INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Espresso

Modélisation de systèmes réactifs polychrones

Rennes

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, INRIA, starting Nov. 1st.]
Lionel Morel [Post-Doctorate, INRIA, starting Sept. 1st.]
Hamoudi Kalla [Post-Doctorate, INRIA, until Sept. 1st]

Doctorate Students
David Berner [INRIA, until Jan. 31th.]
Yann Glouche [INRIA, EADS Fundation, starting Dec. 15th.]
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 [9]. 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 [9] 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 [29] 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 $t_3 \neg \leq t_1$.

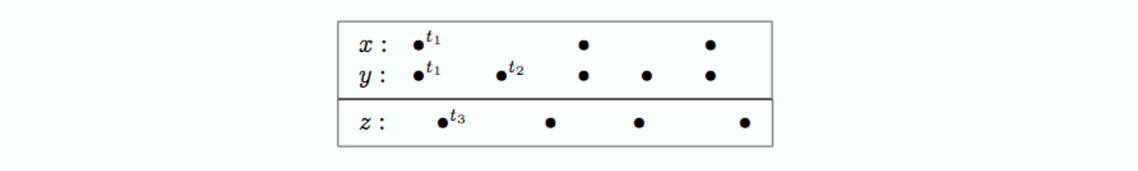


Figure 1. Behavior b over three signals x, y and z in two clock domains

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.

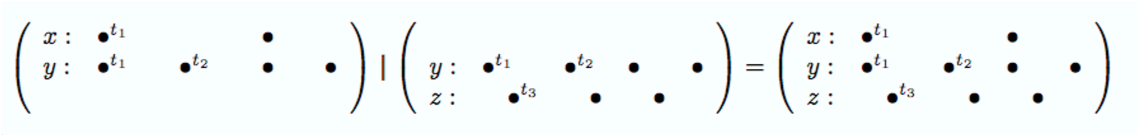


Figure 2. Synchronous composition of $b \in p$ and $c \in q$

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 $t' \neg < t$.

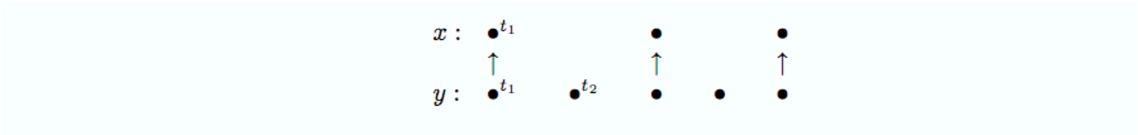


Figure 3. Scheduling relations between simultaneous events

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 $d \leq c$.

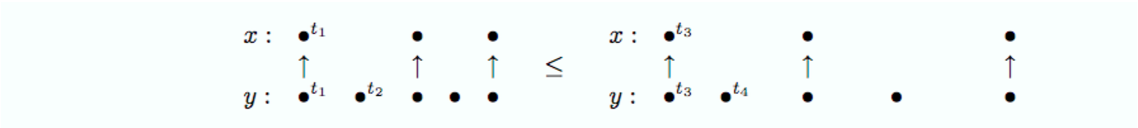


Figure 4. Relating synchronous behaviors by stretching.

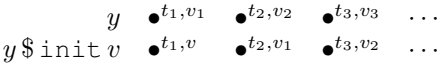
3.1.2. A declarative design languages

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.

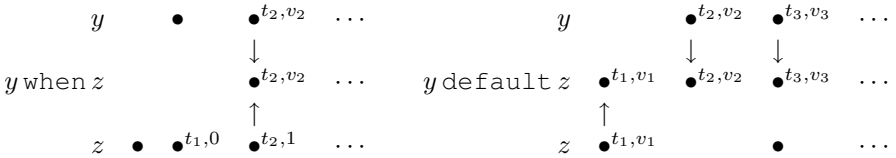
$$P, Q ::= x = y\ f\ z \ \mid\ P/x \ \mid\ P \,|\, Q$$

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).

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

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.

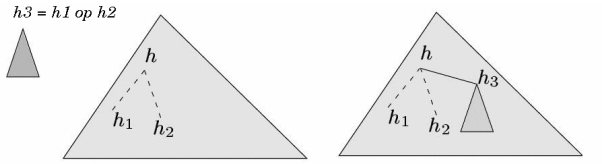


Figure 6. Hierarchization of clocks

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:

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

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:

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

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 [35], based on the calculus of constructions [46], 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 [36].

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 [41], 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).

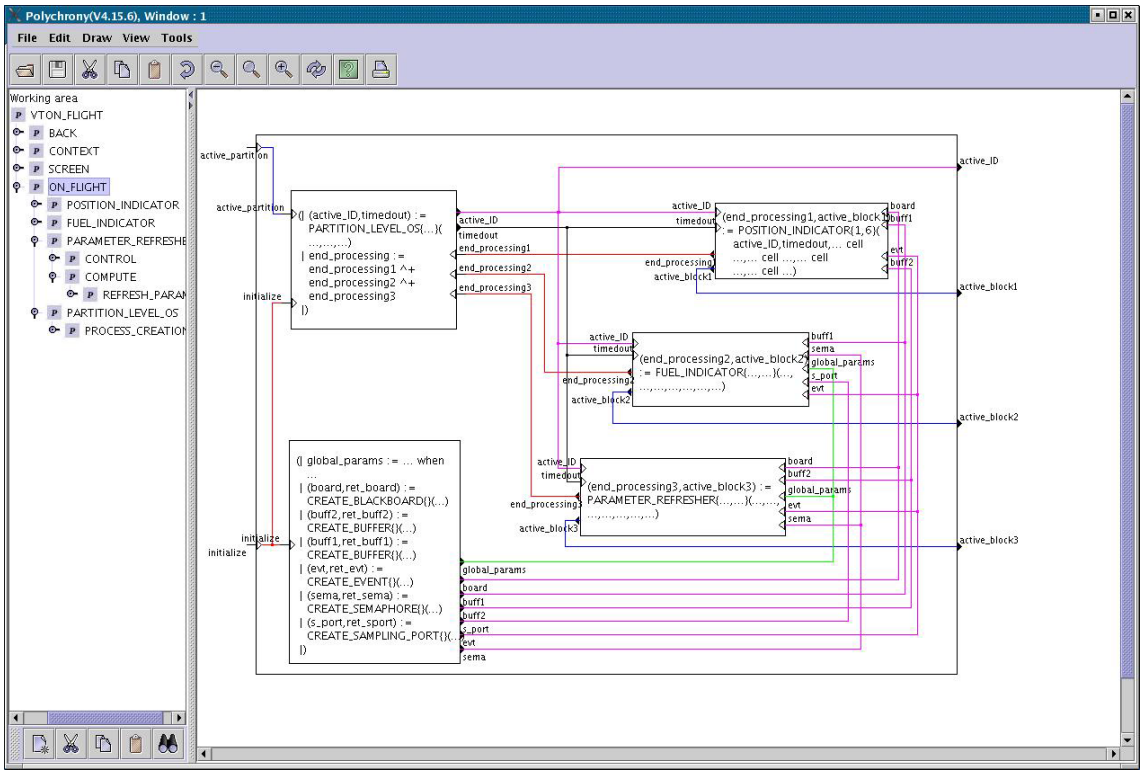


Figure 7. Avionics application modeling using the visual editor of the Polychrony workbench

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*.

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 [36]. 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. Affine clock calculus

Participants: Loïc Besnard, Thierry Gautier.

The affine clock calculus [43] is an extension of the boolean clock calculus based on free boolean conditions. The affine relations allow to express that successive values of some signal are provided at specific micro-instants between any two successive macro-instants in a regular manner. To express affine relations, three predefined processes have been introduced.

affine_sample={integer ϕ***, d} (?*** x ***!*** y ***)***, with $\phi \geq 0$ and $d > 0$, defines a signal y as an undersampling of an other one x. A value of y is available each d^{th} value of x, and the occurrence of the first value of y is given by the phases $(\phi + 1)$. For $\phi = 3$ and $n = 4$, the process is illustrated on figure 8.

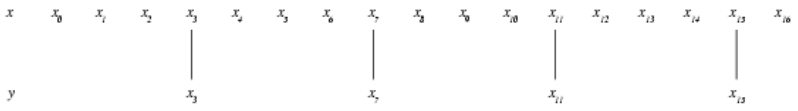


Figure 8. Example: y := affine_sample{3,4} (x)

affine_clock_relation={integer n***,*** ϕ***, d} (?*** x***,*** y***)*** defines the fact that the clock of the input signals x and y are in affine relation with n, ϕ, d as parameters. To implement this process, a clock (I) greater than the clock of x and y is built such that the clock of x is synchronized with the clock of *affine_sample* { $\max(0, -\phi)$, n }(I), and the clock of y is synchronized with the clock of *affine_sample* { $\max(0, \phi)$, d }(I). For $n = 5$, $\phi = 4$ and $d = 7$, the process is illustrated on figure 9.

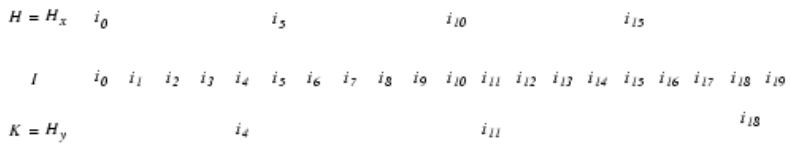


Figure 9. Example: clock_affine{5,4,7}(x,y)

affine_unsample={integer n***,*** ϕ***} (?*** x***,*** z ***!*** y***)*** with $n > 0$ and $\phi \geq 0$, defines the signal y, synchronized with the signal z, as an oversampling from the input signal x; the input signal z is used to fix the values of y when x is absent. For $n = 3$, $\phi = 1$, the process is illustrated on figure 10.

5.5. A meta-model of Polychrony

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

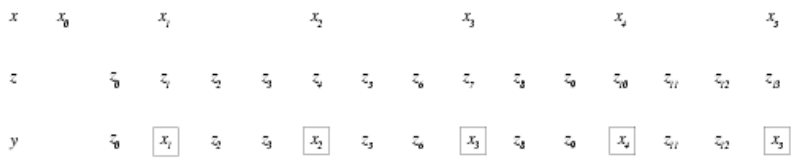


Figure 10. Example: y := clock_unsample{3,1}(x, z)

We have developed a metamodel of Polychrony in the Generic Modeling Environment (GME) detailed in Sections (6.2, 6.3, and 6.4.

- Signal-Meta is the metamodel of the SIGNAL language. It describes all syntactic elements specified in [30]: 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.

6. New Results

6.1. The UML profile MARTE for Real-Time and Embedded Systems Design

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

The collaboration between the AOSTE, DART, ESPRESSO teams from INRIA, CEA and THALES Research and Technology, in the context of the CARROLL Research Programme (see http://www.carroll-research.org) aims at the standardization of a UML 2.0 profile for real-time embedded systems (MARTE) before the Object Management Group (OMG). The main objectives of MARTE are the following : defining time, concurrency and communication models, mixing control-flow and intensive computational data-paths, modeling architectural platforms and adopting Y-chart approaches for allocation of application functions onto architectural resources). It bears strong connections with other OMG standardization attempts (some accepted already, some only proposed) such as the SysML (System Engineering in UML) standard, or AADL and UML4SoC current RFC proposals. The MARTE profile is divided in three subparts (some would say subprofiles):

- TCR (Time and Concurrent Resources) is meant to provide the infrastructure notions of logical/discrete and physical/real time, and the basic concurrency and communication models relevant to the profile. It should extend in many ways the corresponding parts of the SPT profile, in particular in adding the notions of synchronous/clocked systems (with synchrony and priorities).

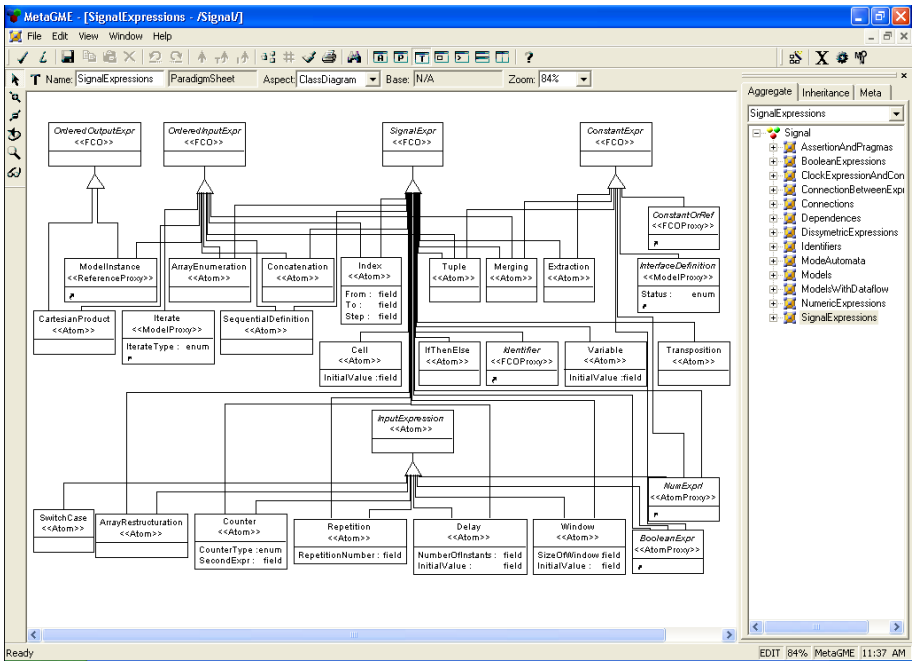


Figure 11. A meta-model of Polychrony in the GME

- SPA (Schedulability and Performance Analysis) should provide with features allowing the non-functional performance evaluation and static or dynamic scheduling policies of systems.
- RTEM (Real-Time Embedded Modeling) will take these time informations into account to provide for behavioral definitions of hierarchical models, as in state and activity diagrams for instance). It also claims for independent high-level modeling of architectural platforms and the platform-based design methodology using useful feature of the two previous subprofiles to build and model optimized allocation links between application and architectures. It also requires dedicated modeling features for frequently encountered structures in real-time and embedded systems.

A first contribution of Espresso to MARTE, Figure 12, is to propose a model of the APEX-ARINC 653 operating system services, based on related work reported in Section 6.4.

In the MARTE profile for UML, Espresso proposes a Chapter on real-time and embedded application modeling (RTEAM). The Chapter is concerned with the association of a timing model with behavioral description diagrams such as activities or state-machines. The main features of this combination appear in Figure 13, which depicts the model of an alarm clock using timed activity diagrams.

It is composed of the timed activity Count which counts the number of ticks comprised between a start and a stop event. It is composed of the timed activity Check which emits an alarm when the count becomes 0. A causality dependency schedules the execution of Count before (or at the same time as) Check. Upon completion, Check either terminates the activity (if the Alarm is raised) or loops back with a clock unit delay until the next event occurs. All control and object flows in red denote an implicit "immediate" stereotype that indicate branches that can be executed immediately. no buffer is provided). All control and object flows with a delay stereotype are executed at the next unit of the parent activity clock.

The activity Count is complemented with a clock relation which defines its clock CountClk by the union of the canonical clocks of its start, stop and tick input events. The clock of the Check activity is defined by

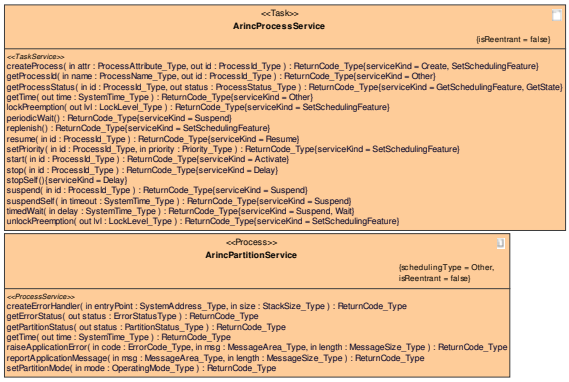


Figure 12. Part of the ARINC 653 Library in the MARTE profile

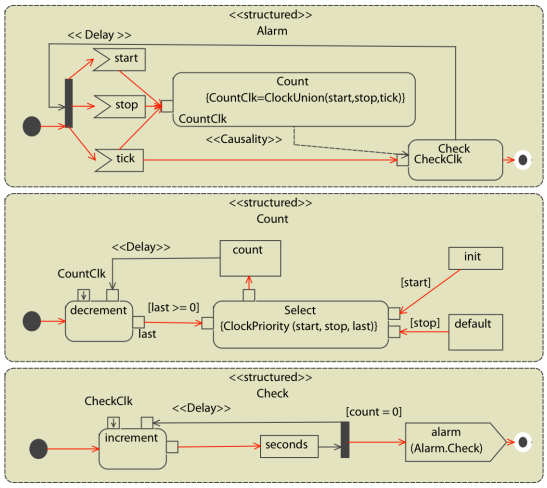


Figure 13. Timed Activity Diagram for an Alarm

the canonical clock of the timed event tick. Additionally, a dependency specifies a scheduling relation that enforces the execution of Count to happen before or at the same time as Check. By looking into the definition of Count and Check, one indeed notices that Count computes the count value that is tested against 0 in the Check activity. An explicit causality stereotypes helps getting one aware of that constraint. Another clock relation is depicted by the OCL constraint of the Select activity of the Count diagram. While the events start, stop and the guard `[last >= 0]` have no explicit clock relation, it is necessary to deterministically choose one of these events in order to deterministically set one of the init, default or last values to count. This is done by making a priority relation between start, stop and last explicit.

6.2. A meta-model of Polychrony

Keywords: *Generic Modeling Environment*, *Metamodeling*, *Model transformation*, *SIGNAL.*

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

In [19], we present the meta-model of Polychrony in the GME. The aim of this work is to generalize the use of formal methods implemented in Polychrony by making them accessible in more popular framework, such as Eclipse. However, in a world of rapid technology obsolescence, model engineering must be platform independent. To achieve this independence, the higher their abstraction expression level is, the more adaptable to various operational environments they will be. Model Driven Software Development is based on a number of common principles such as like XMI, OCL and UML, that can be mapped to different standards and different environments. Thus, we choose to express the SIGNAL language as a metamodel, called Signal-Meta, using these technologies.

To develop our metamodeling approach, we choose the Generic Modeling Environment (GME) developed by the ISIS institute at Vanderbilt University. GME is a configurable UML-based toolkit that supports the creation of domain-specific modeling and program synthesis environments [38]. Metamodels are proposed in the environment to describe *modeling paradigms* for specific domains. Such a paradigm includes, for a given domain, the necessary basic concepts to represent models from a syntactical viewpoint to a semantical one.

Describing a metamodel in GME consists in modeling all paradigm concepts as classes through usual UML class diagrams using some predefined UML-stereotypes. In these class diagrams, GME provides a means to express the visibility of components within a model through the notion of *Aspect* (i.e. one can decide which parts of the descriptions are visible depending on their associated aspects). Signal-Meta comprises three main Aspects: *Interface*, *Computation part* and *Clock and Dependence Relations*. The first Aspect manages all input/output signals and static parameters. The two other reflects respectively data-flow relations and clock relations between signals. Figure 14 represents the description of a modem using Signal-Meta in GME. At the bottom of the windows, the left frame contains all concepts that can be manipulated in the upper frame by drag&drop.

The graphical description constitutes a good front-end for SIGNAL specifications. To complete this front-end, we need a mean to transform the graphical Signal-Meta specifications in the SIGNAL language. GME offers a means to develop and plug components into the GME environment. The role of such a component consists of interacting with the graphical designs. GME distinguishes different families of components that can be plugged to its environment depending of their role. We developed an Interpreter whose role is to check information, such as the correctness of a model, and/or produce a result, such as a description file. This interpreter, outlined Figure 15, is developed in C++ using the *Builder Object Network* (BON2) API provided with GME.

6.3. Multi-clocked mode automata

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

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

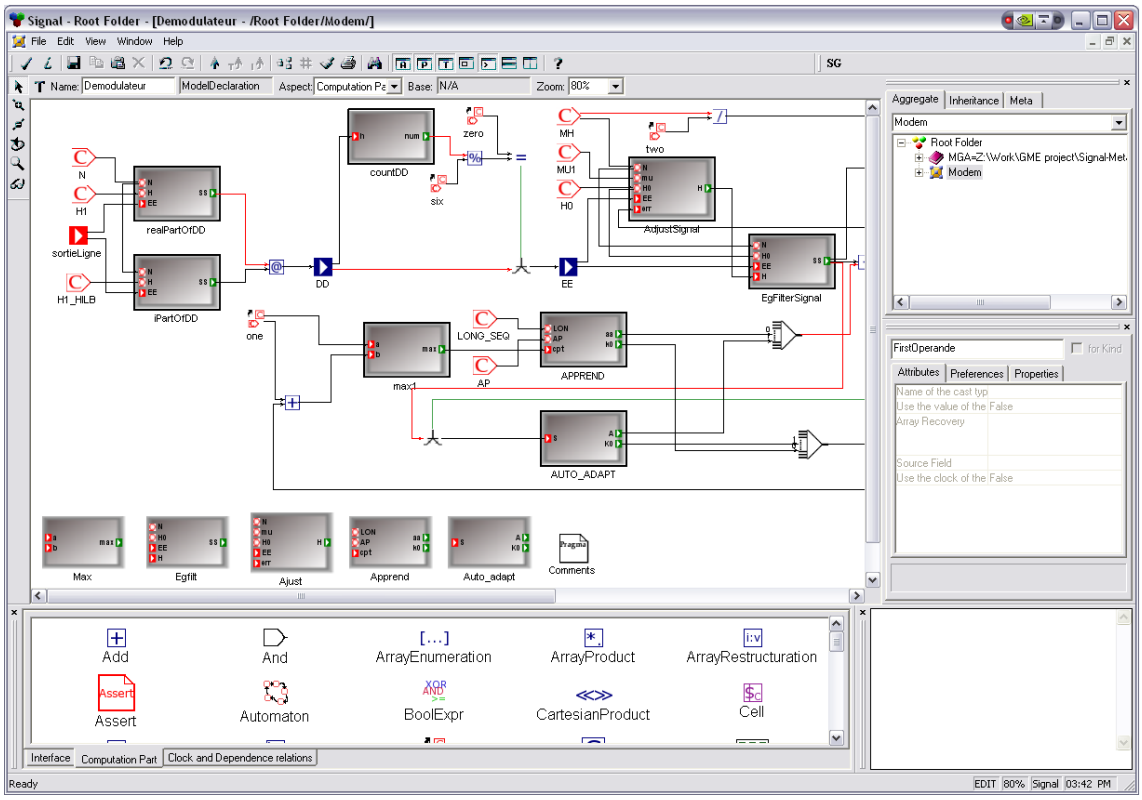


Figure 14. Description of a Modem in GME.

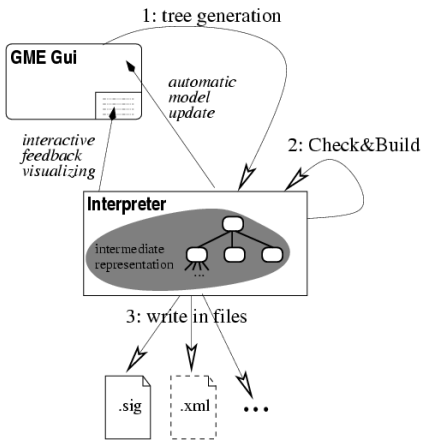


Figure 15. Generation of SIGNAL models from GME.

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 [31] have been proposed until recently, the latest advance being the combination of stream functions with automata in [33]. 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 [42], no attempt has been made to extend mode automata with the capability to model multi-clocked systems and multi-rate systems. In [23], 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).

6.3.1. Example of a switch

To illustrate our modeling techniques, we consider the example of a simple crossbar switch (see Figure 16). The switch is a typical example of specification where an imperative automata-like structure superimposed to a native data-flow structure gives a shorter and more intuitive description of the system's behavior. The mode automata of the switch consists of two states flip and flop, in which routing is performed from $y_{1,2}$ to either $x_{1,2}$ or $x_{2,1}$ depending on the current mode of the automaton. The mode toggles from flip to flop, or converse, upon an occurrence of the event r (see Figure 16).

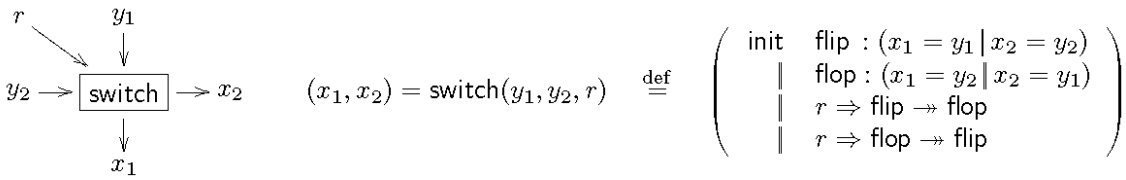
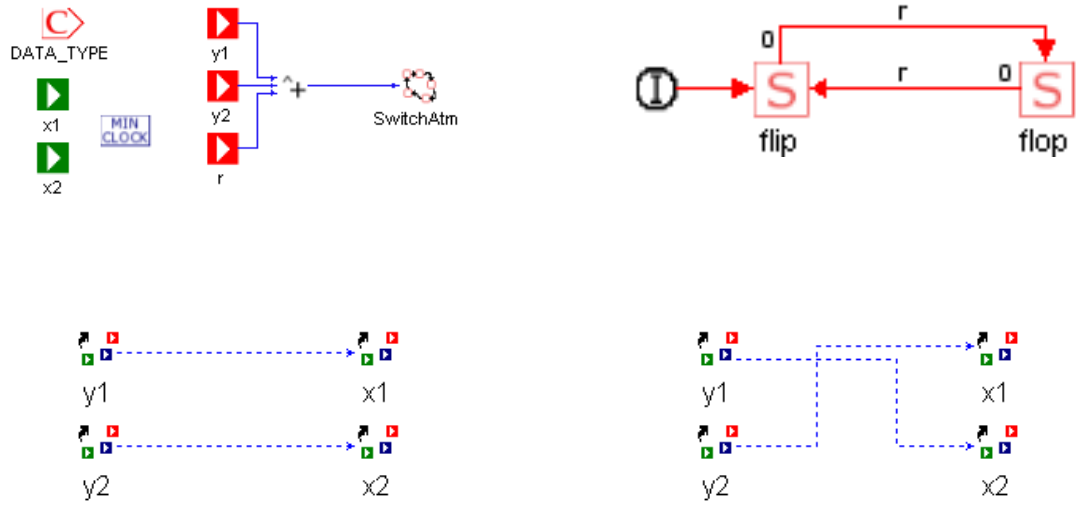


Figure 16. Description of the crossbar switch.

The left of Figure 17 represents the switch process in which `y1`, `y2`, and `r` are declared as input signals, `x1` and `x2` as output signals, and `SwitchAtm` as the mode automaton. `DATA_TYPE` is a parameter only used to define a generic type for input and output signals. The `SwitchAtm` object is a container in which all its states are specified (see right of Figure 17). The `SwitchAtm` automaton contains two terminal states (`flip` and `flop`). *StrongTransitions* are guarded by the event `r`, as labeled on the middle of transitions. The 0 indicates the priority of the transition, which has been added to guarantee the determinism of a mode automata if there are more than one outgoing transition on a state. The left of Figure 17 also represents the synchronization of the `SwitchAtm` clock with the union of the clock of `y1`, `y2`, and `r`. Because output signals are partially defined in states (see the content of state `flip` (resp. `flop`) at the left (resp. right) of Figure 17), their clocks have to be specified explicitly. Therefore, the *MinClock* operator is used to define them as the union of clocks of their partial definitions.



Content of states `flip` and `flop`.

Figure 17. The `Switch` process and the `SwitchAtm` mode automaton specifications in GME.

6.3.2. Model Transformation

The transformation consists in interpreting the graphical formalism as a SIGNAL specification. Therefore, we have extended the Signal-Meta interpreter to support the mode automata extension. The code below corresponds to the application of the interpreter on the switch example specified in Figure 17.

```
process Switch =
  { type DATA_TYPE; }
  ( ? DATA_TYPE y1,y2; event r; ! DATA_TYPE x1, x2; )
  (| min_clock(x2) | min_clock(x1)
   | %Atm%(| __ST_0_flop_To_flip := when (r) when (_Atm_0_zNextState = #flop)
           | __ST_1_flip_To_flop := when (r) when (_Atm_0_zNextState = #flip)
           | _Atm_0_currentState ^= (y1 ^+ y2 ^+ r)
           | _Atm_0_nextState := _Atm_0_currentState
           | _Atm_0_currentState := #flip when __ST_0_flop_To_flip
                                 default #flop when __ST_1_flip_To_flop
                                 default _Atm_0_zNextState
           | _Atm_0_previousState := _Atm_0_currentState$ init #flip
           | _Atm_0_zNextState := _Atm_0_nextState$ init #flip
           | case _Atm_0_currentState in
             {#flop}: (| x2 ::= y1 | x1 ::= y2 |)
             {#flip}: (| x2 ::= y2 | x1 ::= y1 |)
             end
           |)
          where
             event __ST_0_flop_To_flip, __ST_1_flip_To_flop;
             type _Atm_0_type = enum(flop, flip);
             _Atm_0_type _Atm_0_currentState, _Atm_0_previousState;
             _Atm_0_type _Atm_0_nextState, _Atm_0_zNextState;
          end
```

```
|); % process Switch
```

6.4. A modeling paradigm for Integrated Modular Avionics design

Keywords: *Generic Modeling Environment*, *Integrated Modular Avionics*, *metamodeling*.

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

We previously addressed the design of applications based on the *Integrated Modular Avionics* (IMA) architecture [13], [14], which relies on the avionic standard APEX-ARINC [27], [28]. This leads to the implementation of a library of components in Signal, providing real-time executive services defined by the APEX-ARINC standard.

Now, we carry out this library in the *General Modeling Environment* (GME) [18]. The primary purpose is to increase the usability of the library by proposing the same concepts within a non domain-specific tool such as GME. Therefore, without being an expert of synchronous technologies, a user could still be able to design applications based on the IMA modeling approach proposed in the Polychrony environment. Today, we observe that the attention of the industry tends to shift to frameworks based on general-purpose modeling formalisms (e.g. UML), in response to a growing industry demand for higher abstraction-levels in the system design process.

GME [38] is a configurable object-oriented toolkit, which supports the creation of domain-specific modeling and program synthesis environments. *Metamodels* are proposed in the environment to describe *modeling paradigms* for specific domains: basic concepts required for model representation from a syntactical viewpoint to a semantical one.

Our modeling paradigm for IMA design in GME, called MIMAD, is represented by the layer on the top in Figure 18. The layers on the bottom are dedicated to domain-specific technologies. Here, we consider Polychrony, which is associated with Signal. However, one can observe that the idea is extensible to further technologies that offer specific useful functionalities to the MIMAD layer (e.g., the integrated environment UPPAAL, which enables validation and verification of real-time systems using timed automata). As GME enables to import and export XML files, information exchange between layers can rely on this intermediate format. This favors a high flexibility and interoperability.

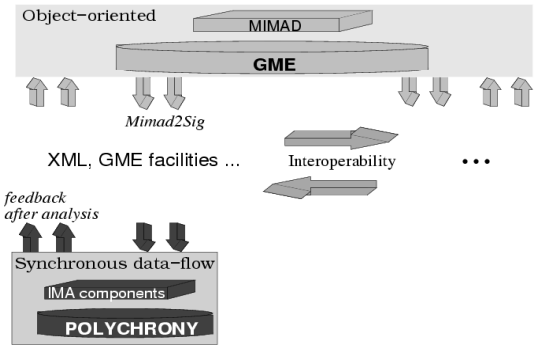


Figure 18. A component-oriented modeling framework for IMA design.

The MIMAD layer 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. Component parameters (e.g. period or deadline of an IMA process model) can be specified. The resulting GME model is transformed in Signal (referred to as *Mimad2Sig* in Figure 18) based on the XML intermediate format.

In the synchronous data-flow layer, the XML description obtained from the upper layer is used to generate a corresponding Signal model of the initial application description. This is achieved by using the IMA-based components already defined in PolychronyPolychrony [6]. Thereon, the formal analysis and transformation techniques available in the platform can be applied to the generated Signal specification. Finally, a feedback is sent to the MIMAD layer to notify the user with possible incoherences in initial descriptions.

6.5. An Eclipse plugin for Polychrony

Keywords: *Eclipse*, *Ecore*, *Metamodeling*, *Model transformation*, *SIGNAL*.

Participants: Christian Brunette, Jean-Pierre Talpin.

In the frame of the Topcased project and of the OpenEmbeDD project, we have to create an eclipse plugin for Polychrony. So, within a collaboration initiated at INRIA in the ATLAS team in Nantes, we study migration of existing GME projects, and particularly Signal-Meta, into Eclipse.

6.5.1. From GME to Eclipse

The ATLAS team has defined a model engineering support on top of the Eclipse Modeling Framework (EMF), called the AMMA (ATLAS Model Management Architecture) platform [25]. To be able to exchange models between an EMF based system and a corresponding GME assumes an abstract understanding of both architectures and a precise organization of the interoperability scheme. EMF and GME allows metamodel and model management. The designed models conforms to a previously defined metamodel. Any metamodel design assumes the existence of a metametamodel (implicit or explicit). So three levels have to be considered: metametamodel concepts mapping (M3), building metamodel projectors (M2) and building model projectors (M1). Projectors are operational bridges between different technical spaces, and are realize here using ATL (ATLAS Transformation Language), which allows model transformations in EMF technical space. The metamodel bridge is already effective. Only a part of the metamodel concepts are translated into the ATLAS file. All graphical information are lost. The file contains only information about concepts and their relations (e.g. inheritance, containment). So, it is possible to work on the old GME project into EMF. The GReAT [26] transformation language provided with GME can also be replaced by ATL to work on the produced artifacts. This work is described in [32] and has been pursued this year by some improvements to keep more information in the EMF Model. For example, the OCL constraints are added as specific comments, which can then be used by OCL constraints checker provided in Eclipse. The translation has also been improved to fit more precisely to EMF metamodels.

6.5.2. The new Polychrony environment

This translation gives a first EMF metamodel, which has been modified to exploit the power of the Eclipse environment. From this metamodel, EMF can generate a plugin to build models based on such metamodel (see figure 20). Using the modeling facilities provided with the Topcased framework, we create a graphical environment for Polychrony (see figure 19).

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.

In the frame of the OpenEmbeDD project, we study the possibility of creating a unique environment for all tools (Polychrony, Syndex [44], and Gaspard [34]) based on the synchronous paradigm. So, we began this work by creating a UML metamodel in which we merge all common concepts of the three tools and we add all specific ones.

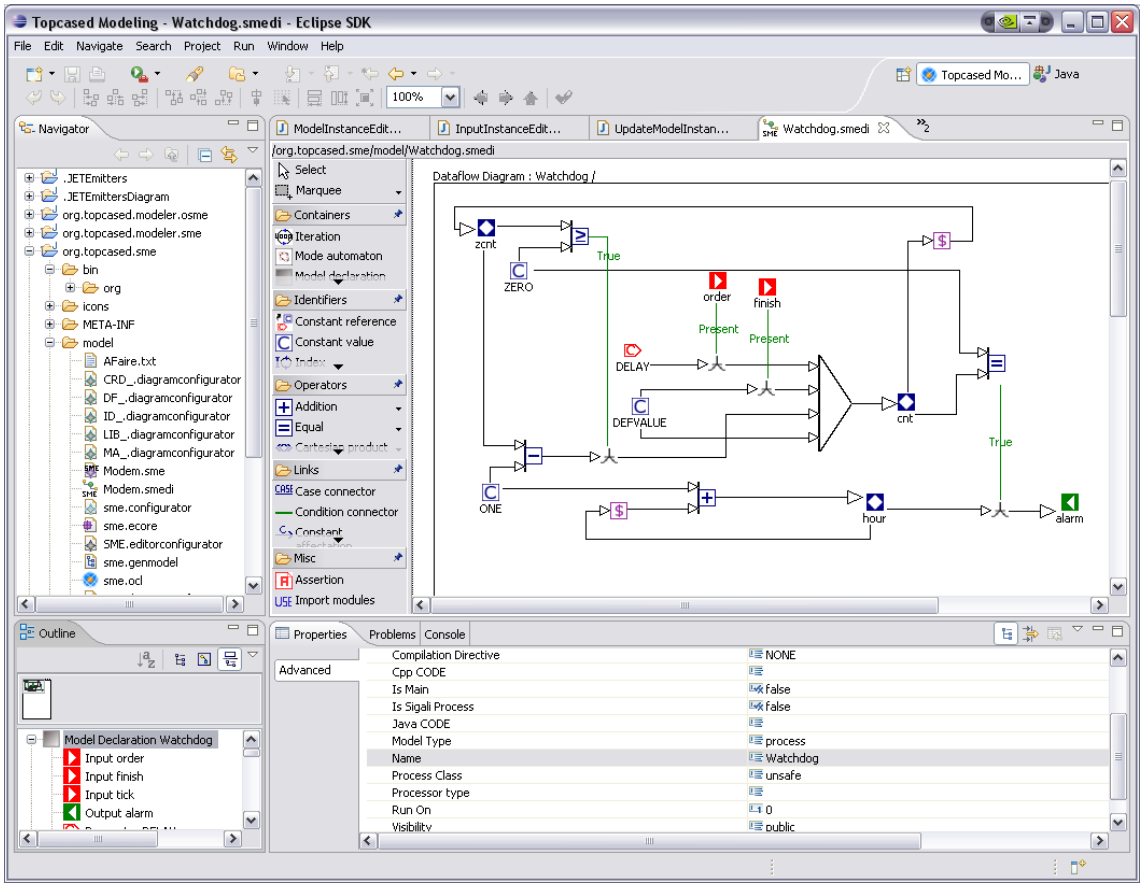


Figure 19. Signal-Meta environment in Eclipse.

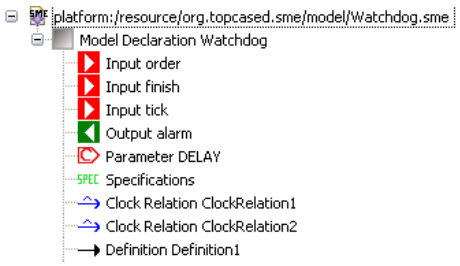


Figure 20. EMF plugin for Signal-Meta.

6.6. Rapid Prototyping of Heterogeneous Real-time Systems

Participants: Loïc Besnard, Hamoudi Kalla, Saïd Lankri, Jean-Pierre Talpin.

6.6.1. Importing C components

The ESPRESSO team has pursued his effort on importing C components into Signal [21]. This work is based on the GCC compiler starting from its version 4. The compiler takes an arbitrary C source code, does the compilation and outputs a tree called GNU TREE SSA (SSA - *Single Statement Assignation*) which represents the original C source code in a sort of high level assembler instructions. The importer takes the resulting tree and produces Signal code. This approach is convenient for reusing the many existing C components and performing static code verification by the means of the Signal compiler.

In an alternative approach, we focus on a particular form of C components, specifically those who have a strong algorithmic part. This type of C code is declined most of the time with computational parts that are not compatible with the synchronous hypothesis (single assignment) and command parts. The aim is to split the imported C component into two parts : the control parts are extracted and translated into Signal formalism, the computational part is left in the C formalism and embedded in the resulting Signal component.

Many reasons are motivating this type of import. First, we do not really need to translate computational parts, which are very well handled in the originating C language and cannot be simply imported. In addition, Signal is not well designed for such type of programming. Second, the control/command part can be easily handled in Signal, after the import we obtain a component whose 'decisional' parts are written in Signal and computational parts embedded as C code.

This import relies on standard technologies. The C component is first translated into its corresponding XML structure with the help of a tool called SRCML, then a set of transformation rules are applied in order to obtain Signal code and embed the computational parts. We have chosen a W3C standard, XSLT for these rules.

6.6.2. Importing Simulink components

Simulink has become the de facto standard tool largely used in the design of discrete-time systems. In order to make the Signal language as a heterogeneous formalism integrator, we need to have correct translations from Simulink to Signal. In this transformation, we offer a way of translating discrete-time Simulink models to Signal, preserving by the way the component structure and hierarchization. We focus of discrete systems using a fixed-step simulation in order to get a Signal model which is semantically equivalent.

Each Simulink 'box' is translated into a Signal process and the known Simulink types are mapped to their corrsponding types in Signal. Types are not mandatory in Simulink, in that case, we let the type-inference system of Signal guess the types of the inputs and outputs.

In addition, we have written a Signal library of many discrete Simulink components. The translated models make use of this library which is organized the same way `libsimulink` is in MATLAB.

Both the library and the transformation rules are extensible and the addition of new known Simulink components is easy. As for C import, we first translate Simulink models into their XML representation, and apply a set of rules to get a Signal code. We use the same XSLT standard in which transformational rules are expressed.

6.7. New features of Polychrony

Participants: Loïc Besnard, Thierry Gautier.

The focus of this year is the *open-source* release of the environment: all the data structures, classes, methods have been documented. The associated documentation is automatically generated using "doxygen" tool. Moreover, this year we have integrated all the sources (Signal batch Compiler, Graphical User Interface and Sigali) under Inria-Gforge (http://gforge.inria.fr). The automatic production of the releases, on LINUX, SUNOS, MACOS X and WINDOWS has been also developped.

A Java interface of some classes of Polychrony (written in C and C++) has been developped. Currently, only the higth level functionalities are provided. It is used by MBDA (Section 6.6) to define an embedded system design environment for the integration of heterogeneous components. Finally, the source of the Sigali tool has been reorganized to avoid the duplication of the code between the sigali engine and the solver part, used for controller synthesis.

6.8. Synthesis of latency-insensitive systems

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

We are continuing our effort initiated in [45] to provide an implementation of weakly-endochronous or self-synchronizing 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.

C code can be generated for such programs with just a few modifications to the existing Signal compiler. We are presently working on the definition of a (non-expensive) static analysis of such programs to determine which of them are self-synchronizing.

On a different path, we are exploring a way to compile thoseself-synchronizing programs into object-oriented C++ programs. Those programs are reactive, dynamically self-scheduled and able to support internal concurrency. We are working on simplifying the required scheduling runtime system and on finding the optimal granularity of code to schedule.

6.9. Periodic clock relations

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

We try to extend the clock calculus of Signal with periodic dependencies between signals. These relations can be expressed with infinite periodic binary words. This sort of constraints is more expressive than the calculus on static dependency to represent the clocks dependency in the execution of the Signal program but more complex to extract.

We are working on a non-assisted method to infer the periodic dependency from Signal equations, or to prove property on Signal variables like an abstract interpretation. 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 calculus with infinite periodic binary words could be use to analyse the scheduling of the instructions generated from a program Signal, and we could try to calculate an bounded time for the execution of the program for a set of instant.

7. Contracts and Grants with Industry

7.1. Carroll project Cortess (10/2006-10/2007)

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

The partners of the **CARROLL project Cortess** (http://www.carroll-research.org) are Thales, CEA-List and the INRIA project-teams Espresso, Aoste and Dart. The aim of the project Cortess is to continue the effort of the former ProtesÊproject to standardize the UML profile for real-time and embedded systems (MARTE) before the OMG. The contribution of the Espresso team to MARTE is decribed Section 6.1.

7.2. Network of excellence Artist2

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

The Espresso project-team is involved in the activity of the Artist2 network of excellence. The book [1] and the URL http://www.artist-embedded.org/FP6) give detailed presentations on the aim and scope of the network.

7.3. Sub-contractant for MBDA

Participants: Loïc Besnard, Jean-Pierre Talpin.

In the context of the ANR project Systematics, the Espresso project-team collaborates with MBDA division of EADS to develop a embedded system design environment based on Polychrony for the integration of heterogeneous components (in C, Lustre, Simulink). Work conducted in this project is reported in Sectionõ6.6.

7.4. 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 7.6, allowing for the seamless integration of heterogeneous embedded system components within a contract-based design MDD environment.

7.5. AESE project Topcased

Participants: Christian Brunette, 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 [24]. Project Topcased is being funded by the ANR and the Midi-Pyréné region.

7.6. 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).

Project Espresso is hosting a platform expert-engineer, Fabien Fillion, whose goal will be to integrate synchronous and asynchronous design environments developped in the frame of the project within a unified environment.

7.7. Rockwell-Collins France

Participants: Loïc Besnard, Jean-Pierre Talpin.

In the frame of the Topcased consortium, the Espresso project-team initiated a bilateral collaboration with Rockwell-Collins France and the University of Minessota. The aim of this initiative is to investigate the use of Polychrony for the system-level design of globally asynchronous and locally synchronous (GALS) avionics architectures. A case study is being carried out to entirely design and verify a Dual Flight Guidance System in collaboration with the University of Minnesota.

7.8. IST project Speeds

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

The Espresso project-team is involved in the activity of 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.

The project will enable European systems industry to evolve from model-based design of embedded systems, towards integrated component based construction of complete virtual system models. It will include:

- Construction of early system prototypes during the design stage.
- Thorough quality and stability assessment at early design stages.
- Active treatment of design assumptions to guide system development.
- Support for concurrent multi-organization development of complex designs.

For ensuring a good acceptance of the SPEEDS approach from the industrial partners, system modeling will be adressed using the following layers:

- Users will stay with their usual design tools and high-level languages;
- A SPEEDS core meta-model will provide a low-level common modeling language, independant from the pre-cited tools. Translation from these high-level languages to the core meta-model will be ensured;
- A rigorous formal semantics of the core meta-model will ensure the consistency of SPEEDS modeling and provides a basis for system analysis.

The project thus includes the definition of a formal model for heterogeneous embedded systems called "HRC" for Heterogeneous Rich Components.

The contribution of the Espresso project-team is to demonstrate how the design of such HRC-based systems can benefit from the existing design and validation experience related to Polychronous systems.

Starting from SignalMeta[19], a profile implementing the Polychronous MoCC[9], a Polychronous Profile will be defined as a library of derived concepts for the HRC meta-model. This will define a one-to-one mapping between the Polychronous MoCC and the Polychronous Profile of HRC. Symmetrically, a mapping (eventually partial) from HRC to the Polychronous Profile will be developped.

8. Other Grants and Activities

8.1. INRIA associated projects program

Participants: David Berner, Paul Le Guernic, 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 [16], [15].

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), 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'06, IEEE DATE'06 and ACM SAC'06 conferences and for the SLAP'06 workshop.

9.3. Events

- Loïc Besnard animated the session on "Embedded and real-time systems" at AUTRANS'06: 3th INRIA Meeting on Experimental Platforms Engineering, March 2006.

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.
- Julien Ouy taught on computer science at the Graduate program of the University of Rennes I
- Hugo Metivier taught UML design at the DIIC 2 Graduate program, on web design at the MIAGE Graduate program and on computer science at the PCGI Graduate program.

9.5. Visits

- In the frame of the BALBOA associate projects program, Sudipta Kundu (UCSD) and Sandeep Shukla (VT) visited project Espresso in August and November 2006. Jean-Pierre Talpin visited the Fermat Laboratory at VT in March 2006 and UCSD in December 2006.
- Sudipta Kundu (UCSD) visited the team August 2006 in the frame of the BALBOA project
- Sandeep Shukla (VT) visited the team November 2006 in the frame of the BALBOA project

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. GAMATIE, 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] 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.

[8] 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.

[9] 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.

[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° 9, Septembre 1991, p. 1321–1336.

[12] 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° 4, October 2000, p. 347–368.

Year Publications

Articles in refereed journals and book chapters

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

[14] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Synchronous Design of Avionic Applications based on Model Refinements*, in "Journal of Embedded Computing (JEC)", (to appear), 2006.

[15] 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^{o} 8, 2006, p. 1458-1474.

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

Publications in Conferences and Workshops

[17] L. BESNARD, H. MARCHAND, E. RUTTEN. *The Sigali Tool Box Environment*, in "WODES'06, 8th International Workshop on Discrete Event Systems, Sessions on Software Tools for DES, Ann Arbor, Michigan, USA", IEEE Computer Society, July 2006.

[18] C. BRUNETTE, R. DELAMARE, A. GAMATIÉ, T. GAUTIER, J.-P. TALPIN. *A modeling paradigm for integrated modular avionics design*, in "Software Engineering and Advanced Application, IEEE Press", September 2006.

[19] C. BRUNETTE, J.-P. TALPIN, L. BESNARD, T. GAUTIER. *Modeling multi-clocked data-flow programs using the Generic Modeling Environment*, in "Synchronous Languages, Applications, and Programming, Elsevier", March 2006.

[20] A. GAMATI, T. GAUTIER, P. LE GUERNIC. *Toward Static Analysis of SIGNAL Programs using Interval Techniques*, in "Synchronous Languages, Applications, and Programming, SLAP 2006, Vienna, Austria", F. MARANINCHI, M. POUZET (editors). , March 2006.

[21] H. KALLA, J.-P. TALPIN, D. BERNER, L. BESNARD. *Automated translation of C/C++ programs into a synchronous formalism*, in "Engineering of Computer Based Systems, IEEE Press", March 2006.

[22] S. SUHAIB, D. MATHIAKUTTY, S. SHUKLA, J.-P. TALPIN. *Polychronous methodology for system design, a true concurrency approach*, in "High-level design, validation and test workshop, IEEE Press", November 2006.

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

[24] 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.

References in notes

[25] ATLAS GROUP (INRIA & LINA, UNIVERSITÉ DE NANTES). ATL, ATLAS T*ransformation* L*anguage, Reference site*, http://www.sciences.univ-nantes.fr/lina/atl/.

[26] A. AGRAWAL. *Graph Rewriting And Transformation (GReAT) : A Solution for the Model Integrated Computing (MIC) Bottleneck*, in "Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE03)", IEEE Computer Society, 2003, p. 364–368.

[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] 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.

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

[31] 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^{o} 2, 1994, p. 155-182, http://citeseer.ist.psu.edu/buck92ptolemy.html.

[32] J. BÉZIVIN, C. BRUNETTE, R. CHEVREL, F. JOUAULT, I. KURTEV. *Bridging the* Generic Modeling Environment *and the* Eclipse Modeling Framework, in "Proc. of the 4th workshop in Best Practices for Model Driven Software Development, OOPSLA", October 2005.

[33] 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.

[34] F. DEVIN, P. BOULET, J.-L. DEKEYSER, P. MARQUET. *GASPARD A Visual Parallel Programming Environment*, in "International Conference on Parallel Computing in Electrical Engineering (PARELEC'02)", 2002, 145.

[35] 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.

[36] 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.

[37] M. KERBŒUF, 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] A. LEDECZI, M. MAROTI, A. BAKAY, G. KARSAI, J. GARRETT, C. THOMASON, G. NORDSTROM, J. SPRINKLE, P. VOLGYESI. *The Generic Modeling Environment.*, in "Proc. of the IEEE Workshop on Intelligent Signal Processing (WISP'01)", May 2001.

[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^{o} 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] 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.

[42] 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.

[43] I. SMARANDACHE, T. GAUTIER, P. LE GUERNIC. *Validation of Mixed Signal-Alpha Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints.*, in "World Congress on Formal Methods (FM'99), Volume 1709 of LNCS, Toulouse, France", October 1999, p. 1364-1383.

[44] Y. SOREL. *SynDEx: System-Level CAD Software for Optimizing Distributed Real-Time Embedded Systems*, in "Journal ERCIM News", vol. 59, October 2004, p. 68-69.

[45] J.-P. TALPIN, D. POTOP-BUTUCARU, J. OUY, B. CAILLAUD. *From multi-clocked synchronous specifications to latency-insensitive systems*, in "Embedded Software Conference (EMSOFT), ACM Press", 2005.

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