INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Espresso

Environnement de spécification de programmes réactifs synchrones

Rennes

THEME COM

Table of contents

1. Team

Team leader (INRIA)
Jean-Pierre Talpin [Research scientist (DR)]

Administrative assistant
Maryse Auffray [AA INRIA]

Staff members (INRIA)
Thierry Gautier [Research scientist (CR)]
Paul Le Guernic [Research director (DR)]

Staff member (CNRS)
Loïc Besnard [Research Engineer (IR)]

Faculty members
Christian Brunette [Post-Doctorate (INRIA)]
Abdoulaye Gamatié [Lecturer (IFSIC), until Se. 1st.]
Hamoudi Kalla [Post-Doctorate (INRIA)]

Ph. D. students
David Berner [INRIA]
Julien Ouy [INRIA]
Hugo Métivier [IFSIC, starting Oct. 1st.]

2. Overall Objectives

2.1. Introduction

The Espresso project-teamproposes models, methods and tools for computer-aided design of embedded systems.

- The model considered by the project-team is polychrony [8]. 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 [3]. 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 [13] 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 [8] 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 [35] 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.

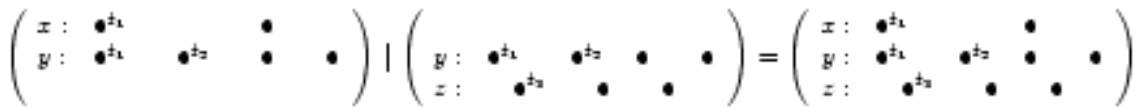


Figure 1. Behavior b over three signals x, y and z in two clock domains

$t_1 < t_2$, 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 $t_3 \neg \leq t_1$.

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.

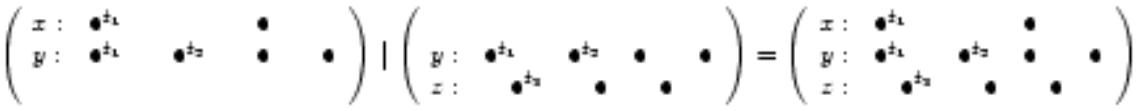


Figure 2. Synchronous composition of $b \in p$ and $c \in q$

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 $t' \neg < t$.

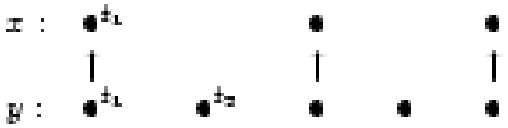


Figure 3. Scheduling relations between simultaneous events

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 the 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 $d \leq c$.

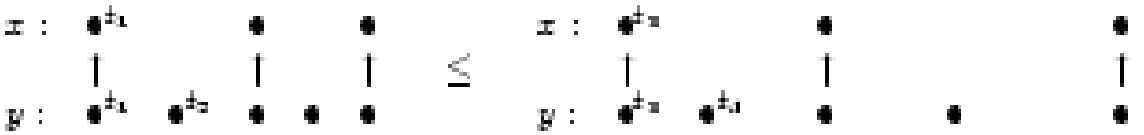


Figure 4. Relating synchronous behaviors by stretching.

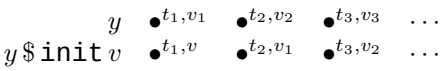
3.1.2. Declarative design languages

Signal [4] 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 analysis

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 $\mathsf{when}\, x$ (resp. $\mathsf{not}\, x$) 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'. Additionally, explicit scheduling relations $x \rightarrow y\, \mathsf{when}\, e$ allow to schedule the calculation of signals (e.g. x after y at the clock e).

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

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

$$
\begin{array}{c}
x := y\ \$\,\mathbf{init}\, v : \hat{}x = \hat{}y \\
x := y\ \mathbf{when}\, z : \hat{}x = \hat{}y\ \mathbf{when}\, z \,|\, y \rightarrow x\ \mathbf{when}\, z \\
x := y\ \mathbf{default}\, z : \hat{}x = \hat{}y \hat{} + \hat{}z \,|\, y \rightarrow x \,|\, z \rightarrow x\ \mathbf{when}\, (\hat{}z \hat{} - \hat{}y)
\end{array}
$$

Figure 5. Clock inference system

3.1.3.2. 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. For instance, 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.

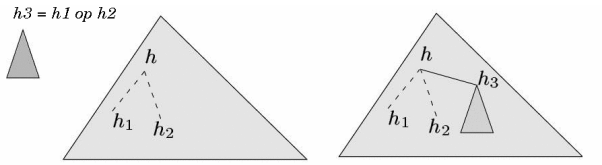


Figure 6.

3.1.3.3. 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;
```

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:

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

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.

3.1.3.4. 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 [48], a co-inductive axiomatization of Signal in the proof assistant Coq [41], based on the calculus of constructions [56], 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 [44], 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 [43].

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 [51], 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).

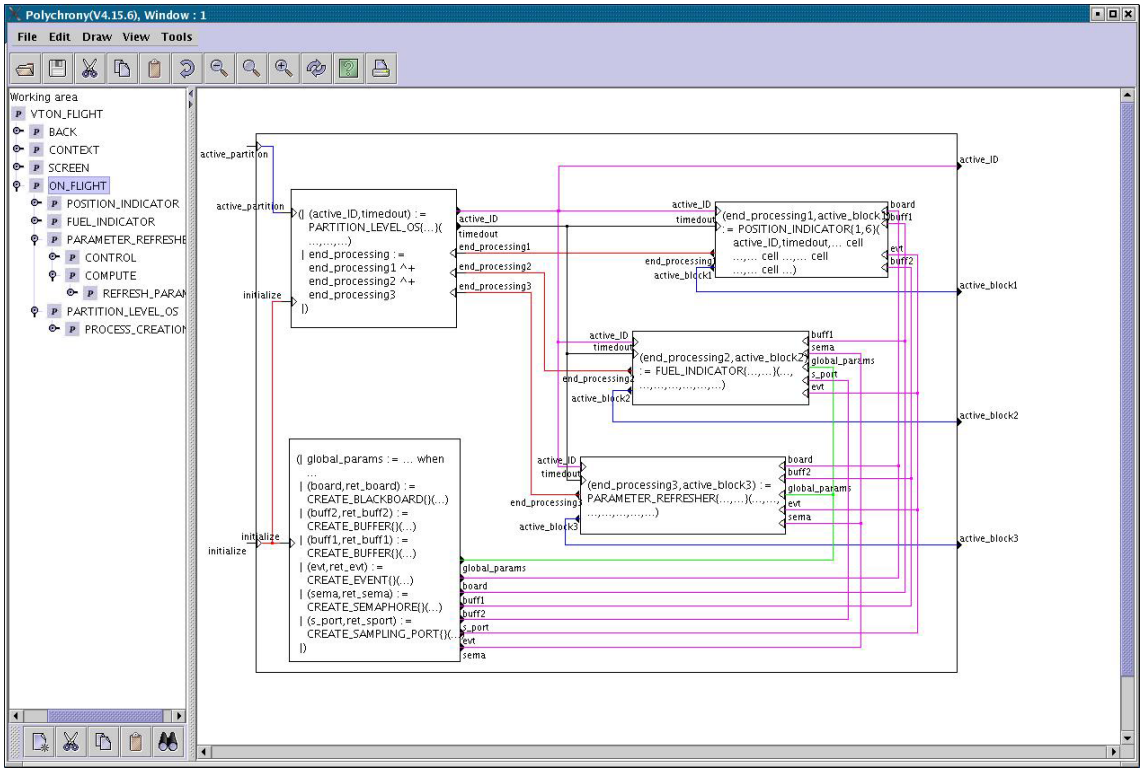


Figure 7. Avionics application modeling using the visual editor of the Polychrony workbench

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. The APEX RTOS library

Participants: Abdoulaye Gamatié, Thierry Gautier.

The Apex interface, defined in the ARINC standard [32], 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, [33]). 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 [12], [21], [29] 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. Signal-Meta, MIMAD, and their Interpreters

Participants: Christian Brunette, Abdoulaye Gamatié.

As detailed in further Sections (6.2, 6.3, and 6.4), we have developed different metamodels in the Generic Modeling Environment (GME):

- Signal-Meta is the metamodel of the SIGNAL language. It describes all syntactic elements specified in [36]: 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. [46] 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 [32], [33].

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.

5.4. SystemCXML: Extracting SystemC structural information

Participant: David Berner.

Structural information of electronic system level (ESL) design projects is important for many applications such as design management and validation. SystemC does not offer any facility to extract or access this information. As for our work in translating SystemC models into the SIGNAL formalism we needed access to this information and, as we did not find any existing tool that conveniently could offer it, we decided to implement it ourselves. The result is a SystemC front-end that parses existing SystemC projects, writes out structural information of the project such as modules, signals, submodules, and ports into an XML file. It also creates an internal data structure with the help of which this information can be conveniently accessed via an API for any further processing and transformation.

The SystemCXML project is conceived as a lightweight and open solution. It avoids doing full fledged parsing of C++ by running the SystemC code through Doxygen, a tool commonly used for the automated generation of code documentation. Doxygen can generate XML output, which is much easier to be parsed than C++. The XML output of Doxygen is then read in with the standard XML parsing library Xerces-C. During this parsing step we extract the information we are interested in and write it into a clean XML file only containing the structure of the SystemC project. In addition to the information about modules, submodules, signals, etc. present in the code, we infer information such as the module hierarchy, the top-level modules, and modules connections. In a further step the XML structural SystemC representation is then read into an internal representation where it is made accessible through an API to client back-end passes such as visualization, test generation, and transformations. We have implemented an example visualization back-end pass visualizing the module hierarchy with the help of the DOT library in merely 60 lines of code [18]. Other than that we have so far used the library for the generation of SIGNAL process skeletons, automated test generation, and reflection and introspection [50].

SystemCXML has been made available as open source in a sourceforge project and can be downloaded from http://systemcxml.sourceforge.net/.

5.5. 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 [48]. 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 [44], the correctness of an implementation of a Signal protocol for loosely timed-triggered architectures [43].

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.

6. New Results

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

Participants: Thierry Gautier, Jean-Pierre Talpin.

The CARROLL Research Programme (see http://www.carroll-research.org), which allows specific collaborations between teams from INRIA, CEA and THALES Research and Technology, has been the right vector to develop the idea of a synchronous UML model, through the PROTES project. This project groups together the Espresso, Aoste and DaRT project-teams from INRIA, the CEA-List, and THALES Communications France, THALES Airborne Systems, THALES Underwater Systems. The aim of the project is to define and standardize at the Object Management Group (OMG) a UML 2.0 profile for real-time embedded systems.

At the OMG Technical Meeting held in Burlingame, CA, USA, in January 2005, was officially voted a new RFP (*Request For Proposals*) invitation for a OMG profile on Real-Time and Embedded systems modeling and analysis (codename: MARTE). The proposal was initiated from two main sources: first, the joint French project PROTES between THALES, INRIA and CEA, aiming at providing modeling elements with a fine understanding of (logical or physical) timing issues at this level; second, a will and a demand from the authors of the previous SPT (schedulability, performance and time) profile to extend it and align it to the more recent UML 2.0 standard. Then several other sources joined in to contribute further requirements, amounting to the current set of objectives for the profile RFP, described below.

Main objectives of MARTE are the following[24]: defining time structures, 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.

MARTE 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 events simultaneity and well-defined priorities).
- 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 (for instance in communication and multimedia treatments).

6.2. SIGNAL Metamodel

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

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

Our aim is to generalize the use of formal methods, and more precisely those proposed by Polychrony. Therefore, such methods must be 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.

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

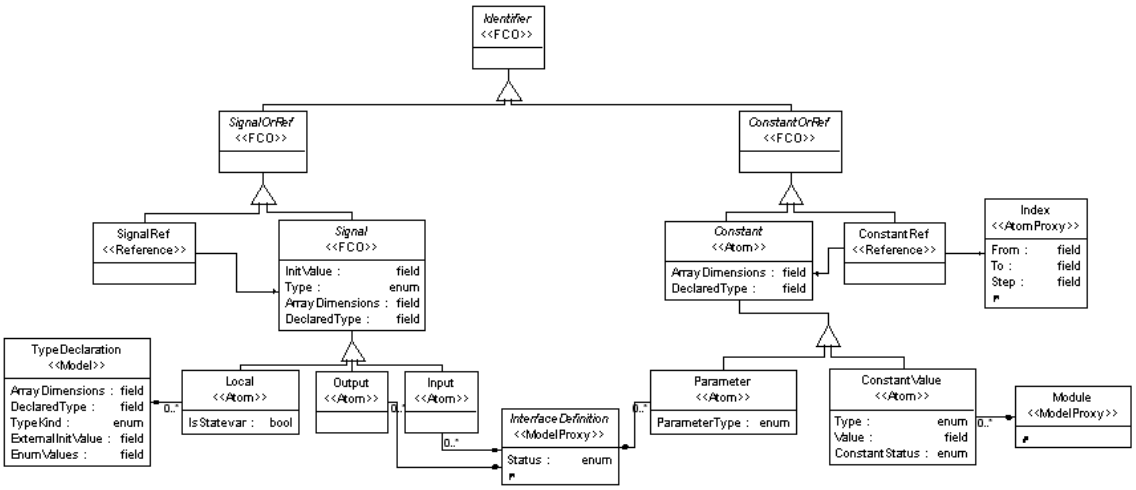


Figure 8. Identifier paradigm sheet of Signal-Meta.

creation of domain-specific modeling and program synthesis environments [45]. 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.

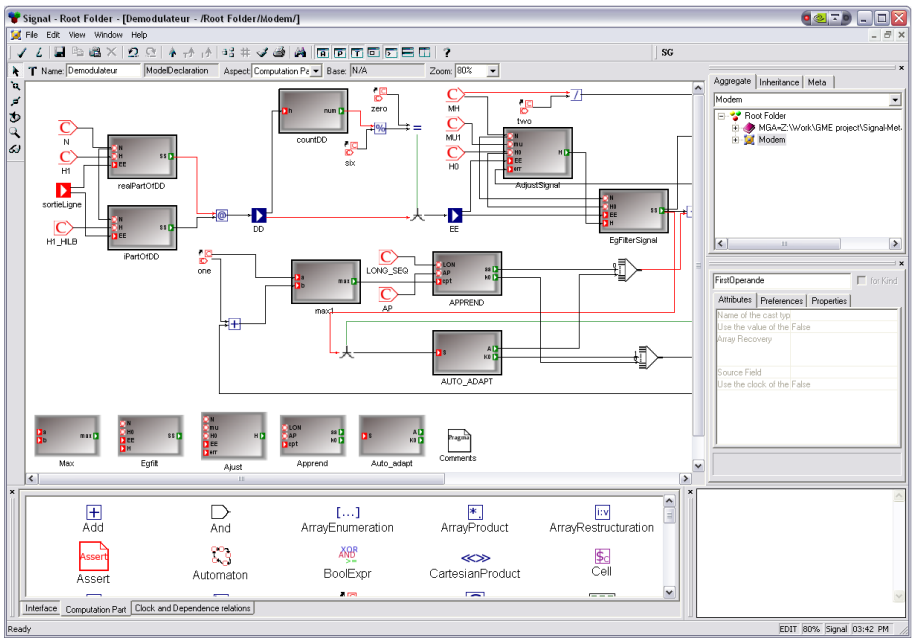


Figure 9. Description of a Modem in GME.

Describing a metamodel in GME consists in modeling all paradigm concepts as classes through usual

UML class diagrams using some predefined UML-stereotypes, as shown in Figure 8. Signal-Meta class diagrams describe all the syntactic elements defined in SIGNAL V4 [36]. Among the described concepts, SIGNAL operators (e.g. arithmetic operators, clock relations) are represented by elementary objects, SIGNAL process frames or modules are represented as GME containers, called Model, and relations between SIGNAL operators, such as definition and dependency, are represented as classes using the Connection stereotype.

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 9 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.

6.2.1. Model Transformation

The graphical description constitutes a good front-end for SIGNAL specifications. To complete this front-end, we need a means to transform the graphical Signal-Meta specifications to 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 is developed in C++ using the *Builder Object Network* (BON2) API provided with GME.

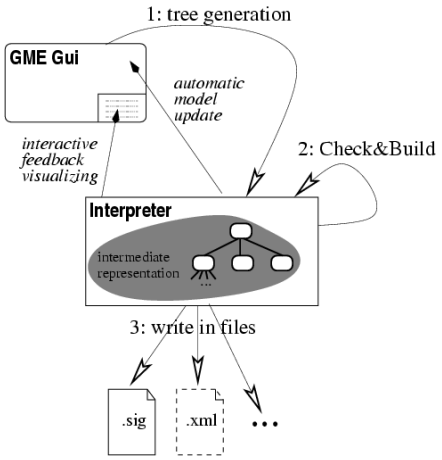


Figure 10. Generation of SIGNAL models from GME.

There are three main steps in the interpretation as shown in Figure 10. The first step consists in generating the structure of the SIGNAL program as a tree in which each node corresponds to a SIGNAL process model, and each leaf to a symbol (signal, constant). The second step consists in building the inner SIGNAL equations of each node of the tree created at the previous step and to detect structural errors in the graphical specifications. Finally, the last step writes the building SIGNAL equations in a file.

6.2.2. Bridge to Eclipse

Within a work initiated at INRIA in the ATLAS team in Nantes, we studied the possible migration of existing GME projects, and particularly Signal-Meta, into 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 [30]. 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 and all OCL constraints 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 [31] transformation language provided with GME can also be replaced by ATL to work on the produced artifacts. This work is described in [19].

6.3. Compositional modeling and transformation of multi-clocked mode automata

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

Participants: Jean-Pierre Talpin, Christian Brunette.

Gathering advantages of declarative and imperative approaches, mode automata were originally proposed by Maraninchi et al. [46] 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 [37] have been proposed until recently, the latest advance being the combination of stream functions with automata in [39]. 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 [52], no attempt has been made to extend mode automata with the capability to model multi-clocked systems and multi-rate systems. Taking advantage of recent works extending Polychrony with a metamodel (see Section 6.2) in the model-driven engineering framework of GME (Generic modeling environment [45]), 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 the switch

To illustrate our modeling techniques, we consider the example of a simple crossbar switch (see Figure 11). 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

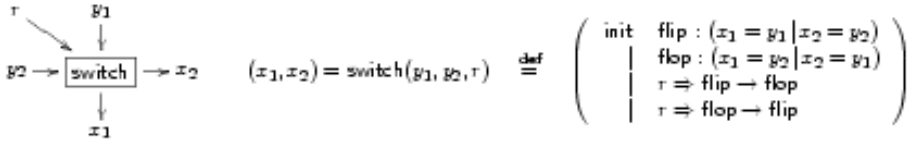


Figure 11. Description of the crossbar switch.

$x_{1,2}$ 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 11).

The left of Figure 12 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 12). The `SwitchAtm` automaton contains two terminal states (`flip` and `flop`). Transitions 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.

Figure 12. The `Switch` process and the `SwitchAtm` mode automaton specifications in GME.

The left of Figure 12 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 13), 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.

Figure 13. Content of states `flip` and `flop`.

6.3.2. Model Transformation

The transformation consists in expressing the graphical formalism as a SIGNAL code. 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 12 and 13.

```
process Switch =
   { type DATA_TYPE; }
   ( ? DATA_TYPE y1, y2; event r;
     ! DATA_TYPE x1, x2; )
   (| min_clock(x1)  |  min_clock(x2)
    | SwitchAtm::(| _SwitchAtm_0_currentState ^= (y1 ^+ y2 ^+ r)
                  | _SwitchAtm_0_reinit := ^0
                  | _SwitchAtm_0_nextState := (#flip when _SwitchAtm_0_reinit)
                                             default (#flop when (r)) when (_SwitchAtm_0_currentState = #flip)
                                             default (#flip when (r)) when (_SwitchAtm_0_currentState = #flop)
                                             default _SwitchAtm_0_currentState
                  | _SwitchAtm_0_currentState := _SwitchAtm_0_nextState$ init #flip
                  | case _SwitchAtm_0_currentState in
                    {#flip}: (| x2 ::= y2 | x1 ::= y1 |)
                    {#flop}: (| x1 ::= y2 | x2 ::= y1 |)
                    end
                  |)
                 where   type _SwitchAtm_0_type = enum(flip, flop);
                         _SwitchAtm_0_type _SwitchAtm_0_currentState, _SwitchAtm_0_nextState;
                         event _SwitchAtm_0_reinit;
                 end
    |)
   where   label SwitchAtm;
   end; % process Switch %
```

6.4. A modeling paradigm for Integrated Modular Avionics design

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

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

We previously addressed the design of applications based on the *Integrated Modular Avionics* (IMA) architecture, which relies on the avionic standard APEX-ARINC [32], [33]. 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) [28]. 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 [45] 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 14. 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

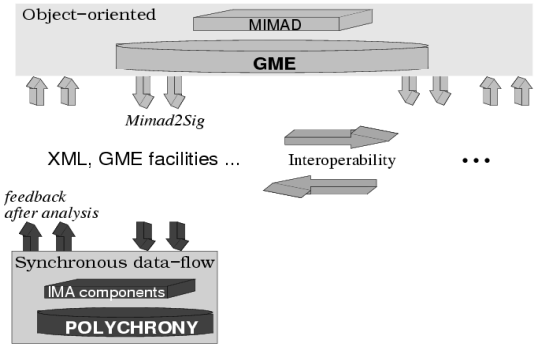


Figure 14. A component-oriented modeling framework for IMA design.

enables to import and export XML files, information exchange between layers can rely on this intermediate format. This favors a high flexibility and interoperability.

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 14) 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 [5]. 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. Dealing with Real-Time Issues within the Polychronous Framework

Keywords: *Polychrony*, *Real-time execution*.

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

We continue our investigations on the use of the *polychronous* model to deal with practical issues in real-time design [22]. In particular, we motivated and illustrated how this model can be used to address the dual notions of abstraction (high level vision) and refinement (moving to the implementation), which are central to decompose, understand and integrate the design of real-time systems. The key points we address include *temporal scalability* (how to describe the deployment of a given component on different execution platforms), *modeling of interrupts within real-time executions*, *characterization of real-time constraints* often imposed to a system (mainly constraints involved by environments and those involved by execution platforms), and *temporal refinement* (based on the over-sampling mechanism of the Signal language). From now, the discussed ideas need to be “materialized” in order to be used by designers in a pragmatic way. Therefore, our future efforts aim at carrying out these ideas within the Polychrony platform.

6.6. A methodology to automatic building of formal models from SystemC description

Keywords: *Formal Methods*, *Signal*, *Synchronous Formalism*, *System Level Design*, *SystemC*.

Participants: Hamoudi Kalla, David Berner, Jean-Pierre Talpin, Loïc Besnard.

Design correctness of software and hardware functionalities of embedded system is one of the major challenges and priorities for designers using software programming languages such as SystemC and C/C++ to describe their systems. These programming languages allow for a comfortable design entry, fast simulation, and software/hardware co-design. Moreover, as the complexity of systems increase, designers are bound to reuse existing Intellectual Property components (IPs) in their design to improve the design productivity. However, system validation is a critical challenge for design reuse based on software programming languages. In recent years, many automated simulator and test tools [49] have been developed to deal with design verification problems. However, mere simulation with non-formal development tools does by no means cover all design errors. What we therefore need is to use formal methods [42] to ensure the quality of system designs. One major problem with formal methods however is the building of the formal system models. This is still considered too complex for a standard design engineer and an error prone and time consuming task. To deal with this problem we propose an approach in which we automatically translate SystemC models into the synchronous formalism Signal, hence enabling the application of formal methods without having to deal with the complex and error prone task to build formal models by hand. We take advantage of the formal nature of Signal to validate system design at different levels of abstraction [15]. The Signal compiler allows static checking for types, dependencies, and clock constraints. Dynamic properties of Signal models can be checked by the model-checking tool, Sigali [47].

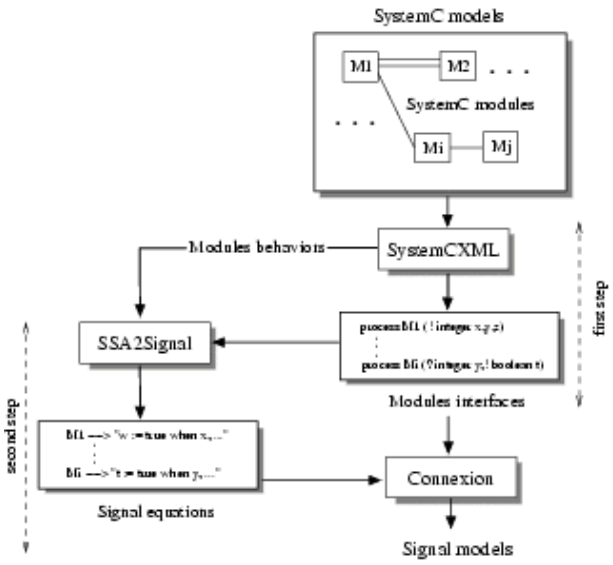


Figure 15. The methodology

Figure 15 shows the methodology of our approach to translate SystemC modules into Signal processes. In a first step we analyze the original model and detect the structural information. The structure of the model is mainly the modular partitioning, the hierarchy within the model and the netlist between the modules. It is important to retain the structure in the resulting formal model, since this allows to formally verify only parts of interest of the design, to abstract certain modules, and to localize errors when they occur. The extraction of the structure is done with a tool called SystemCXML, that has an API, offering the structural information also to other applications such as test generation and visualization. From this design structure, we generate Signal process skeletons, that express the structure of the model in Signal, a formal, synchronous language. This is done with a modified version of GCC that parses the code in question, simplifies the structure, converts it into its intermediate Single Statement Assignement (SSA) form [40] and performs optimization passes such

as dead code elimination on it. We added a transformation pass into GCC that we call SSA2Signal, which generates Signal code that is functionally equivalent to the SystemC input. Once this is done, the Signal process skeletons are filled with the behavior from the translation, and this for each module. The result is a complete formal model of the system in question, obtained in a completely automated fashion.

If an error is detected in the formal model, it has to be corrected directly in the SystemC model. This is because the transformation process is automated and there is no way to go back in the opposite direction. An error found in the formal model therefore still has to be located in the SystemC model, however the preservation of structure of the transformation is helping to localize it.

6.6.1. Extracting and translating SystemC structural information

The SystemCXML tool-flow (Figure 16) consists of three steps. In the first step we process the SystemC code with Doxygen in order to generate an XML output that contains all the information of the original SystemC code, but embedded withing XML tags. While Doxygen is a tool for automated generation of documentation of program code, it has an option for including the source code in the generated output. It can generate several output formats, among them XML, which can be easily read and traversed with the help of any standard XML parsing library. We take advantage of this functionality to approach the difficult problem of parsing C++ code.

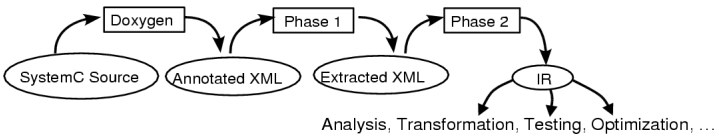


Figure 16. Tool-flow of the SystemCXML project

The second step consists of reading in the Doxygen generated XML data with the help of the Xerces-C++ XML parser [55], extract the structural information of interest, and write this information back into an XML format in a way such that it is readily accessible for other purposes. We represent this extracted information in an Abstract System Level Description (ASLD) XML file.

In the third step, we read in the ASLD and check if it conforms to the DTD description. We process the information and store it in an internal structure that is both, easily accessible and one that closely resembles the structure of SystemC code. As the structure of the IR is the basis for all data manipulations and back-end passes, it is important for it to be generic and the appropriate accessory functions need to be implemented to query the IR. These functions make is possible to traverse the module hierarchy and extract the required structural information. The IR contains classes for all constructs that we extract such as *Inport*, *Outport*, *Signal*, *Sensitivity*, *Process*, and *Module*. Some information that is not readily available in the XML can be obtained by analyzing the available data. The *topmodules* list that points to modules that are not instantiated as a submodule, or the connection class that holds connectivity information are examples for this.

6.6.2. Extracting and translating SystemC behavioral code

The SSA2Signal tool-flow (Figure 17) consists of two steps. The primary main objective SSA2Signal is to extract behavioral code from SystemC modules, and then transforming this code into Signal equations, and Signal processes.

The transformations of SSA2Signal are based on SSA and the GCC compiler. The SSA formalism allows for a smooth translation of SystemC code into Signal equations. The SSA representation can be automatically generated from SystemC modules using the GCC compiler. One of the reasons why we chose to use the SSA form in our approach is that SSA has been adopted as an optimization framework by compilers, such as GCC and the *Java virtual machine Jikes RVM*. This allows an easy use of our approach by designers using a common software programming language to describe their systems. Our SSA2Signal compiler take as inputs

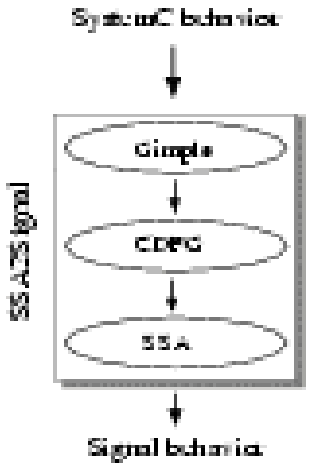


Figure 17. Translating SystemC behavior

the original SystemC modules and their interfaces generated by SystemCXML. It first translates SystemC modules into Gimple, CDFG form and then into SSA form. Next, it generates for each module interface its behavioral code as Signal equations.

One advantage of our translation scheme is that systems modeled using programming languages that are supported by GCC other than C++, such as Java and Fortran, could as well be translated into Signal processes using the same transformation pass.

6.7. Extreme Formal Modeling

Keywords: *Polychrony*, *Real-time execution*.

Participant: David Berner.

The XFM approach is based on the hypothesis that the root of many implementation errors is actually within the specification. Often specifications are not sufficiently clear and contain logical errors that go undetected until very late into the development process. Costs for errors detected late are high, so the idea is to build a clear formal specification with as few errors as possible in order to avoid later debugging costs. This spec can then serve as a golden reference model for later implementation steps.

The intention behind this approach is obvious, however its implementation is not without problems. The development of formal specifications is quite complex and not part of the skillset of the average engineer. Building such a formal model also means to spend more time during specification stage, which can mean to reach a first functional prototype later. The verification of formal specifications can take a lot of time and resources in terms of memory and processor time, and abstractions have to be made for verifications to complete in an acceptable time. However, once the formal specification is complete and verified, it can be used as a starting point for a corresponding implementation. Depending on the languages and environments chosen it may be difficult here to get a transition without semantic rupture, but even if there is a semantic rupture, the formal specification can help to avoid many errors.

For the construction of formal specification models we propose a somewhat different approach. We use agile methods that have been applied with success in the field of Extreme Programming (XP) in order to incrementally construct formal models that verify from the beginning. The models grow with the number of formal properties integrated in a correct by construction (CBC) fashion. We find that using this approach

we are able to build correct formal models faster and that they are better structured, which is facilitating the transition to the implementation.

The journal paper [14] sums up all of our work so far on the subject of XFM. We describe there in detail the different stages of the model building process such as how to describe formal properties and how to iteratively extend the model. It also demonstrates the viability of the approach with several more or less complex examples. There we also elaborate on the fact as to how the modeling order of properties is affecting the modeling result. As we realize that by changing the order in which the formal properties are added the size and state space of the resulting model changes, we study how different ordering schemes influence this behavior. We see if their impact is reproducible for different models and what would be an optimal ordering strategy when aiming for a minimal size, clear structure, and minimal state space of the final model. The paper also presents a graphical interface that can be used to experiment with different orderings for the modeling of formal properties.

6.7.1. Validation of Latency Insensitive Protocols

When modeling complex embedded systems with fast system clocks, there is a limit where certain wires are longer than the signal propagation during one clock cycle. Based on optimistic estimations, a 10 GHz chip in 50 nm technology will contain wires with delays of 10 clock cycles [34]. For such a chip, a strictly synchronous design is not possible any more, we need to introduce multiple clock domains or desynchronizations. The field of latency insensitive protocols [38] is trying to deal with this problem domain. Several protocols have been proposed that - in a more or less automated way - try to eliminate the consequences of long wires without being forced to make drastic changes in the original synchronous components. One common problem of latency insensitive protocols is the proof of correctness of the transformation. Even though the majority of the approaches claim to be correct by construction, few of them deliver formal proofs for the behavior preservation and those that do, are mostly incomplete or difficult to follow.

In [26] we formally verify the preservation of behavior between the original synchronous model and the latency insensitive one. This is done for several different protocols, one of them being our modification of the Carloni implementation that eliminates the need for relay stations. This protocol results in fewer alterations of the design as no blocks are added along the long interconnects; instead, some wires are duplicated in order to make up for the performance loss. In [25] we use functional programming in order to validate different latency insensitive protocols and finally in [54] we resume all our work in this area.

6.7.2. Using Structural Information for Design Validation

The increasing complexity and size of system level design models introduces a difficult challenge for validating them. Hence, in most industries, design validation takes a large percentage of the overall design time. The immediate solution is to automate certain procedures of generating testbenches from the design given certain information about the model. However, the volatile nature of models used for design exploration results in the designer having to alter the testbenches or the automation for the testbenches to reflect the design changes. In efforts to alleviate this problem of constantly changing designs and generating appropriate testbenches for the changed design, we propose a methodology of using structural reflection to extract structural information from design sources allowing the use of tools such as testbench generators and model viewers to seamlessly employ this extracted information. In [17] we present a methodology to automatically extract structural information from already existing SystemC projects and we show how this information can be exploited for system management and validation tasks. We illustrate example uses such as visualization, design management tasks, and automated test generation. As part of this work we implemented an open source tool to extract structural information from SystemC models called SystemCXML mentioned in Section 5.4

6.8. Co-design with data-flow and polyhedral models

Keywords: *affine clocks, polyhedral model.*

Participants: Loïc Besnard, Thierry Gautier.

With Anne Marie Chana, PhD student in the R2D2 project-team (half-time in Rennes, half-time at Yaoundé University, Cameroon), we have started a new cooperation between Espresso and R2D2.

The context is the design of integrated circuits for multimedia applications using jointly data-flow and polyhedral models. The objective is to take benefit of both models in order to optimize systems containing both control aspects and intensive computations. The study relies on two modeling platforms: Polychrony with Signal, and MMAlpha with Alpha.

The former FT project CAIRN [53] can be a first basis for this new study. In this project, we have defined interfacing levels between Signal and Alpha. In Signal-Alpha systems, the refinement of an Alpha program from a functional level to an architectural level oriented toward a particular implementation also induces a refinement of the temporal indices in Signal. The new time indices are obtained through *affine transformations* on the instants of time of the initial Signal specification. An affine clock calculus, complementary to the current boolean clock calculus, has been defined in Irina Smarandache's thesis. This affine clock calculus has been partly integrated this year in Polychrony.

Other paths are being explored for the present study, considering in particular a functional level at which the Alpha program is considered as a node of the Signal graph. Several work directions are envisaged, including scheduling under constraints of the Alpha program (some constraints can be provided by Signal), pipelining of the Signal graph, retiming of the Signal graph...

6.9. Synthesis of GALS architectures

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

Gathering advantages of both the synchronous and asynchronous approaches, the Globally Asynchronous Locally Synchronous (GALS) architectures are emerging as an architecture of choice for implementing complex specifications in both hardware and software. In a GALS system, locally-clocked synchronous components are connected through asynchronous communication lines.

We consider the problem of synthesizing correct-by-construction GALS implementations from modular synchronous specifications. This involves the synthesis of asynchronous wrappers that drive the synchronous clocks of the modules and perform input reading in such a fashion as to preserve, in a certain sense, the global properties of the system. Our approach is based on the weakly endochronous synchronous model, which gives criteria guaranteeing the existence of simple and efficient asynchronous wrappers. We focus on the transformation (by means of added signalling) of the synchronous modules of a multiclock synchronous specification into weakly endochronous modules, for which simple and efficient wrappers exist.

In [16], We propose a process algebraic model to support system design with a formal model of computation and serve as a type system to capture the behavior of system components at the interface level. The proposed algebra is conceptually minimal, equipped with a formal semantics defined in a synchronous model of computation. It supports a scalable notion and a flexible degree of abstraction. We demonstrate its benefits by considering the type-based synthesis of latency-insensitive protocols, showing that the synthesis of component wrappers can be optimized by behavioral information carried by interface type descriptions and yield minimized stalls and maximized throughput.

In [27], we focus on the analysis and the transformation of high-level modular synchronous and multi-clocked specifications (written in languages such as Signal, Lustre, or Esterel) to ensure the weak endochrony and absence of deadlocks. We define a new intermediate representation for synchronous programs, which does not suffer from the state explosion problem of the microstep automata of the weakly endochronous synchronous model (so that real-life systems can be represented and analyzed). At this level, we define symbolic analysis and synthesis algorithms that ensure the needed properties by means of added signalling.

In [23], a survey and discussions are proposed on existing techniques and challenges for the synthesis of GALS architectures starting from multi-clocked synchronous specifications.

6.10. Verification of GALS architectures

Participant: Jean-Pierre Talpin.

Starting with modules described in Signal synchronous programming language, we present in [20] an approach to verification of GALS systems. Since asynchronous parts of a GALS system can not be described in Signal, we use a mixture of synchronous descriptions in Signal and asynchronous descriptions in Promela. Promela is the input language to the SPIN asynchronous model checker. This allows us to achieve globally asynchronous composition (Promela) of locally synchronous components (Signal).

We present three key results:

- First, we present a translation from Signal modules to Promela processes and prove their equivalence.
- Second, we present a technique to abstract a communication bus designed for GALS, the Loosely Time-Triggered Architecture (LTTA) bus, to a finite FIFO channel. The benefit of this abstraction is improved scalability for model checking larger specifications using SPIN.
- Third, we prove the trace equivalence of the model of the GALS system in Promela and a hardware implementation of it. This allows the verification of GALS systems based on the Promela model.

We then use our technique to verify a central locking system for automobiles built on a GALS architecture using the LTTA.

6.11. Toward multi-clocked synchronous stream functions

Participant: Jean-Pierre Talpin.

Functional programming framework both provide the necessary sound semantic framework to formally reason on system modeling, and significantly raise design productivity by offering programming environments in which error-prone and time-consuming engineering tasks such as type consistency and memory management can be automatically handled by the compiler.

However, functional programming frameworks used so far for capturing such models are not capable of expressing multi-clocked computations. The ability to automatically recognize the independence of computation fragments from each other, and therefore the ability to assign distinct clocks to these computations can have both performance implications, as well as other resource scheduling implications.

In [26], we propose a type inference system for representing a synchronous and multi-clocked model of computation in the typed and functional programming language ML. Along the way, we address the issue of performing an automated refinement of implicitly timed stream functions in a model of computation that supports reasoning on partially ordered signal clocks, allowing for formal design transformation and verification to be performed in the context of a functional programming environment.

6.12. New features in Polychrony

Participants: Loïc Besnard, Thierry Gautier.

This year, the affine clock calculus [53] has been partly integrated in Polychrony. It 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 18.

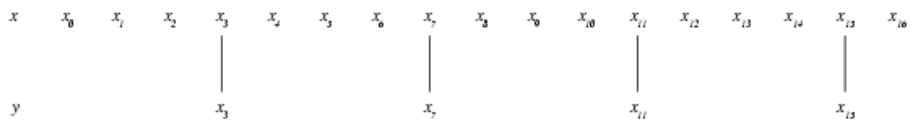


Figure 18. Example: y := affine_sample{3,4} (x)

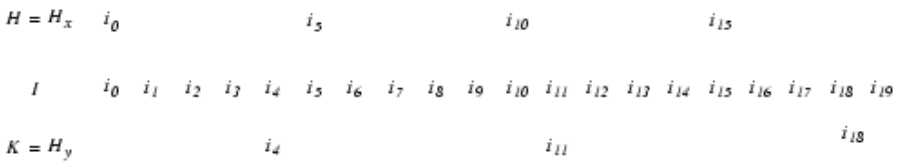


Figure 19. Example: clock_affine{5,4,7}(x,y)

- ***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 19.
- ***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 20.

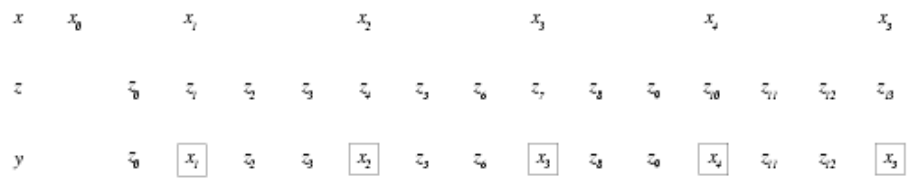


Figure 20. Example: y := clock_unsample{3,1}(x, z)

Concerning the integrated development environment Polychrony 5.1, it has been ported this year to WINDOWS XP OS. So, Polychrony is now available on LINUX, SUNOS, MACOS X and WINDOWS. To make the simulation program portable, the graphical part is written in Java. The predefined graphical library defined in Signal and provided in Polychrony has been extended and many examples have been ported using this graphical library.

A mid-term goal (beginning of 2006) is now to provide an *open-source* diffusion of the environment. To achieve this, we have documented most sources of Polychrony using the system DOXYGEN (http://www.stack.nl/~dimitri/doxygen). This work is performed in parallel with the packaging of the data structures of Polychrony as libraries. The next step will be the definition of the user interface of these libraries. The aim is the use of these APIS for the definition of model transformations described in high-level IDE tools (UML-based, GME "Model-Integrated Computing"...).

Another use of the APIS is the importation of models (SystemC, RTJava...) in Polychrony. In particular, we study the transformation of Gnu SSA to Polychrony. *Static Single Assignment* (SSA) is an intermediate representation of the GCC compiler.

7. Contracts and Grants with Industry

7.1. Carroll project Protes (10/2003-10/2005)

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

The partners of the **CARROLL project Protes** (http://www.carroll-research.org) are Thales, CEA-List and the INRIA project-teams Espresso, Aoste and Dart. The aim of the project Protes is to propose a UML profile for real-time and embedded systems and to defend it before the OMG. The participation of the project-team to this collaboration is addressed 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 URL http://www.artist-embedded.org/FP6) gives a detailed description of the aim and scope of the network.

8. Other Grants and Activities

8.1. INRIA associated projects program

Participants: David Berner, Abdoulaye Gamatié, 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.

Visits performed in the frame of the collaboration brought new contacts to prospective collaborations with Connie Heitmeyer (Naval Research Laboratories) and Ingolf Krueger (UC San Diego) and discussions and proposals being for additional prospective collaborations and exchanges. Applications of Polychrony to embedded system design in automotive are being investigated with Ingolf Krueger and his student Massimiliano Menarini. A collaboration on synchronous modeling is being discussed with Connie Heitmeyer. The partners of the collaboration participate to the proposal REUSSI submitted by INRIA to the IRES program of the NSF.

The Third ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'05) took place at the University of Verona, Italy, from July 11 until July 15, 2005. The conference started on day one with a keynotes speech of Nicolas Halbwachs on "A Synchronous Language at Work: The

Story of LUSTRE". This was followed by presentations throughout the day on the topics of modeling languages, model checking and synthesis and finished with a tutorial on "Making PVS Do What You Want" given by Myla Archer from the Naval Research Laboratory. There were 18 papers accepted for MEMOCODE'05, which made up the core content for the presentations and discussions on the topic areas discussed above, giving the conference a continued reputation of high-quality competitive selection. Acknowledgment of support from the Industry and Academia including INRIA, the ARTIST2 Network of Excellence, BlueSpec Incorporated and Celoxica, was included in printed material. Complete information on the MEMOCODE conference series can be found at http://memocode.irisa.fr.

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.
- Paul Le Guernic and Jean-Pierre Talpin are steering committee members of the ACM-IEEE conference on methods and models for codesign (MEMOCODE).
- Jean-Pierre Talpin is elected member of the evaluation commission at INRIA.
- Jean-Pierre Talpin is external advisory board member of the center of embedded systems at Virginia Tech.
- Jean-Pierre Talpin is organization committee member of the GALS workshop series.

9.2. Conferences

- Jean-Pierre Talpin served as general co-chair of ACM-IEEE MEMOCODE'05 and technical program committee co-chair of its satellite FMGALS'05 workshop. He served as technical program committee member for the IEEE DATE'05 conference, for the embedded system track of the ACM SAC'05 symposium and for the FESCA'05 ETAPS workshop.
- Paul Le Guernic served as technical program committee member of the ACM-IEEE MEMOCODE'04 conference.
- Thierry Gautier served as technical program committee member of SLAP'05 (Synchronous Languages, Applications, and Programming), an ETAPS'05 Satellite Event.

9.3. Events

- Loïc Besnard, Thierry Gautier and Jean-Pierre Talpin organized the 34th. IRISATECH seminar June 22nd. on formal modeling of embedded system design and received 50 participants. Interactive multi-media presentations of the seminar are available on IRISA's website:
  http://www.irisa.fr/videos/irisatech/creaEntreprises/presentation.htm
  The seminar started with the opening remarks by Jean-Loïc Delhaye, director for relations with industry at INRIA-Rennes, and the introduction by Jean-Pierre Talpin, ESPRESSO project-team leader. The first lecture was given by Albert Benveniste, project-team S4, on "Rich components for heterogeneous systems modeling". This lecture was followed by a presentation by Patrick Farail, project leader at AIRBUS Industry, on the TOPCASED initiative on an open-source environment for avionics architecture design. Then, Thierry Gautier et Loïc Besnard, ESPRESSO project-team members, gave an interactive demonstration of the POLYCHRONY workbench. The closing lecture was given by Sandeep Shukla, Deputy Director of the FERMAT Laboratory at Virginia Tech, on "System level design languages and the US CAD industry trend".
- Loïc Besnard and Abdoulaye Gamatié gave a demonstration of Polychrony at the Application of Concurrency to System Design Conference - ACSD'05 (http://acsd2005.irisa.fr)

9.4. Thesis

- Jean-Pierre Talpin served as referee in the thesis defense committee of Matthieu Moy at VERIMAG, December 9.
- Jean-Pierre Talpin served as examiner in the thesis defense committee of Chaker Nakhly at VERIMAG, September 2.

9.5. Teaching

- Thierry Gautier and Loïc Besnard taught on real-time programming at the DIIC 2 Graduate Program of University of Rennes I.
- Abdoulaye Gamatié gave courses as teaching assistant at the University of Rennes I.

9.6. Visits

- David Berner spent five weeks at Virgina Tech starting March 28 in the frame of our collaboration with the INRIA associated team FERMAT lab of Prof. Sandeep Shukla. He collaborated with several of the students there on subjects such as the formal verification of latency insensitive protocols and continuing on the joint work on the extraction and exploitation of structural information from SystemC models.
- Jean-Pierre Talpin visited Virginia Tech in March 2005 and UC San Diego in August 2005 in the frame of the associated projects program.
- Abdoulaye Gamatié visited Virginia Tech in March 2005 in the frame of the associated projects program and gave an invited talk at Virginia Tech on the “Polychronous Modeling of Real-Time Systems”.
- Sandeep Kumar Shukla (Virginia Tech) visited IRISA as Invited Professor of the University of Rennes in July 2005.
- Rajesh Gupta (UC San Diego) visited IRISA in February 2005 in the frame of the associated projects program.
- Hiren Patel and Deepak Mathaikutty (Virginia Tech) visited IRISA in September 2005 in the frame of the associated projects program.

10. Bibliography

Major publications by the team in recent years

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

Articles in refereed journals and book chapters

[12] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Synchronous Design of Avionic Applications based on Model Refinements*, in "Journal of Embedded Computing (JEC), IOS Press", To appear, 2005.

[13] D. POTOP-BUTUCARU, R. DE SIMONE, J.-P. TALPIN. *The synchronous hypothesis and synchronous languages*, in "Embedded Systems Handbook", 2005.

[14] S. SUHAIB, D. MATHAIKUTTY, D. BERNER, S. SHUKLA. *XFM :An Incremental Methodology for Developing Formal Models*, in "ACM Transactions on Design Automation of Electronic Systems (TODAES) Special Issue on Validation of Large Systems", October 2005.

[15] J.-P. TALPIN, P. LE GUERNIC, S. SHUKLA, R. GUPTA. *Compositional behavioral modeling of embedded systems and conformance checking*, in "International Journal on Parallel processing, special issue on testing of embedded systems", 2005.

[16] J.-P. TALPIN, P. LE GUERNIC. *An algebraic theory for behavioral modeling and protocol synthesis in system design*, in "Formal Methods in System Design, Special Issue on formal methods for GALS design", 2005.

Publications in Conferences and Workshops

[17] D. BERNER, H. PATEL, D. MATHAIKUTTY, S. SHUKLA. *Automated Extraction of Structural Information from SystemC-based IP for Validation*, in "Proc. of 6th International Workshop on Microprocessor Test and Verification (MTV'05), Austin Texas, USA", November 2005.

[18] D. BERNER, H. PATEL, D. MATHAIKUTTY, J.-P. TALPIN, S. SHUKLA. *SystemCXML: An Extensible SystemC Front End Using XML*, in "Proceedings of the Forum on specification and design languages (FDL), Lausanne, Switzerland", September 2005.

[19] 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.

[20] F. DOUCET, M. MENARINI, I. KRUGER, J.-P. TALPIN, R. GUPTA. *A verification approach for GALS integration of synchronous components*, in "Proceedings of the International Workshop on Formal Methods for Globally Asynchronous Locally Synchronous Design (FMGALS), Verona, Italy", July 2005.

[21] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Conception Synchrone d'Applications Avioniques par Raffinement de Modèles*, in "Proc. of the 13th International Conference on Real-time Systems (RTS'2005), Paris - France", April 2005.

[22] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Dealing with Real-Time Issues within the Polychronous Framework*, in "Proceedings of the 17th Euromicro Conference on Real Time Systems (ECRTS'05), Work-in-Progress Session, Palma de Mallorca, Balearic Islands - Spain", July 2005.

[23] J. OUY. *A survey of desynchronization in a polychronous model of computation*, in "Proceedings of the International Workshop on Formal Methods for Globally Asynchronous Locally Synchronous Design (FMGALS),

Verona, Italy", July 2005.

[24] L. RIOUX, T. SAUNIER, S. GERARD, A. RADERMACHER, R. DE SIMONE, T. GAUTHIER, Y. SOREL, J. FORGET, J.-L. DEKEYSER, A. CUCCURU, C. DUMOULIN, C. ANDRÈ. *MARTE: A New OMG Profile RFP for the Modeling and Analysis of Real-Time Embedded Systems*, in "DAC 2005 Workshop UML for SoC Design, UML-SoC'05, Anaheim CA, USA", June 2005.

[25] S. SUHAIB, D. BERNER, D. MATHAIKUTTY, J.-P. TALPIN, S. SHUKLA. *A Functional Programming Framework for Latency Insensitive Protocol Validation*, in "Proceedings of the International Workshop on Formal Methods for Globally Asynchronous Locally Synchronous Design (FMGALS), Verona, Italy", July 2005.

[26] S. SUHAIB, D. MATHAIKUTTY, D. BERNER, S. SHUKLA. *Validating Families of Latency Insensitive Protocols*, in "To be published in Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDVT), Napa Valley, California, USA", November 2005.

[27] 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.

Internal Reports

[28] C. BRUNETTE, R. DELAMARE, A. GAMATIÉ, T. GAUTIER, J.-P. TALPIN. *A Modeling Paradigm for Integrated Modular Avionic Design*, Technical report, n° 5715, INRIA, October 2005, http://www.inria.fr/rrrt/rr-5715.html.

[29] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Polychronous Design of Embedded Real-Time Systems*, Technical report, n° 5509, INRIA, March 2005, http://www.inria.fr/rrrt/rr-5509.html.

Bibliography in notes

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

[31] 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.

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

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

[34] L. BENINI, G. D. MICHELI. *Networks on Chips: A New SoC Paradigm*, in "Computer", vol. 35, n° 1, 2002, p. 70–78.

[35] 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.

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

[37] 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.

[38] L. CARLONI, K. MCMILLAN, A. SANGIOVANNI-VINCENTELLI. *The theory of Latency Insensitive design*, in "IEEE Transactions on Computer Aided Design of Integrated Circuits and System", vol. 20, n° 9, 2001, p. 1059-1076.

[39] 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.

[40] R. CYTRON, J. FERRANTE, B. K. ROSEN, M. N. WEGMAN, F. K. ZADECK. *Efficiently computing static single assignment form and the control dependence graph*, in "ACM Trans. on Programming Languages and Systems", vol. 13, n° 4, 1991, p. 451–490.

[41] 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.

[42] A. HALL. *Seven Myths of Formal Methods*, in "IEEE Software", vol. 7, n° 5, September 1990, p. 11–19.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] H. MARCHAND, P. BOURNAI, M. L. 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. 325-346.

[48] 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.

[49] OSCI, SC-HDL, NC-SYSTEMC. *Simulators for SystemC*, http://www.systemc.org/.

[50] H. D. PATEL, D. A. MATHAIKUTTY, D. BERNER, S. K. SHUKLA. *CARH: An Introspective and Service Oriented Architecture for Validation System Level Designs*, in "To be published in IEEE Transactions on CAD (TCAD)".

[51] 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.

[52] 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.

[53] 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.

[54] S. SUHAIB, D. MATHAIKUTTY, D. BERNER, S. SHUKLA. *Validating Families of Latency Insensitive Protocols*, in "To be published in IEEE Transactions on Computers (TCOMP), Special Issue on Simulation-Based Validation", October 2006.

[55] THE APACHE SOFTWARE FOUNDATION. *Xerces C++ validating XML Parser*, http://xml.apache.org/xerces-c/.

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