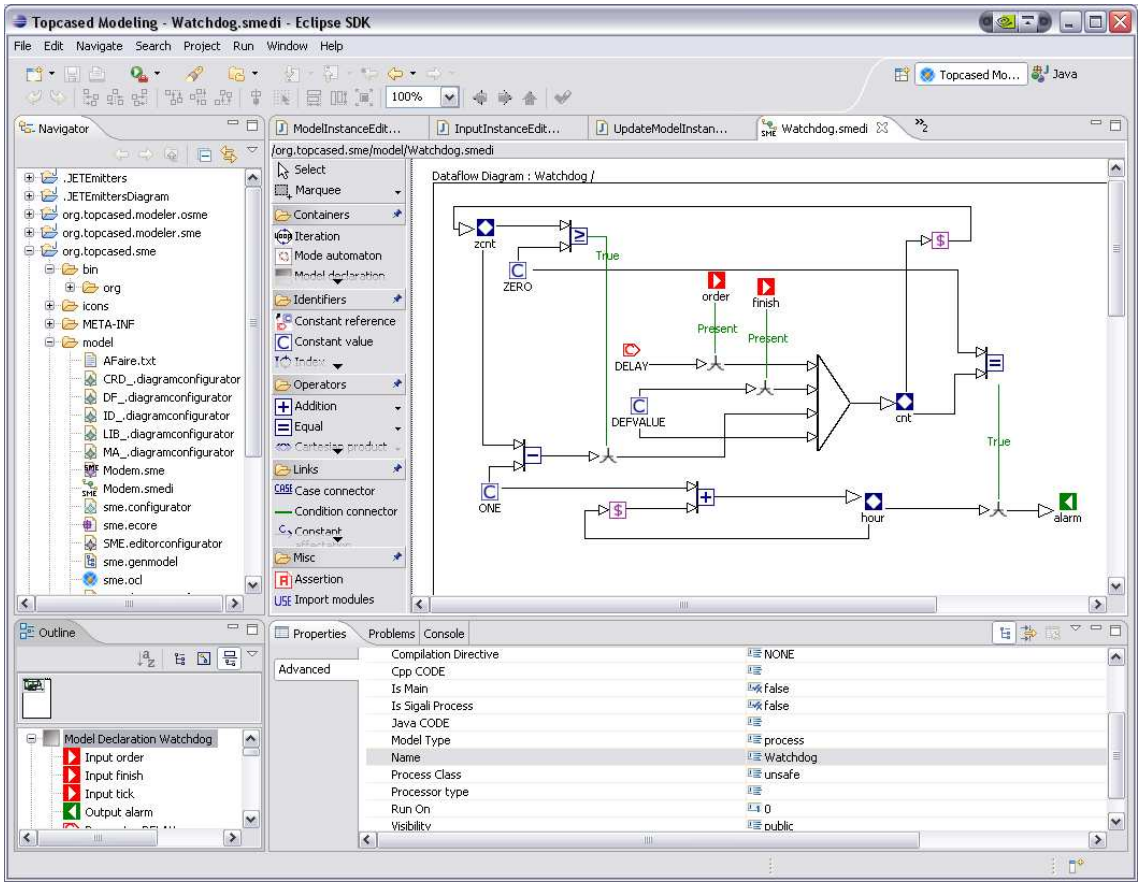


Figure 8. Signal-Meta environment in Eclipse.

The environment is complete concerning the modeling part. We reproduce in Eclipse the GME notion of *Aspect*. So, the modeling of a SIGNAL process is split in three diagrams: one to model the interface of the process, one to model the computation part, and one to model all explicit clock relations and dependences. Actually, the graphical environment under Eclipse cannot generate Signal code the way we did it in GME. Our objective in Eclipse is to deeply connect the graphical environment with the Polychrony compiler to dynamically check the correctness of the model. Through this connection to the compiler, we can use the facilities of the compiler to generate the SIGNAL program.

5.5. Multi-clocked mode automata

Keywords: *Generic Modeling Environment*, *Mode automata*, *Model transformation*, *SIGNAL*.

Participants: Jean-Pierre Talpin, Thierry Gautier, Christian Brunette.

Gathering advantages of declarative and imperative approaches, mode automata were originally proposed by Maraninchi et al. [39] to extend the functionality-oriented data-flow paradigm with the capability to model transition systems easily and provide an additional imperative flavor. Similar variants and extensions of the same approach to mix multiple programming paradigms or heterogeneous models of computation [33] have been proposed until recently, the latest advance being the combination of stream functions with automata in [34]. Nowadays, commercial toolsets such as the Esterel Studio's Scade or Matlab/Simulink's Stateflow are largely inspired from similar concepts.

While the introduction of preemption mechanism in the multi-clocked data-flow formalism Signal was previously studied by Rutten et al. in [43], no attempt has been made to extend mode automata with the capability to model multi-clocked systems and multi-rate systems. In [46], we extend Signal-Meta with an inherited metamodel of multi-clocked mode automata. A salient feature is the simplicity incurred by the separation of concerns between data-flow (that expresses structure) and control-flow (that expresses a timing model) that is characteristic to the design methodology of SIGNAL.

While the specification of mode automata in related works requires a primary address on the semantics and on compilation of control, the use of SIGNAL as a foundation allows to waive this specific issue to its analysis and code generation engine Polychrony and clearly expose the semantics and transformation of mode automata in a much simpler way by making use of clearly separated concerns expressed by guarded commands (data-flow relations) and by clock equations (control-flow relations).

6. New Results

6.1. New features of Polychrony

Participants: Loïc Besnard, Thierry Gautier.

This year, we have finalized the *open-source* release of the environment: all the data structures, classes, methods are documented. The associated documentation is automatically generated using "doxygen" tool. Polychrony will be an open-source software licenced under the Cecill-B licence (URL http://www.cecill.info/licences/Licence_CeCILL-B_V1-en.html). This work has been performed in parallel with the packaging of the data structures of Polychrony as libraries. These libraries are now used for the definition of model transformations described in high-level IDE tools (UML-based, GME "Model-Integrated Computing"...) and the connection with the synthesis tools defined in the R2D2 team.

Moreover, this year we have implemented a new algorithm for the computation of the exclusive clocks of a program. This new algorithm takes into account the inclusion properties induced by the clock hierarchy:

- For two exclusive clocks h_1, h_2, all the clocks of the sub-tree with h_1 as root and of the sub-tree with h_2 as root are exclusive
- For two non exclusive clocks h_1, h_2, all the clocks of the sub-tree with h_1 as root and of the sub-tree with h_2 as root are non exclusive

The application of these properties have allowed to reduce the computation time (divided by 10 for some applications) of the exclusive clocks of a program.

We have also added this year a new code generation service in Polychrony. Until this year, Polychrony allowed several cases of code generation for a sequential simulation code

- Mono-block code for autonomous execution
- Code partitioned in clusters, with static scheduling of the clusters, for autonomous execution.
- Mono-block code for separate compilation
- Clusters with static scheduling, for separate compilation

We have added a code generation with *threads*, based on the partitioning of the graph in *clusters*:

- Clusters are defined such that each flow of a given cluster depends on the same set of input flows
- A cluster can be executed atomically as soon as its inputs are available
- There is a specific cluster for state variables updating (called "Cluster delays")

This partitioning is used for a multi-thread simulation code, with a *dynamical* scheduling:

- One task per cluster + one task per input/output + one task that manages the steps
- The code of clusters is left unchanged, but synchronizations are added
- One semaphore Sem_i per task T_i
- At the beginning of each iteration step of a task T_i: as many `wait(`Sem_i`)` as T_i has predecessors
- At the end of each iteration step of a task T_i: one `signal(`Sem_k`)` for each T_k successor of T_i
- The iteration step is managed by a specific task that executes a `signal()` on the semaphores associated with the tasks that have no predecessor, and then, a `wait()` on its own semaphore (a `signal()` on this latter is executed at the end of the task corresponding to the "Cluster delays")

This new code generation will be the basis of the distibuted code generation currently developed.

6.2. New features of the Eclipse plug-ins

Keywords: *Eclipse*, *Ecore*, *Meta-modeling*, *Model transformation*, *SIGNAL*.

Participants: Loïc Besnard, Christian Brunette, Jean-Pierre Talpin.

This year, the focus has been put on the connection between the plug-ins and the Polychrony compiler. We develop a Java/C interface to communicate with the compiler through native libraries (for Linux and Windows). The principle of the communication (represented on figure 9) is the following:

- transform the model conform to the SME meta-model to the abstract syntax tree (AST) representation inside the compiler,
- transform this AST representation to a internal graph structure,
- apply the different Polychrony services to this graph (Clock resolution, code generation,...),
- report the errors of the compiler provided by the compiler to the graphical part (not yet implemented).

The current Polychrony Eclipse suite is composed of:

- a reflexive editor to describe and visialize an SME model as a tree,
- a graphical modeler using the TopCased facilities (v1.0.0) to describe SME models as different kind of hierarchical diagrams,
- a reflexive editor to describe and visialize compilation scenario models as a tree,
- an interactive view to help the user to describe compilation scenarios as the interactive compilation mode in the classical Polychrony graphical interface,
- some basic SME and compilation scenario model examples,
- an Eclipse user-documentation, which describes how to use these plug-ins.

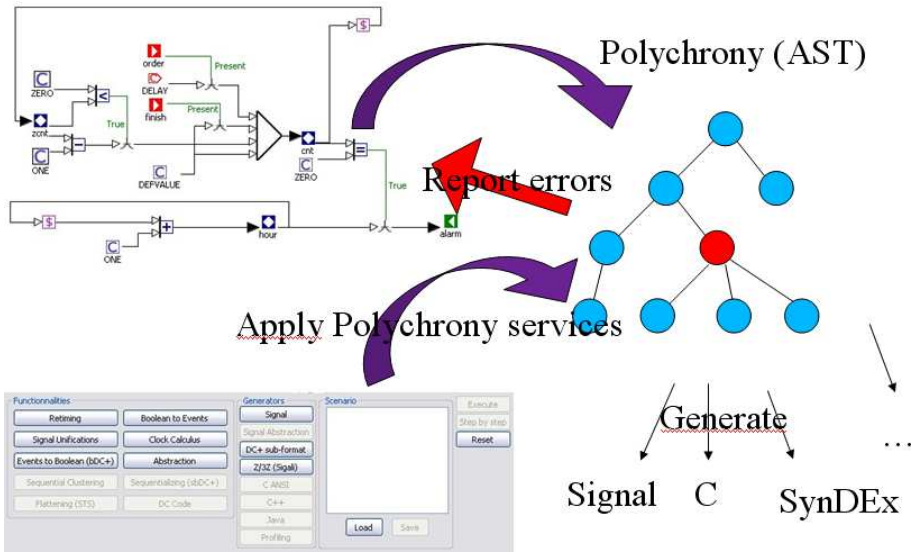


Figure 9. Interaction with Polychrony.

These plug-ins are included in the current experimental OpenEmbeDD platform, and will normally be added before the end of this year to the TopCased distribution.

To reuse easily all programs and libraries done in Signal, we work on transformations of Signal textual (.SIG) files to SME model files. For this purpose, we study the feasability of this transformation using TCS (INRIA Atlas) [26], or Sintaks (INRIA Triskell) [47]. In the same time, we began the development of a XMI (conform to the SME meta-model) generator inside the Polychrony toolbox.

6.3. Periodic clock relations

Participants: Thierry Gautier, Paul Le Guernic, Hugo Metivier, Jean-Pierre Talpin.

We try to extend the clock calculus of Signal with periodic timing relations between signals. These relations can be expressed using infinite periodic binary words and yield a more expressive clock calculus (than that based on symbolic clock relations). Our extension of the clock calculus aims at providing the capability to model timed systems.

Periodic relations are expressed between pairs of signals. A first analysis infers periodic synchronization relations from the equations of a program and proceeds in an iterative manner much like an abstract interpretation until a solution is found.

We are working on such a non-assisted method to infer the periodic clock relations from Signal equations. This method would permit to extend the set of programs accepted by the Signal compiler thanks to an automatic insertion of bounded buffer for the N-synchronizable signals. In future work, the periodic clock calculus could be use to analyse and optimize scheduling in the code generated from a Signal program.

The periodic systems often need buffers to delay data, and need ressource of computation to perform cyclic calculus. A second analyse of the periodic processes permit us to insert buffer to guarantee the communication, and to guarantee the computation for each cycle of the system. This method would permit to extend the set of programs accepted by the Signal compiler thanks to an automatic insertion of bounded buffer for the N-synchronizable signals.

We work on the application of this method to model and analyse periodic systems like embedded systems in satellites, video converters and 4-stroke engines.

6.4. Compositional design of isochronous systems

Participants: Loïc Besnard, Paul Le Guernic, Julien Ouy, Jean-Pierre Talpin.

We are continuing our effort initiated in [45] to provide a compositional implementation of isochronous systems in Polychrony. We have defined a class of reactive and deterministic Signal specifications that can be separately compiled and concurrently executed, allowing one to use Signal for simulating globally asynchronous locally synchronous architectures.

As code generation for synchronous programs requires strong safety properties to be satisfied, compositionality becomes a difficult goal to achieve. Most synchronous languages, such as Esterel, Lustre or Signal require a given module or compilation unit to be insensitive to latency that communication with its envi- ronment may incur. In Lustre or Signal, for instance, a compilation unit must satisfy the so-called property of endochrony. To preserve endochrony in an asynchronous environment, an ad-hoc protocol is synthesized to interface the module. However, endochrony is not preserved by composition. Consequently, the proto- col has to be rebuilt every time a new module is added in the environment. We propose a methodology and code generation scheme which simplifies this concern. It consists of weakening the global objective of globally preserving endochrony. Instead, we aim at the preservation of a more liberal and compositional objective, weak endochrony, which is compositional and much closer from the expected requirement of insensitivity to communication latency. As a result, our code generation scheme supports true separate compilation: a locally compiled synchronous module does not require its synthesized interface with the environment to be rebuilt once composed with another module.

Those results have been published in [22], [25].

6.5. A contract-based module system

Participants: Yann Glouche, Thierry Gautier, Lionel Morel, Jean-Pierre Talpin.

Contract-based systems, based on the assume-guarantee paradigm, have become a popular formalism for the modular specification of object-oriented programs. In the context of the development of embedded systems, functional (behavioral) contracts are being applied and become part of mainstream industrial tool.

Our goal is to exploit contracts during the developement phase, for instance as a support for early execution (simulation). In this context, we have introduce a new paradigm to express the essence of encapsulation and inheritance in a synchronous and a concurrent modeling framework.

Reusability of components is achieved by an encapsulation mechanism for which the type of an object is caracterized by a contract and the type of this input/output. A static interpretation of inheritance allows a verification of the contracts satisfaction associated with the components.

We are currently working on the defintion of a denotational semantics of the contract language, to give a precise definition of a process P and, more interestingly, of its complementary $\neg P$: if P is a discrete-event system, then $\neg P$ is obviously continuous (or dense). Then, it is necessary to define some sort of abstraction or sampling to prove the conformance a (discret) process to a (continuous) contract.

6.6. Verification of GALS architectures

Participants: Julio Peralta, Jean-Pierre Talpin.

In the context of the Topcased project, we focus on defining a translation schema to pass from Signal specifications into Fiacre [31] specifications. Fiacre, in turn, is one of the languages (currently under development) for describing concurrent models input to the CADP [35] model checker (developed at INRIA Rhone-Alpes), or the TINA [29] toolset.

As a first step we explored some tradeoffs in the space of translation possibilities through hand-made translations of a Synchronous Dual-Flight Guidance System [23]. Our first experiments suggest that in order to reduce the complexity of the generated Fiacre programs we ought to proceed in tight cooperation with the Signal compiler, so as to have access to scheduling and code generation information. Also, GALS verification will benefit from advances in desynchronization of Signal specifications, advocated in other goals of the Espresso project-team (subsection 6.3).

Our future work considers using model transformation languages and tools (ATL or Kermeta) in order to describe the desired translation between the Signal metamodel and a Fiacre Metamodel. Such translation will be exercised using avionics application examples.

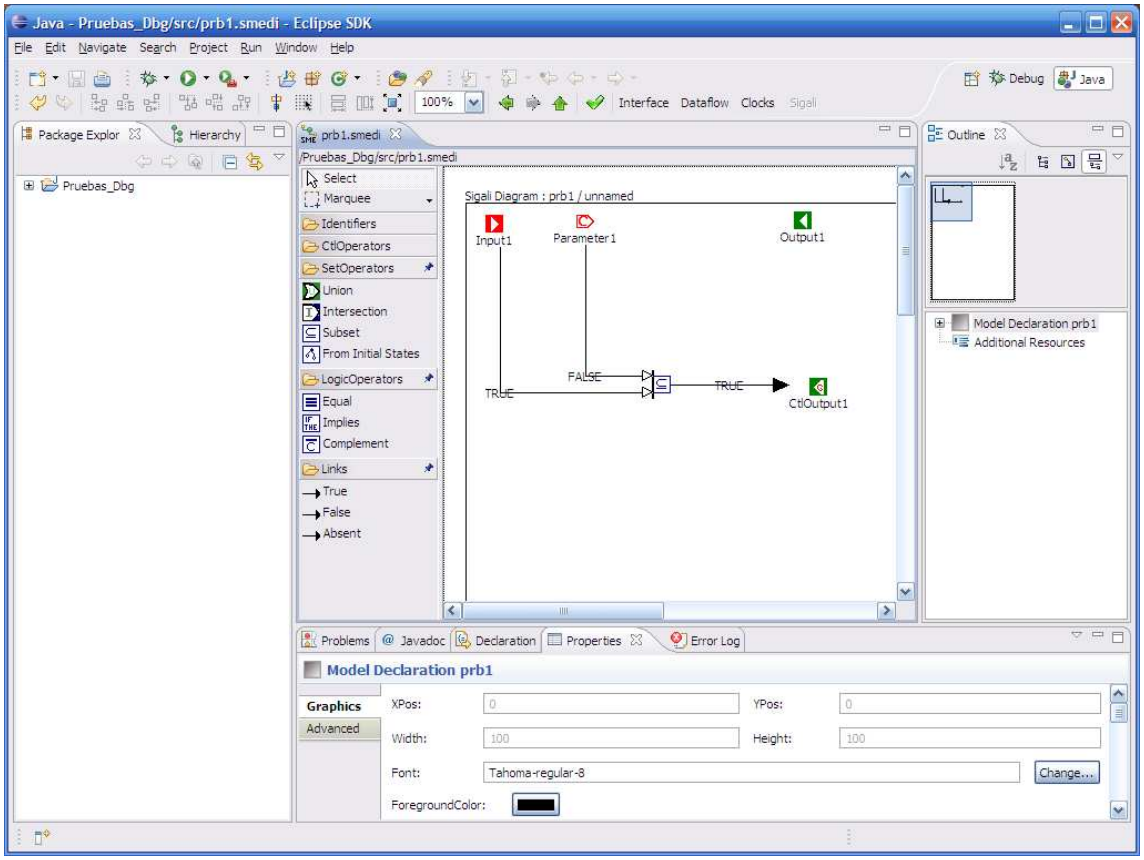


Figure 10. Sigali Aspect inside Signal-Meta environment.

One other development in this branch is the addition of a verification *aspect* (for the Sigali model checker, Fig. 10) to the existing Signal graphical editor (see subsection 5.4). At the moment we are testing and debugging a first version of this graphical editor, and in the near future we expect to connect the graphical editor with the actual tool (Sigali model checker), as well as to allow controller synthesis (currently unavailable from this graphical editor aspect).

6.7. Virtual prototyping of avionic architecture descriptions

Participants: Yue Ma, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.

In the context of the Topcased project, leaded by Airbus, we are designing and developping a tool for the virtual prottyping of avionic architecture specifications. Our aim is to interprete AADL specifications in the synchronous model of computation of Polychrony in order to provide a framework for the simulation, test and verification of integrated modular avionics. Our goal is to validate our by the case study of an autonomous robot.

7. Contracts and Grants with Industry

7.1. IST project Speeds

Participants: Lionel Morel, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.

The Espresso project-team is participating to the IST project SPEEDS (Speculative and Exploratory Design in Systems Engineering, http://www.speeds.eu.com/).

SPEEDS is a concerted effort to define the new generation of end-to-end methodologies, processes and supporting tools for safety-critical embedded system design. It aims at improving substantially the competitiveness of the European industry in this critical economic sector by marrying design competence with deep technical insights and theoretical foundations.

7.2. Network of excellence Artist2

Participants: Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin, Eric Vecchie.

The Espresso project-team participates to the Artist2 network of excellence. Detailed presentations on the aim and scope of the network can be found in the book [1] and the website http://www.artist-embedded.org/FP6 of the project.

7.3. RNTL project OpenEmbeDD

Participants: Christian Brunette, Fabien Fillion, Jean-Pierre Talpin.

The Espresso project-team coordinates (in collaboration with project Tryskel) the RNTL project OpenEmbeDD. The goal of OpenEmbeDD is to develop an open-source toolset consisting of:

- Model-driven design infrastructures
- Asynchronous design and verification environments
- Synchronous design and verification environments

for the design of embedded software. The project comprises many industrial partners to carry out case studies and validate the design environment. In the frame of OpenEmbeDD, project Espresso develops an Eclipse plugin for Polychrony in collaboration with Airbus (Topcased technology) and project Atlas (ATL technology).

7.4. ANR project Topcased

Participants: Paul Le Guernic, Julio Peralta, Yue Ma, Jean-Pierre Talpin.

The Espresso project-team participates to the Topcased initiative of Airbus. The aim of the Topcased initiative is to developp an open-source toolset for the design of avionic architectures. A summary of Topcased appears in [48]. Project Topcased is funded by the ANR and the Midi-Pyréné region.

7.5. RNTL project Spacify

Participants: Loic Besnard, Julien Ouy, Jean-Pierre Talpin.

The Espresso project-team participates to the RNTL project Spacify, leaded by CNES. The aim of the Spacify is to developp an open-source design environment devoted to engineering embedded software for spatial applications.

7.6. ANR project FotoVP

Participant: Jean-Pierre Talpin.

The Espresso project-team participates to the ANR project FotoVP, leaded by Verimag. The aim of the FotoVP is to developp virtual prototyping tools and techniques for the simulation and verification of system-level C/SystemC specifications in synchronous models of computation.

7.7. Fondation EADS Grant

Participants: Yann Glouche, Jean-Pierre Talpin.

The Espresso project-team received a grant from the EADS Foundation to fund a Doctorate on contract-based design in a polychronous model of computation. The aim of this program, carried in collaboration with case studies from MBDA, is to develop and model-driven engineering framework, based on the Eclipse plugin for Polychrony (developped in the frame of 7.3, allowing for the seamless integration of heterogeneous embedded system components within a contract-based design MDD environment.

8. Other Grants and Activities

8.1. INRIA associated projects program

Participant: Jean-Pierre Talpin.

The design productivity gap has been recognized by the semiconductor industry as one of the major threats to the continued growth of system-on-chips and embedded systems. Ad-hoc system-level design methodologies, that lift modeling to higher levels of abstraction, and the concept of intellectual property (IP), that promotes reuse of existing components, are essential steps to manage design complexity. However, the issue of compositional correctness arises with these steps. Given components from different manufacturers, designed with heterogeneous models, at different levels of abstraction, assembling them in a correct-by-construction manner is a difficult challenge. We address this challenge by proposing a behavioral type inference system to capture SystemC components' behavior at the interface level. The proposed type theory grounds a modeling and specification methodology, formulated in terms of a module system, that reduces compositional design correctness verification to the validation of synthesized proof obligations. The proposed type theory is conceptually minimal, equipped with a formal semantics, defined in a synchronous model of computation and supports a scalable notion and a flexible degree of abstraction. Our collaboration targets the *de facto* standard SystemC, yet with generic and language-independent techniques. Its applications range from the detection of local design errors to the compositional assembly of modules [44], [41].

9. Dissemination

9.1. Advisory

- Paul Le Guernic is executive board member of the Réseau National en Technologies Logicielles and steering committee member of the Réseau National en Micro-Nano Technologies.
- Jean-Pierre Talpin is elected member of INRIA's evaluation commission at INRIA, external advisory board member of the center of embedded systems at Virginia Tech, steering committee member of the ACM-IEEE conference on methods and models for codesign (MEMOCODE), steering committee member of the SLAP++ workshop series, organization committee member of the GALS workshop series, and editorial board member of the EURASIP Journal on Embedded Systems.

9.2. Conferences

- Jean-Pierre Talpin served as technical program committee member for the ACM-IEEE MEMOCODE'07, IEEE ACSD'07, IEEE SIES'07 and ACM SAC'07 conferences.

9.3. Thesis

- Jean-Pierre Talpin participated as examiner to the thesis defense of Yann Regis-Gianas at University Paris VII on November 29.

9.4. Teaching

- Thierry Gautier and Loïc Besnard taught on real-time programming at the DIIC 2 Graduate program of the University of Rennes I.
- Hugo Metivier taught programming scientific langage at the Licence 1 PCGI Graduate program, taught web design at the Licence 3 MIAGE and at the Master 2 graduate program of the University of Rennes I.
- Yann Glouche taught on functionnal programming at the STPI Graduate program of the INSA of Rennes.

9.5. Visits

- In the frame of the BALBOA associate projects program, and with the support of the University of Rennes I, Sandeep Shukla (VT) visited project Espresso in June 2007. Jean-Pierre Talpin visited Pr. Shukla, at VT, and Pr. Arvind, at MIT, in October 2007.

[16], [22], [24], [25], [15], [17], [18], [20], [21], [19], [23]

10. Bibliography

Major publications by the team in recent years

[1] B. BOUYSSOUNOUSE, J. SIFAKIS (editors). *Embedded Systems Design. The ARTIST Roadmap for Research and Development*, Thierry Gautier, contributor, Springer, Lecture Notes in Computer Science, Vol. 3436, 2005.

[2] T. P. AMAGBEGNON, L. BESNARD, P. LE GUERNIC. *Implementation of the Data-flow Synchronous Language Signal*, in "Proceedings of the ACM Symposium on Programming Languages Design and Implementation (PLDI'95)", ACM, 1995, p. 163–173.

[3] A. BENVENISTE, B. CAILLAUD, P. LE GUERNIC. *From synchrony to asynchrony*, in "CONCUR'99, Concurrency Theory, 10th International Conference", J. C. M. BAETEN, S. MAUW (editors), Lecture Notes in Computer Science, vol. 1664, Springer, August 1999, p. 162–177.

[4] A. BENVENISTE, P. CASPI, S. EDWARDS, N. HALBWACHS, P. LE GUERNIC, R. DE SIMONE. *The Synchronous Languages Twelve Years Later*, in "Proceedings of the IEEE Special issue on Modeling and Design of Embedded Systems", vol. 91(1), 2003.

[5] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT. *Synchronous programming with events and relations: the Signal language and its semantics*, in "Science of Computer Programming", vol. 16, 1991, p. 103-149.

[6] A. GAMATIÉ, T. GAUTIER. *Synchronous Modeling of Avionics Applications using the SIGNAL Language*, in "Proceedings of the 9th IEEE Real-time/Embedded technology and Applications symposium (RTAS'03), Washington D.C., USA", IEEE Press, May 2003.

[7] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Polychronous Design of Embedded Real-Time Applications*, in "ACM Transactions on Software Engineering and Methodology (TOSEM)", 2006.

[8] T. GAUTIER, P. LE GUERNIC. *Code generation in the SACRES project*, in "Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99, Huntingdon, UK", F. REDMILL, T. ANDERSON (editors), Springer, February 1999, p. 127–149.

[9] A. KOUNTOURIS, C. WOLINSKI. *High-level Pre-synthesis Optimization Steps using Hierarchical Conditional Dependency Graphs*, in "Proceedings of the EUROMICRO'99, Milan, Italie", IEEE Computer Society Press, August 1999.

[10] P. LE GUERNIC, T. GAUTIER. *Data-Flow to von Neumann: the Signal approach*, in "Advanced Topics in Data-Flow Computing", J. L. GAUDIOT, L. BIC (editors), 1991, p. 413–438.

[11] P. LE GUERNIC, T. GAUTIER, M. LE BORGNE, C. LE MAIRE. *Programming Real-Time Applications with Signal*, in "Proceedings of the IEEE", vol. 79, n^{o} 9, Septembre 1991, p. 1321–1336.

[12] P. LE GUERNIC, J.-P. TALPIN, J.-C. LE LANN. *Polychrony for system design*, in "Journal of Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design", 2003.

[13] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment*, in "Discrete Event Dynamic System: Theory and Applications", vol. 10, n^{o} 4, October 2000, p. 347–368.

[14] J.-P. TALPIN, P. LE GUERNIC. *An algebraic theory for behavioral modeling and protocol synthesis in system design*, in "Formal Methods in System Design", 2006.

Year Publications

Articles in refereed journals and book chapters

[15] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Synchronous Design of Avionic Applications based on Model Refinements*, in "Journal of Embedded Computing", 2007.

[16] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Polychronous Design of Embedded Real-Time Applications*, in "ACM Transactions on Software Engineering and Methodology (TOSEM)", 2007.

[17] L. MOREL. *Array Iterators in Lustre: from a language extension to its exploitation in validation*, in "EURASIP Journal of Embedded Computing", 2007.

Publications in Conferences and Workshops

[18] P. BOSTRM, L. MOREL, M. WALDEN. *Stepwise development of Simulink models using the refinement calculus framework*, in "International Colloquium on Theoretical Aspects of Computing", Springer Verlag, 2007.

[19] E. HERZOG, E. PALACHI, M. ZARKARI, T. GAUTIER, S. GRAF, M. WINOKUR, V. GAFNI. *Assessment of Existing Standards*, in "Deliverable D.2.5.1", IST project Speeds, 2007.

[20] L. MOREL, L. MANDEL. *Executable Contracts for Incremental Prototypes of Embedded Systems*, in "Workshop on Formal Foundations of Embedded Software and Component-Based Software Architectures", Electronic Notes in Theoretical Computer Science, Elsevier, 2007.

[21] OBJECT MANAGEMENT GROUP. *A UML Profile for MARTE, Beta 1*, in "Document n. ptc/07-08-04", OMG, 2007, http://www.omg.org/cgi-bin/doc?ptc/2007-08-04.

[22] J. OUY, J.-P. TALPIN, L. BESNARD, P. LE GUERNIC. *Separate compilation of polychronous specifications*, in "Formal Methods for Globally Asynchronous Locally Synchronous Design", Electronic Notes in Theoretical Computer Science, Elsevier, 2007.

[23] J. PERALTA. *Bridging Signal with the CADP model checker, a preliminary study*, in "Deliverable F342", ANR project Topcased, 2007.

[24] S. SHUKLA, S. SUHAIB, D. MATHAIKUTTY, J.-P. TALPIN. *On the polychronous approach to embedded software design*, in "Next generation design and verification methodologies for distributed embedded systems", Springer Verlag, 2007.

[25] J.-P. TALPIN, J. OUY, L. BESNARD, P. LE GUERNIC. *Compositional design of isochronous systems*, in "Technical report n. 6227", INRIA, 2007, http://hal.inria.fr/inria-00156499.

References in notes

[26] ATLAS GROUP (INRIA & LINA, UNIVERSITÉ DE NANTES). TCS, T*extual* C*oncrete* S*yntax, Reference site*, http://www.eclipse.org/gmt/tcs/.

[27] AIRLINES ELECTRONIC ENGINEERING COMMITTEE. *ARINC Report 651-1: Design Guidance for Integrated Modular Avionics*, Technical report, Aeronautical radio, Inc., Annapolis, Maryland, 1997.

[28] AIRLINES ELECTRONIC ENGINEERING COMMITTEE. *ARINC Specification 653: Avionics Application Software Standard Interface*, Technical report, Aeronautical radio, Inc., Annapolis, Maryland, 1997.

[29] B. BERTHOMIEU AND P.-O. RIBET AND AND F. VERNADAT. *The tool TINA - construction of abstract state spaces for Petri Nets and Time Petri Nets*, in "International Journal of Production Research", 2004.

[30] A. BENVENISTE, P. CASPI, L. CARLONI, A. SANGIOVANNI-VINCENTELLI. *Heterogeneous Reactive Systems Modeling and Correct-by-Construction Deployment*, in "Embedded Software Conference (EMSOFT'03)", Springer Verlag, 2003.

[31] B. BERTHOMIEU, ET AL. *The Syntax and Semantics of Fiacre*, in "Deliverable No. 4.2.4", ANR project OpenEmbeDD, 2007.

[32] L. BESNARD, T. GAUTIER, P. LE GUERNIC. *SIGNAL V4-INRIA version: Reference Manual*, http://www.irisa.fr/espresso/Polychrony/.

[33] J. BUCK, S. HA, E. A. LEE, D. G. MESSERSCHMITT. *Ptolemy: A Framework for Simulating and Prototyping Heterogenous Systems*, in "Int. Journal in Computer Simulation", vol. 4, n° 2, 1994, p. 155-182, http://citeseer.ist.psu.edu/buck92ptolemy.html.

[34] J.-L. COLACO, B. PAGANO, M. POUZET. *A conservative extension of synchronous data-flow with state machines*, in "In Embedded Software Conference.", ACM Press, 2005.

[35] H. GARAVEL, F. LANG, R. MATEESCU, W. SERWE. *CADP 2006: A Toolbox for the construction and Analisys of Distributed Processes*, in "International Conference on Computer Aided Verification", Springer, 2007.

[36] E. GIMÉNEZ. *Un Calcul de Constructions Infinies et son Application à la Vérification des Systémes Communicants*, Ph. D. Thesis, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, December 1996.

[37] M. KERBOEUF, D. NOWAK, J.-P. TALPIN. *The steam-boiler problem in SIGNAL-COQ*, in "International Conference on Theorem Proving in Higher-Order Logics", Lecture Notes in Computer Science, Springer Verlag, 2000.

[38] M. KERBOEUF, D. NOWAK, J.-P. TALPIN. *Formal proof of a polychronous protocol for loosely time-triggered architectures*, in "Formal Methods and Software Engineering: 5th International Conference on Formal Engineering Methods", Lecture Notes in Computer Science n. 2885, Springer Verlag, 2003.

[39] F. MARANINCHI, Y. RÉMOND. *Mode-automata: a new domain-specific construct for the development of safe critical systems*, in "Sci. Comput. Program.", vol. 46, n° 3, 2003, p. 219–254.

[40] D. NOWAK, J.-R. BEAUVAIS, J.-P. TALPIN. *Co-inductive axiomatization of a synchronous language*, in "International Conference on Theorem Proving in Higher-Order Logics", Lecture Notes in Computer Science, Springer Verlag, 1998.

[41] H. D. PATEL, D. A. MATHAIKUTTY, D. BERNER, S. SHUKLA. *ARH: service-oriented architecture for validating system-level designs*, in "IEEE Transactions on Computer-Aided Design", vol. 25, n° 8, 2006, p. 1458-1474.

[42] A. PNUELI, O. SHTRICHMAN, M. SIEGEL. *Translation validation: from Signal to C*, in "Correct System Design Recent Insights and Advance", Lecture Notes in Computer Science, vol. 1710, Springer Verlag, 2000.

[43] E. RUTTEN, F. MARTINEZ. *Signal GTI: implementing task preemption and time intervals in the synchronous data flow language Signal*, in "Proceedings of the 7th Euromicro Workshop on Real-Time Systems, Odense, Denmark", IEEE Publ., june 1995.

[44] S. SUHAIB, D. MATHAIKUTTY, D. BERNER, S. SHUKLA. *Validating Families of Latency Insensitive Protocols*, in "IEEE Transactions on Computers", vol. 55, n° 11, 2006, p. 1391-1401.

[45] J.-P. TALPIN, C. BRUNETTE, T. GAUTIER, A. GAMATIÉ. *Polychronous mode automata*, in "Embedded Software Conference, ACM Press", September 2006.

[46] J.-P. TALPIN, C. BRUNETTE, T. GAUTIER, A. GAMATIÉ. *Polychronous mode automata*, in "Embedded Software Conference, ACM Press", September 2006.

[47] TRISKELL TEAM (INRIA). *Sintaks, reference site*, http://www.kermeta.org/sintaks/.

[48] F. VERNADAT, C. PERCEBOIS, P. FARAIL, R. VINGERHOES, A. ROSSIGNOL, J.-P. TALPIN, D. CHEMOUIL. *The Topcased project - a toolkit in open-source for critical application and system development*, in "International Space System Engineering Conference, Eurospace", May 2006.

[49] B. WERNER. *Une Théorie des Constructions Inductives*, Ph. D. Thesis, Université Paris VII, May 1994.