INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team espresso

Synchronous programming for the trusted component-based engineering of embedded systems and mission-critical systems

Rennes - Bretagne-Atlantique

Theme : Embedded and Real Time Systems

Table of contents

1. Team

Research Scientist

Thierry Gautier [Researcher, INRIA]
Paul Le Guernic [Senior Researcher, INRIA]
Jean-Pierre Talpin [[Team leader, Senior Research, INRIA, HdR]

Technical Staff

Loïc Besnard [Research Engineer, CNRS]

PhD Student

Yann Glouche [INRIA]
Yue Ma [INRIA]
Hugo Métivier [University of Rennes, until August 31st.]

Post-Doctoral Fellow

Christian Brunette [Expert Engineer, INRIA, until March 31st.]
Kenneth Johnson [Post-Doctorate, INRIA, since April 1st.]
Julien Ouy [Expert Engineer, INRIA]
Julio Peralta [Expert Engineer, INRIA]
Huafeng Yu [Expert Engineer, INRIA, since July 1st.]

Visiting Scientist

Sandeep Shukla [Invited Professor, Virginia Tech, until June 1st.]

Administrative Assistant

Stéphanie Lemaile [Secretary, INRIA, since October 1st.]
Lydie Mabil [Secretary, INRIA, until October 1st.]

Other

François Fabre [Junior Engineer, INRIA, since October 1st.]
Vincent Mahé [Expert Engineer, INRIA, since April 1st.]

2. Overall Objectives

2.1. Introduction

The Espresso project-team proposes models, methods and tools for computer-aided design of embedded systems. The model considered by the project-team is polychrony [14]. 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 GeenSys supplies its commercial implementation, RT-Builder, used for industrial scale projects by Snecma/Hispano-Suiza and EADS – Airbus Industries (see http://www.geensys.com).

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 [5]. 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.

2.4. Highlights

Sandeep Shukla, Associate Professor with Virginia Tech, continued his visit in the Espresso project-team at the occasion of his sabbatical. He was jointly funded by the University of Rennes, INRIA Rennes-Bretagne-Atlantique, the Scientific Board of INRIA and the Artist Network of Excellence.

The main objective of the sabbatical was to jointly investigate the state of the art to modeling multi-clocked synchronous embedded systems, as in Polychrony, for instance, and to explore alternatives modeling, analysis and compilation techniques. These discussions resulted in a number of joint publications with INRIA participants to Artist-Design and are subject to several related and ongoing work [41], [18], [32], [24], [28], [27], [39], [31], [22], [37], [21], [23]. The most salient dissemination and publication outcomes resulting of the visit are

- the joint organization of a one-day tutorial at the Design Automation and Test in Europe (DATE) 2009 Conference on "Correct-by-Construction Embedded Software Synthesis: Formal Frameworks, Methodologies, and Tools" (see http://www.date-conference.com/date09/conference/date09-tutorial-C for more details) followed up with the publication of a book with Springer, to appear in 2010.
- the joint organization of the 4th Internation Workshop on the Formal Methods for Globally Asynchronous and Locally Synchronous Design (FMGALS09) as a DATE Friday workshop (see http://www.date-conference.com/date09/conference/date09-workshop-W7 for more details) which brought together researchers from different communities interested in GALS design, and in applying formal methods in creating CAD tools enabling correct by construction GALS design.

Abdoulaye Gamatié, formerly Ph.D. student in the Espresso team, published a book on Signal [2].

3. Scientific Foundations

3.1. Scientific Foundations

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

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

We shall describe the synchronous hypothesis and its mathematical background, together with a range of design techniques enpowered by the approach. Declarative formalisms implementing the synchronous hypothesis can be cast into a model of computation [14] 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 [53] by parameterizing composition with arbitrary timing relations.

3.1.1. A synchronous model of computation

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

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

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

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

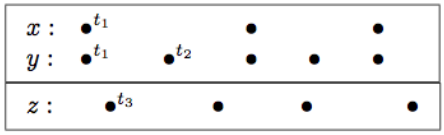

Figure 1. Behavior b over three signals x, y and z in two clock domains

3.1.1.1. Composition

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

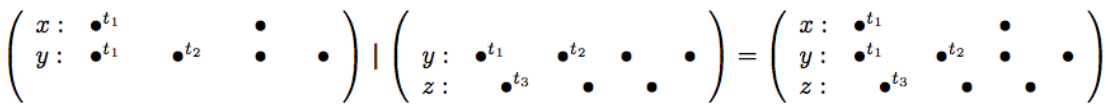

Figure 2. Synchronous composition of $b \in p$ and $c \in q$

3.1.1.2. Scheduling

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

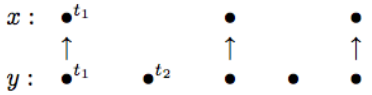

Figure 3. Scheduling relations between simultaneous events

3.1.1.3. Structure

A synchronous structure is defined by a semi-lattice structure to denote behaviors that have the same timing structure. The intuition behind this relation is depicted in Figure 4. It is to consider a signal as an elastic with ordered marks on it (tags). If the elastic is stretched, marks remain in the same relative (partial) order but have more space (time) between each other. The same holds for a set of elastics: a behavior. If elastics are equally stretched, the order between marks is unchanged.

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

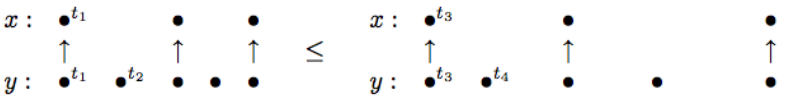

Figure 4. Relating synchronous behaviors by stretching.

3.1.2. A declarative design language

Signal [6] 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}.

$$\begin{array}{rcccc} y & \bullet^{t_1,v_1} & \bullet^{t_2,v_2} & \bullet^{t_3,v_3} & \dots \\ y \,\$\, \texttt{init}\ v & \bullet^{t_1,v} & \bullet^{t_2,v_1} & \bullet^{t_3,v_2} & \dots \end{array}$$

- 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).

$$\begin{array}{rcccc} y & \bullet & \bullet^{t_2,v_2} & \dots \\ & & \downarrow & \\ y\ \texttt{when}\ z & & \bullet^{t_2,v_2} & \dots \\ & & \uparrow & \\ z & \bullet & \bullet^{t_1,0} \quad \bullet^{t_2,1} & \dots \end{array} \qquad \begin{array}{rcccc} y & & \bullet^{t_2,v_2} & \bullet^{t_3,v_3} & \dots \\ & & \downarrow & \downarrow & \\ y\ \texttt{default}\ z & \bullet^{t_1,v_1} & \bullet^{t_2,v_2} & \bullet^{t_3,v_3} & \dots \\ & \uparrow & & & \\ z & \bullet^{t_1,v_1} & & \bullet & \dots \end{array}$$

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

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

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

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

3.1.3. Compilation of Signal

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

3.1.3.1. Synchronization and scheduling specifications

In Signal, the clock $\hat{}x$ of a signal x denotes the set of instants at which the signal x is present. It is represented by a signal that is true when x is present and that is absent otherwise. Clock expressions represent control. The clock `when` x (resp. `when 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 clock expressions e combined using conjunction, disjunction and symmetric difference. Clock equations E are Signal processes: the equation $e\hat{} = e'$ synchronizes the clocks e and e' while $e\hat{} < e'$ specifies the containment of e in e'. Explicit scheduling relations $x \rightarrow y$ `when` e allow to schedule the calculation of signals (e.g. x after y at the clock e).

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

3.1.3.2. Synchronization and scheduling analysis

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

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

Figure 5. Clock inference system

3.1.3.3. Hierarchization

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

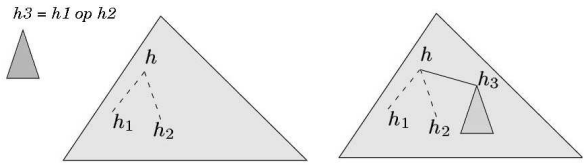

Figure 6. Hierarchization of clocks

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

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, Speeds, and through the national ANR projects Topcased, OpenEmbeDD, Spacify. 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.

5. Software

5.1. The Polychrony workbench

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

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. 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.

Polychrony 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 GeenSys supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects by Snecma/Hispano-Suiza and Airbus Industries (see http://www.geensys.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. 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).
3. GUIs (in Java and on top of Eclipse) with interactive access to compiling functionalities.

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 (http://www-rocq.inria.fr/syndex). Polychrony is open-source software under CECILL-B license packaging its data-structure and algorithms as service libraries. These libraries are used by the Eclipse plugins of the SME modeler (Sec. 5.2).

5.2. Eclipse plugins for Polychrony

Participants: Christian Brunette, Loïc Besnard.

We have developed a metamodel and interactive editor of Polychrony in Eclipse. Signal-Meta is the metamodel of the Signal language. It describes all syntactic elements specified in [55]: 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 to extend the functionality-oriented data-flow paradigm with the capability to model transition systems easily and provide an additional imperative flavor (Sec. 5.4).

These metamodels aim 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 attributes. Using the modeling facilities provided with the Topcased framework, we have created a graphical environment for Polychrony (see figure 7) called SME (Signal-Meta under Eclipse). To hightlight the different parts of the modeling in Signal, we split the modeling of a Signal process in three diagrams: one to model the interface of the process, one to model the computation (or dataflow) part, and one to model all explicit clock relations and dependences. The SME environment is available through teh Espresso update site [47], in the current OpenEmbeDD distribution [46], or in the TopCased v2.0 experimental distribution [48].

5.3. Integrated Modular Avionics design using Polychrony

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

The Apex interface, defined in the ARINC standard [50], 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 [51]). A main feature in an IMA architecture is that several avionics applications (possibly with different critical levels) can be hosted on a single, shared computer system. Of course, a critical issue is to ensure safe allocation of shared computer resources in order to prevent fault propagations from one hosted application to another. This is addressed through a functional partitioning of the applications with respect to available time and memory resources. The allocation unit that results from this decomposition is the *partition*.

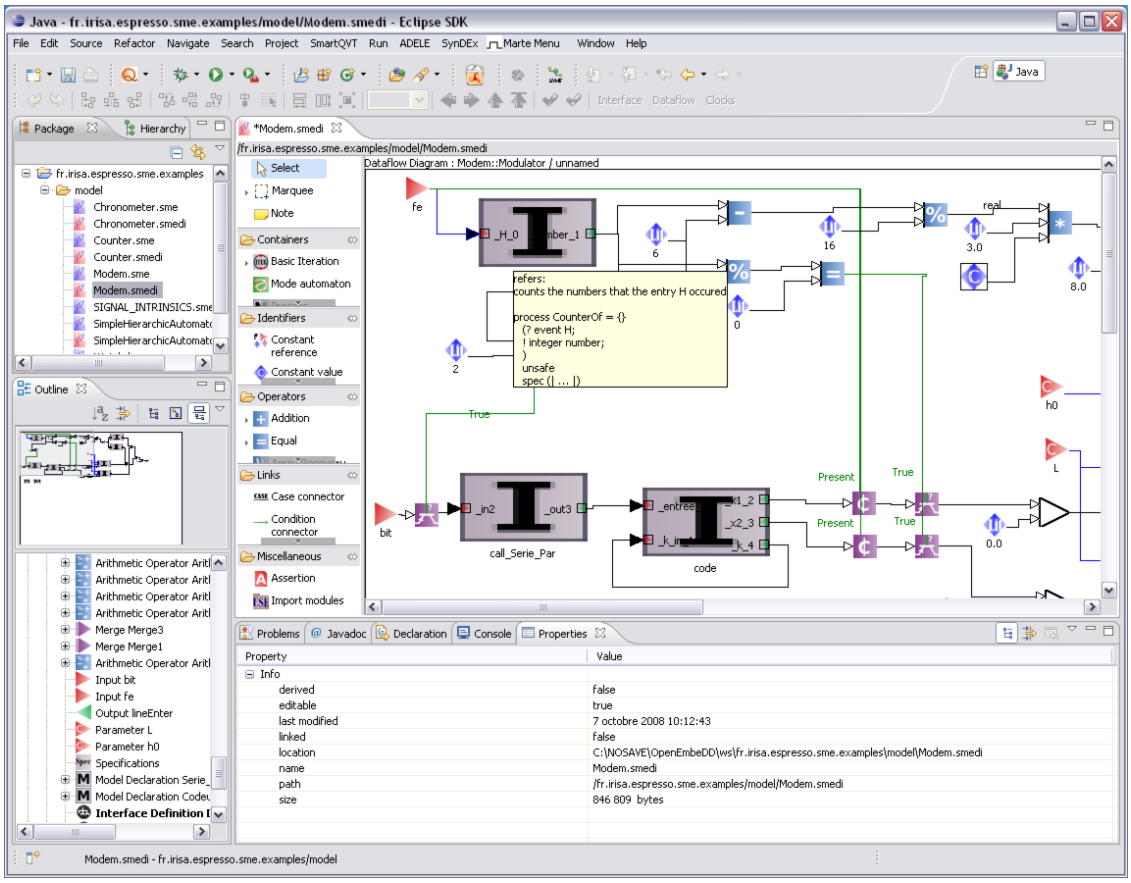

Figure 7. Eclipse SME Environment.

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 [7] is now part of the Polychrony distribution 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.

5.4. Multi-clocked mode automata

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

Gathering advantages of declarative and imperative approaches, mode automata were originally proposed by Maraninchi et al. 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 [56] have been proposed until recently, the latest advance being the combination of stream functions with automata in [57]. 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 [62], no attempt has been made to extend mode automata with the capability to model multi-clocked systems and multi-rate systems. In [63], 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 exposes the semantics and transformation of mode automata in a much simpler way by making use of clearly separated concerns expressed by guarded commands (data-flow relations) and by clock equations (control-flow relations).

6. New Results

6.1. Polychrony and Kahn Process Networks

Participants: Paul Le Guernic, Thierry Gautier.

We have started fundamental yet published research on reconsidering the polychronous model of computation with respect to Kahn Process Networks (KPN).

The first question is: are Signal programs KPNs? While synchronous operators are flow functions, polychronous operators (when, default) are not. But they are "synchronized" flow functions when we add a special value denoting meaningful absence to the domain of values. Then, a result is that a Signal specification denotes a flow function if it can be rewritten, using syntactic equivalence and clock calculus, as the composition of endochronous processes, such that the master clocks are free or input signals.

Considering now the question "are KPN Signal programs?", we propose to extend Signal with an equation allowing to specify pure flow equality (in Signal+). Asynchronous composition can be defined as synchronous composition with flow equality (unbounded FIFOs). Then a synchronized program can be expressed as an unbounded FIFO program, composed with a set of constraints on bounds expressed as Signal equations.

6.2. An algebraic theory of data-flow processing

Participants: Kenneth Johnson, Paul Le Guernic, Jean-Pierre Talpin.

The objective of this work is to define a uniform theoretical framework in the context of the algebraic theory of data for studying polychronous equational systems and their transformations from abstract specifications to real-time implementations on specific architecture. The Signal language is used as a concrete support for these studies.

In the algebraic theory of data, data types are modelled by an algebra consisting of an interface and an implementation. The interface is given by a signature consisting of symbols naming both data and operations on the data. The meaning, or implementation of the signature is given by a many sorted algebra consisting of carrier sets and functions on the sets implementing the data and operations named in the signature.

Our research will study the Signal framework as a data type of streams. A stream is a collection of of data distributed through time. Let the data come from a set V and time be points (or tags) in a set T. We model streams by partial functions assigning the data in V to points in T. Functions on streams are derived from functions on both data and time. Various sets T of tags are being defined depending upon the step in the design process. Morphisms on these sets will be used to define transformations (such as refinement, code distribution,...).

One major investigation is of the expressive power of the Signal framework. We aim to give an analysis of the type of system behaviour that may be expressed in terms of Signal equations. The choice of the Signal data and operations determine the expressiveness of the equations. Our research emphasises the process of choosing new data and new Signal operations. Our choices need not be limited to the discrete domain. We consider cases where data and tags are continuous sets such as the real numbers. Thus, we consider Signal processes operating in continuous time over continuous data.

6.3. New features of Polychrony

Participants: Loïc Besnard, Thierry Gautier.

To facilitate building the Polychrony environment on the various platforms (Linux, Windows, MacOS), we have integrated the use of *cmake* http://www.cmake.org tool. Cmake is a cross-platform build generator. Projects specify their build process with platform-independent files included in each directory of a source tree. Users build a project by generating a build system for a native tool on their platform. This technique is used to export binary distributions.

The team worked on the traceability of program and model transformations in collaboration with Geensys http://www.geensys.com with applications to the simulation embedded spacecraft software. The interoperating tools, the Signal compiler and the Geensys simulator, have been modified in order, for the latter, to reach a variable in the code generated by the former.

Another connection with simulation and visualization tools has been developped to interface the code generator with the IEEE VCD (Value Change Dump) format. During the simulation of an application, a VCD file is generated. It can be used as input by viewers such that *gtkwave* http://gtkwave.sourceforge.net/ or IVI http://sourceforge.net/projects/ivi. The simulation can equally be played step-by-step and interactively by using pipes instead of files.

6.4. Integration of Polychrony in the TopCased platform

Participants: Loïc Besnard, Christian Brunette, Julio Peralta.

The Espresso project-team participates to the TopCased (Toolkit in OPen source for Critical Applications and SystEms Development) project, initiative of Airbus. The integration of a tool, such as Polychrony, in TopCased requires to respect the TopCased quality kit. It consists in the production of the several documents that cover the different steps of the development:

- The software development plan which provides the definition of the context and the objectives, the hypotheses and the constraints, the roadmap, the licence, the project organization, the process of the development, the strategy of verification and validation, the document management, the project management.
- The specification of the software, which consists in the definition of the functional requirements, the operational environment requirements (hardware and software environment), the interface requirements (interface with other tools, Topcased...), the performance requirements.
- The design of the software. It is the definition of the architectural design that describes the hierarchy dependency of software components, the design description that defines the interfaces of software components (API).
- The verification plan. It defines the verification means, the description of the tools used to carry out the verifications, the verification environments (environments used for the processing of the verification), the strategy describing all the types of activities done on the tool: integration tests, validation tests, performance tests, regression tests, tests of reused components, analysis, the description of the verifications: for each verification, the objective, the covered specification requirements, the inputs, the actions and expected results, the name of the file for an automatic verification, and the traceability to verify that all the specification requirements are tested.
- The verification result which consists in the definition of the date of the verification campaign and the verified release, the verification results (verification reference, status of the test), and the justifications for the verifications.
- The software configuration plan which provides the list of the objects (files, components, packages, makefiles, installation scripts...) of the tool and the list of the modifications/evolutions of the tool (release notes).
- The installation and administration guide.
- The user guide.

To respect the Topcased Quality Kit, the verification plan and the verification result have still to be completed for Polychrony. Currently, Polychrony is provided in the TopCased distribution as an experimental tool.

6.5. Verifications of GALS architectures

Participants: Julio Peralta, Thierry Gautier, Loïc Besnard, Jean-Pierre Talpin.

This work is sponsored by project TOPCASED and aims to bridge specifications expressed in a subset of the synchronous SIGNAL language with tools for model checking, with the aim of validating such specifications when composed asynchronously (thus forming so-called GALS architectures).

Our current work aims two languages amenable for model checking: FIACRE and SMV [60]. FIACRE is a language for describing LTS (Labelled Transition Systems), and thus is action-based, whereas SMV is state-based. Also, FIACRE gives rise (through model checkers CADP [58] and TINA [54]) to so-called explicit-state models, while SMV is symbolic-based. And finally, SMV is deemed closer to hardware description than FIACRE since the former semantics assumes a tick while the latter doesn't. For this reason, the translation into FIACRE poses more problems.

On the side of translating SIGNAL to FIACRE we found that our translation was not semantics preserving if we apply it on the source SIGNAL programs, however if we apply the translation after the clock calculus we found out that a particular interpretation (as LTS) of clock synchronization classes renders the translation into FIACRE semantics-preserving for the SIGNAL class of endochronous programs. At the moment we are successfully experimenting on extending our results to so-called poly-endochronous programs. In particular, our current problem boils down to showing that a trace-based reduction of LTS (our FIACRE generated programs) corresponds to the right SIGNAL semantics, and the related question of what temporal properties are preserved through such LTS reduction operation. It is worth noting that the translation here discussed is implemented in the model transformation language ATL, and it is integrated, together with its documentation, into two Eclipse plugins.

On the work of translating SIGNAL programs into SMV, we have better results even though SMV is single-clocked and Signal sources may be multiple-clocked. Additionally, we have succeeded in translating and model-checking multi-clocked SIGNAL programs [34]. These experiments have helped in correcting some translation bugs from SIGNAL to its other model-checker, namely SIGALI. It remains to implement this translation to render it fully automatic, nonetheless our first experiments are promising.

6.6. A contract-based module system

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

Contract-based systems, based on the assume-guarantee paradigm, have become a popular formalism for the modular specification of object-oriented programs. In the context of the development of embedded systems, functional (behavioral) contracts are being applied and become part of mainstream industrial tool. Our goal is to exploit contracts during the developement phase, for instance as a support for early execution (simulation). In this context, we have introduce a new paradigm to express the essence of encapsulation and inheritance in a synchronous and a concurrent modeling framework.

A *contract* is a pair (*assumptions, guarantees*). *Assumptions* describe properties expected by a component to be satisfied by the context in which this component is used; on the opposite *guarantees* describe properties that are satisfied by the component when the context satisfies *assumptions*. We want to provide designers with such a formal model allowing “simple” but powerful and efficient computation on contracts. Thus we define a novel algebraic framework to enable logical reasoning on contracts [29]. It is based on two simple concepts.

First, the assumptions and guarantees of a component are defined as filters: assumptions filter the processes a component may accept and guarantees filter the processes a component provides. A filter is the set of processes, whatever their input and output variables are, that are compatible with some property (or constraint), expressed on the component variables.

Second and foremost, the structure of filters is a Boolean algebra and allows for reasoning on contracts with great flexibility to abstract, refine and combine them. In addition to that, and unlike the related work, the negation of a contract can formally be expressed from within the model. Moreover, contracts are not limited to expressing safety properties, as is the case in most related frameworks, but encompass the expression of liveness properties. This is all again due to the central notion of filter.

We use this algebra to work for the definition of a general purpose module language based on the paradigm of contract described in [30], for a synchronous multi-clocked formalism, SIGNAL, and applied it to the specification of a component-based design process. The paradigm we are putting forward is to regard a contract as the behavioral type of a component and to use it for the elaboration of the functional architecture of a system together with a proof obligation that validates the correctness of assumptions and guarantees made while constructing that architecture.

The module system embedding data-flow equations defined in syntax, has been implemented in OCaml. It produces a proof tree that consists of

- an elaborated Signal program, that hierarchically renders the structure of the system described in the original module expressions,

- a static type assignment, that is sound and complete with respect to the module type inference system,
- a proof obligation consisting of refinement constraints, that are compiled as an observer or a temporal property in Signal.

The property is then tended to SIGNAL's model-checker, Sigali, which allows to prove or disprove that it is satisfied by the generated program. Satisfaction implies that the type assignment and produced SIGNAL program are correct with the initially intended specification. The use of our module system is demonstrated by considering the specification of a protocol for Loosely Time-Triggered Architectures.

This works is presented in the PhD thesis of Yann Glouce [17].

6.7. Virtual prototyping of avionic architecture descriptions

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

In the context of the TopCased project, we are designing and developing a tool for the virtual prototyping of avionic architecture specifications. It aims to interpret specifications expressed in AADL to the synchronous model of computation of Polychrony in order to provide a framework for the simulation, test and verification of integrated modular avionics. In particular, we worked on a translation from AADL programs to SIGNAL programs with tools for simulation and verification.

The SAE AADL is a standard for high level designing and evaluation of the architecture of real-time and embedded systems. The implementation of embedded systems is often distributed across asynchronous communication infrastructures. Such a distributed system is usually composed of locally synchronous processes communicating in a globally asynchronous manner, a GALS system. Yet, in a step-wise refinement based approach, one would prefer to model, simulate and validate such a system in a synchronous programming framework, and then automatically generate its GALS implementation. Our main objective is to perform simulation and validation that take into account both the system architecture and functional aspects. We consider the case where software components are implemented in the synchronous programming language SIGNAL.

First, we use the existing techniques and library of the Polychrony environment, which consist of a model of the APEX-ARINC-653 real-time operating system sevices. A set of rules are defined for these components translations, for example, an AADL processor is translated using the APEX partition-level-OS, and an AADL thread is using the APEX process.

Second, we are experimenting on automatic code distribution starting from system-level AADL specifications using SIGNAL distribution pragmas [32]. we present a methodology to implement such an approach using the polychronous model of computation how to generate distributed simulation code starting from system-level AADL specifications.

Third, we also work on the AADL behavior annex translations. Behavior annex is an extension for the specifications of the actual behaviors. This translation relies on the use of SSA (Static Single Assignment) as intermediate representation of programs. We also define a new library for the non-deterministic behaviors, such as timing actions delay()/computation().

We are currently working on producing a prototype ATL translator from a subset of the AADL metamodel into Signal metamodel, under Eclipse, and some real test cases (such as avionics application examples) will be used for testing. Our goal is to producing an automatic translatior from an AADL model to a SIGNAL model. Future work will focus on the verification and model-checking with some real test cases.

6.8. A compilation tool-chain for Synoptic, a space application DSL

Participants: Julien Ouy, Jean-Pierre Talpin.

In the context of the ANR project Spacify, we are using Polychrony/SME as compilation infrastructure for the domain-specific modeling language Synoptic, defined in the context of the project from industrial user requirements.

We implemented a functional version of the tool to perform model transformation from Synoptic to SME. This transformation is done using the language Kermeta. It is partly described in the Spa4 internal document of the project. The main developments this year have focused on:

- Processing models of blocks and processes: the transformation should provide an equivalence between the Synoptic blocks proposed by the designer and SME synchronous blocks that will be used for implantation, this equivalence should be reflected in the names of the processes, in their interface and the name of their ports, in the use of constants and types.
- Extraction of implicit information: Synoptic models are based on building patterns which require properties on the objects they manipulate: hidden signals, synchrony of signal, chronology of actions. Moreover, the Synoptic design allows the user to provide models partially defined, transformation must add the elements that are necessary if they are implicit and ensure that additions and changes implemented do not betray the thought of user.
- Dataflow transformation: the design of dataflow Synoptic is fairly close to that of Signal, the difficulty occurs mainly in the use of variables: the Synoptic language is far more permissive than SME about synchronous readings - writes to variables. Replication of variable has been set up to ensure the assignment of unique values in each variable in one synchronous moment. Automaton transformation: Synoptic automaton are more complex than those of SME, a major transformation work is needed to pass each other, this work is based on the document (Spa 6) describing the synchronous semantics of Synoptic .
- Traceability of transformations: traceability concern two points, the first is in the transformation from Synoptic to SME, care has been made to names of variables to be able to trace to user design errors found during code generation. The second point concerns the simulation. For simulation, a table is generated to provide RT-builder simulator with equivalency names of variables and processes. So the display of the simulation can be done using the names proposed by the user.

6.9. Virtual prototyping of imperative programs

Participants: Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin.

The Espresso project team has integrated a tool which allows to import C functions into Signal and SME. This work was partly funded by the project FotoVP [25]. We extending this virtual prototyping tool to include the specification and verification of C components using a deterministic thread library: FairThreads.

In C components using the Fairthreads library, one or more schedulers are defined to which threads can be connected. The threads associated with a scheduler are expected to be cooperative and are executed in a strict round-robin fashion. The operation and scheduling of threads connected to a scheduler are completely deterministic.

The importer translating C programs in an SSA representation to Signal specifications will be extended to include C programs using the API commands for the fairthreads. The scope of the project is to consider threaded programs with a bounded number of threads connected to one scheduler. Each thread is a C procedure and is translated using existing methods. A template for the scheduler has been handwritten, and can be algorithmically generated to accommodate any number of threads.

6.10. A simulation infrastructure for CCSL, the timing model of UML MARTE

Participants: Huafeng Yu, Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin.

In the framework of the CESAR project, our work concerns formal analysis and simulation for the design of embedded systems. We are interested in timed systems that can be specified by using clock constraint specification language (CCSL) [52] introduced in the MARTE profile [61]. These systems subject to clock constraints are modeled, specified, analyzed, and simulated within two software environments: TimeSquare [49] and Polychrony. Clock constraints are solved using a heuristic algorithm in TimeSquare, which is generally non-deterministic. Simulation can be carried out and demonstrated in the form of waves. In comparison, Polychrony enables deterministic specifications and formal analysis for the design of safety-critical systems. It is a promising approach to integrate the complementary technologies present in the two software environments for the purpose of system design.

In order to benefit from the advantages provided by Polychrony, the clock constraints specified in CCSL are translated into Signal, therefore, they are analyzed by tools and technologies associated with Polychrony. For instance, the hierarchization technique is used for the clock analysis and affine clock system allows clock synchronization analysis. In addition, the code generated in C or Java by Polychrony enables to obtain deterministic execution traces, compared to the traces obtained by the constraint solver of TimeSquare. Furthermore, some expected properties such as invariance and reachability can be specified so that a controller can be calculated and synthesized (through Sigali) to ensure these properties on target system. Hierarchization extension is also expected so that it can be applied directly on CCSL clock constraints without a Signal translation.

It is also interesting to translate clock relations specified in Signal into CCSL for the simulation purpose as TimeSquare provides a graphical interface for simulation demonstration as well as non-deterministic specifications can be handled. However only Boolean equations are considered in Signal due to CCSL expressivity. The first advantage of this approach is that non-deterministic simulation driven by the constraints solver in TimeSquare can be carried out even if the original Signal program is rejected by the Signal compiler due to the non-determinism issue. The second advantage is that graphical demonstrations of simulation results in the format of waveform are enabled for Polychrony.

6.11. Clock-driven real-time implementation of synchronous specifications

Participants: Dumitru Potop-Butucaru [EPI Aoste], Robert de Simone [EPI Aoste], Yves Sorel [EPI Aoste], Jean-Pierre Talpin.

One important line of work in our project concerns the model-based mapping of the computations and communications of the functional specification onto corresponding resources of the implementation architecture. This mapping comprises both temporal scheduling and spatial allocation aspects. Therefore, we promote an approach which starts from loosely-timed/asynchronous models and proceeds by refining them to fully synchronized ones, using so-called clock calculus techniques under the architecture constraints.

This year, we provided a modeling framework [35] based on an intermediate representation format, called clocked graphs, for polychronous endochronous specifications, which are the ones that can be safely considered for deterministic distributed real-time implementation using static scheduling techniques. Our formalism allows the specification of both ?intrinsic? correctness properties of the specification, such as causality and clock consistency, and ?external? correctness properties, such as endochrony, which ensure compatibility with the desired implementation architecture, including both hardware and software aspects. Using this formalism, we define a new method for distributed real-time implementation of synchronous specification, where the move from (endochronous) synchronous specification to realtime scheduled implementation is a seamless sequence of model decorations.

When compared with current state-of-the-art, represented by the AAA/SynDEx methodology, our approach has the advantage of providing a seamless transformation all the way from specification to implementation models. Our approach also has the advantage of promoting activation conditions (known as clocks) as first class citizens, as opposed to SynDEx, where they can be defined only through structured dataflow constructs (which often results in pessimization of both the specification and the implementation).

6.12. From Concurrent Multiclock Programs to Deterministic Asynchronous Implementations

Participants: Dumitru Potop-Butucaru [EPI Aoste], Robert de Simone [EPI Aoste], Yves Sorel [EPI Aoste], Jean-Pierre Talpin.

Current techniques for the compilation of multi-clock synchronous programs often produce implementations that are over-synchronized. In such implementations, all the clocks (activation conditions) are forced to derive from a single base clock to allow a simple, hierarchical code generation. This approach is well-suited when the target is a sequential processor. For distributed implementations, however, it results in unnecessary inter-processor synchronizations with may result in important performance losses.

We proposed this year a general method to characterize and synthesize correctness-preserving, asynchronous wrappers for synchronous processes on a globally asynchronous locally synchronous (GALS) architecture [36]. Our technique is mathematically founded on the theory of weakly endochronous systems, due to Potop, Caillaud, and Benveniste. Weak endochrony gives a compositional sufficient condition establishing that a concurrent synchronous specification exhibits no behavior where information on the absence of an event is needed. Thus, the synchronous specification can safely be executed with identical results in any asynchronous environment (where absence cannot be sensed). Weak endochrony thus gives a latency-insensitivity and scheduling-independence criterion.

We defined the first general method to check weak endochrony on multi-clock synchronous programs.The ethod is based on the construction of so-called generator sets. Generator sets contain minimal synchronization patterns that characterize all possible reactions of a multi-clocked program. These sets are used to check that a specification is indeed weakly endochronous, in which case they can be used to generate the GALS wrapper. In case the specification is not weakly endochronous, the generators can be used to generate intuitive error messages. Thus, we provide an alternative to classical compilation schemes for multi-clock programs, such as the clock hierarchization techniques sed in Signal/Polychrony.

We are currently working on the application of our technique in the compilation of the Signal language, and on the generation of simpler communication protocols in the SynDEx tool.

6.13. A Megamodel for Cartography of Software Engineering Platforms

Participants: Vincent Mahé, Frédéric Jouault [EPI AtlanMod], Jean Bézivin [EPI AtlanMod], Jean-Pierre Talpin.

Software engineering for real time and critical systems relies on methods and techniques which are embedded in tools which are packaged in multiple platforms like Topcased, Autosar or Simulink. The engineering designing and developing an embedded system must rely on features and techniques partially covered by different engineering platforms which do not collaborate together. We elaborate a prototype of modeling tool able to capture needed information about those platforms and their tools and process the model to manipulate platforms and their features at a higher abstraction level. We built such a model by doing reverse engineering of Eclipse-based platforms and handle extra informations [33].

7. Contracts and Grants with Industry

7.1. ANR project Topcased

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

The Espresso project-team participates to the Topcased initiative of Airbus. The aim of the Topcased initiative is to develop an open-source toolset for the design of avionic architectures. A summary of Topcased appears in [64]. The Topcased project is funded by the ANR and the Midi-Pyrénée region.

7.2. RNTL project Spacify

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

The Espresso project-team participates to the RNTL project Spacify, leaded by CNES and ONERA. Spacify is a research project aiming at developing a design environment for spacecraft flight software.

More precisely, the project shall promote a top-down method based upon multi-clock synchronous modeling, formally-verified transformations, exhaustive verification through model-checking and a runtime framework featuring realtime-friendly distribution and dynamic-reconfiguration services. Furthermore, the various tools shall be released under FLOSS (free/libre/open-source software) licenses, favouring cost-sharing and sustainability.

The project is led by the French Agences CNES and ONERA. It gathers prime contractors Astrium Satellites and Thales Alenia Space, GeenSys (formerly TNI Software) and Anyware Technologies, and academic teams Cama from TELECOM Bretagne, Espresso from INRIA, MV from LaBRI and Acadie from IRIT. It is a 3-years project (starting in February 2006) partly funded by the French Research Agency (ref. ANR 06 TLOG 27).

7.3. ANR project FotoVP

Participants: Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin, Kenneth Johnson.

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

7.4. Fondation EADS

Participants: Yann Glouche, Jean-Pierre Talpin.

The Espresso project-team received a grant from the EADS Foundation to fund a Doctorate on contract-based design in a polychronous model of computation. The aim of this program is to develop a model-driven engineering framework, based on the Eclipse plugins for Polychrony developed in the frame of the ANR project OpenEmbeDD, allowing for the seamless integration of heterogeneous embedded system components within a contract-based design MDD environment.

7.5. Artemisia project CESAR

Participants: Huafeng Yu, Vincent Mahé, Loic Besnard, Thierry Gautier, Jean-Pierre Talpin.

The Espresso project team participates to the European Artemis Joint Undertaking project CESAR, leaded by AVL, with EADS, OFFIS, Airbus, Volvo, Messier-Bugatti, ABB as work packages leaders. CESAR means Cost-Efficient methods and processes for SAfety Relevant embedded systems. The project aims at a stronger integration of safety engineering methods and techniques, all along the phases of the development process. The purpose is to optimize globally the safety critical embedded system architecture by taking in account simulatneously all viewpoints and associated criteria (cost, mass, safety).

7.6. ANR project OpenEmbeDD

Participants: Christian Brunette, Vincent Mahé, Jean-Pierre Talpin.

The final review of the ANR project OPENEMBEDD (http://openembedd.inria.fr) was held at IRISA in March 2009. OpenEmbeDD was acknowledged as a “projet phare” for ANR by the board of experts evaluating the project. The results of the OpenEmbeDD project have been presented at the Neptune’09 workshop (http://neptune.irit.fr/index.php?option=com_content&view=category&layout=blog&id=21&Itemid=10) and will be presented at the forthcoming STIC colloquium (http://colloque-stic.org)

7.7. ITEA2 project OPEES

Participants: Thierry Gautier, Yann Glouche, Yue Ma, Jean-Pierre Talpin.

The mission of OPEES is to build a community able to ensure long-term availability of innovative engineering technologies in the domain of dependable / critical software-intensive embedded systems. Its main objectives are to secure the industrial strategy, improve their competitiveness and develop the European software industry. A general kickoff meetin of the project is expected early 2010. The contribution of team Espresso in OPEES consists of the following:

- Provide the SME/Polychrony toolset as an infrastructure for the co-simulation and co-verification of embedded architectures designed with Geneauto, for its functional aspects, and AADL, for its non-functional aspects. The infrastructure will be evaluated through industry-scale case studies in collaboration with CS and Airbus
- Provide the SME/Polychrony distributed code generation and formal verification functionalities in order to provide means for model-checking automatically generated distributed C code with respect to intermediate representations of Geneauto/AADL specifications in SME/Polychrony.

8. Other Grants and Activities

8.1. National Actions

8.1.1. ARC Triade – Combining models of computation for the design of real-time and embedded applications

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

The Triade Cooperative Research Action (ARC) is a partnership between the AOSTE, DaRT, and ESPRESSO teams of INRIA. Triade aims at using formal models with structuring programmatic constructs as means to translate programs and descriptions written in formalisms widely used in Embedded System and SoC design, and provide a seamless flow of increasingly time-defined and time-accurate models, so as to progressively obtain the final mapped implementation through provably correct steps from the early description elements.

Thanks to travel financing from Triade we have had regular meetings with our colleagues from Rennes (ESPRESSO) and Lille (DaRT). Two publications [36], [35] have resulted, and we intend to launch a common implementation initiative aiming at combining know-how of the various teams.

8.2. European Actions

8.2.1. Network of excellence ARTIST2

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

The Espresso project-team participates to the Artist2 network of excellence. Detailed presentations on the aim and scope of the network can be found in the book [1] and the website http://www.artist-embedded.org/FP6 of the project. In particular, we have contributed to a survey of real-time programming languages edited by Alan Burns [59].

8.3. Research visitors

- With the support of the University of Rennes I, of INRIA, and of the Artist2 Network, Sandeep Shukla (Virginia Tech) is visiting team Espresso from July 2008 until April 2009.
- With the support of the NSF project PEGASUS, Jean-Pierre Talpin visited the Fermat Laboratory at Virginia Tech in October.

9. Dissemination

9.1. Advisory

- Jean-Pierre Talpin is external advisory board member of the center of embedded systems at Virginia Tech, steering committee member of the ACM-IEEE conference on methods and models for code-sign (MEMOCODE), steering committee member of the SLAP++ workshop series, organization committee member of the FMGALS workshop series, and editorial board member of the EURASIP Journal on Embedded Systems.
- Paul Le Guernic animated the working group on software evaluation at INRIA Rennes and was editor for the report made by the national working group, July 2009, animated by Daniel Pilaud. Paul Le Guernic is now a member of the steering comittee for the BIL (base d'information sur les logiciels) whose aim is to implement the conclusions of the report.

9.2. Tutorials

Jean-Pierre Talpin and Sandeep Shukla jointly organized a one day tutorial, on "Correct-by-Construction Embedded Software Synthesis", at the Design Automation and Test in Europe Conference (DATE'09, http://www.date-conference.com/date09/conference/date09-tutorial-C).

9.3. Invited Lectures

Jean-Pierre Talpin gave an invited lecture at the The 7th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES'09, http://pan.vmars.tuwien.ac.at/jtres2009).

9.4. Workshops

Jean-Pierre Talpin and Sandeep Shukla jointly organized the 4th Internation Workshop on the Formal Methods for Globally Asynchronous and Locally Synchronous Design (FMGALS'09) as a DATE Friday workshop http://www.date-conference.com/date09/conference/date09-workshop-W7.

9.5. Conferences

Jean-Pierre Talpin served as technical program committee member for the conferences

ESOP'09 - http://www.cs.york.ac.uk/etaps09

ACSD'09 - http://www.informatik.uni-augsburg.de/acsd

SIES'09 - http://sies2009.epfl.ch

Jean-Pierre Talpin served as Guest Editor for the IEEE Transactions on Computing, Special Section on Science of Design for Safety Critical Systems.

9.6. Teaching

- Thierry Gautier and Loïc Besnard taught on real-time programming at the DIIC 2 Graduate program of the University of Rennes I.
- Thierry Gautier taught on formal methods for component and system synthesis at the Master 2 Graduate program of the University of Rennes 1.
- Hugo Metivier taught programming scientific language at the Licence 1 PCGI Graduate program, taught web design at the Licence 3 MIAGE and taught oriented object language at the DIIC 1 (Licence 3 of Engineering school) Graduate program.
- Yann Glouche taught on functionnal programming and object oriented programming at the STPI Graduate program of the INSA of Rennes. Alse he taught on contrainst programming at the fourth year of the INSA of Rennes.

10. Bibliography

Major publications by the team in recent years

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

[2] A. GAMATIÉ (editor). *Designing Embedded Systems with the SIGNAL Programming Language*, Springer, 2009, http://www.springer.com/engineering/circuits+%26+systems/book/978-1-4419-0940-4.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

[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", 2006.

Year Publications

Doctoral Dissertations and Habilitation Theses

[17] Y. GLOUCHE. *Une méthodologie de spécification et de validation de systèmes hétérogènes fondée sur un modèle de contrats pour la conception des systèmes embarqués*, Université de Rennes 1, 2009, Ph. D. Thesis.

Articles in International Peer-Reviewed Journal

[18] S. AHUJA, S. GURUMANI, C. SPACKMAN, S. SHUKLA. *Hardware Coprocessor Synthesis from an ANSI C Specification*, in "IEEE Design and Test of Computers", 2009, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5209963&isnumber=5209950.

[19] C. BRUNETTE, J.-P. TALPIN, A. GAMATIÉ, T. GAUTIER. *A metamodel for the design of polychronous systems*, in "Journal of Logic and Algebraic Programming", vol. 78, n^{o} 4, April 2009, p. 233–259, http://dx.doi.org/10.1016/j.jlap.2008.11.005.

[20] A. GAMATIÉ, T. GAUTIER. *The Signal Synchronous Multi-Clock Approach to the Design of Distributed Embedded Systems*, in "IEEE Transactions on Parallel and Distributed Systems", 2009, http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.125.

[21] A. SANGIOVANNI-VINCENTELLI, G. YANG, S. SHUKLA, A. MATHAIKUTTY, J. SZTIPANOVITS. *Metamodeling: An Emerging Representation Paradigm for System-Level Design*, in "IEEE Design and Test of Computers", 2009, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5167508&isnumber=5167496.

[22] S. SHUKLA. *Model-Driven Engineering and Safety-Critical Embedded Software*, in "IEEE Computer", 2009, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5233515&isnumber=5233491.

[23] S. SUHAIB, A. MATHAIKUTTY, S. SHUKLA. *A Trace-Based Framework for Verifiable GALS Composition of IPs*, in "IEEE Transactions on Very Large Scale Integration Systems", 2009, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4599241&isnumber=4603036.

International Peer-Reviewed Conference/Proceedings

[24] S. AHUJA, A. MATHAIKUTTY, G. SINGH, J. STETZER, S. SHUKLA, A. DINGANKAR. *Power estimation methodology for a high-level synthesis framework*, in "Quality of Electronic Design", ISQED, 2009, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4810352&isnumber=4810250.

[25] L. BESNARD, T. GAUTIER, M. MOY, J.-P. TALPIN, K. JOHNSON, F. MARANINCHI. *Automatic translation of C/C++ parallel code into synchronous formalism using an SSA intermediate form*, in "Automated Verification of Critical Systems", EASST, 2009.

[26] A. CORTIER, L. BESNARD, J.-P. BODEVEIX, J. BUISSON, F. DAGNAT, M. FILALI, G. GARCIA, T. GAUTIER, J. OUY, M. PANTEL, A. RUGINA, M. STRECKER, J.-P. TALPIN. *Synoptic: a Domain Specific Modeling Language for embedded flight-software (extended abstract)*, in "Formal Methods for Aerospace", Elsevier, November 2009, p. 76–78.

[27] E. DAYLIGHT, S. SHUKLA. *On the Difficulty of Concurrent-System-Design, Illustrated with a 2x2 Switch Case Study*, in "International Symposium on Formal Methods", Springer, 2009.

[28] E. DAYLIGHT, S. SHUKLA, D. SERGIO. *Expressing the Behavior of Three Very Different Concurrent Systems by Using Natural Extensions of Separation Logic*, in "International Workshop on Expressiveness in Concurrency", Elsevier, 2009.

[29] Y. GLOUCHE, P. L. GUERNIC, J.-P. TALPIN, T. GAUTIER. *A Boolean algebra of contracts for assume-guarantee reasoning*, in "Formal Aspects of Component Software", Elsevier, 2009.

[30] Y. GLOUCHE, P. L. GUERNIC, J.-P. TALPIN, T. GAUTIER. *A module language for typing by contracts*, in "NASA Formal Methods Symposium", Springer, 2009.

[31] B. JOSE, B. XUE, S. SHUKLA. *An Analysis of the Composition of Synchronous Systems*, in "International Workshop on the Application of Formal Methods for Globally Asynchronous and Locally Synchronous Design", Elsevier, 2009, http://www.sciencedirect.com/science/article/B75H1-4WXBHBT-6/2/a42f939f6e228137c747dc6d22d44388.

[32] Y. MA, J.-P. TALPIN, S. SHUKLA, T. GAUTIER. *Distributed simulation of AADL specifications in a polychronous model of computation*, in "International Conference on Embedded Software and Systems", IEEE Press, 2009.

[33] V. MAHÉ, F. JOUAULT, H. BRUNELIÈRE. *Megamodeling Software Platforms: Automated Discovery of Usable Cartography from Available Metadata*, in "Reverse Engineering Models from Software Artifacts", IEEE, 2009.

[34] J. PERALTA, T. GAUTIER. *Towards SMV model checking of Signal (multi-clocked) Specifications*, in "Automated Verification of Critical Systems", EASST, 2009.

[35] D. POTOP-BUTUCARU, R. D. SIMONE, Y. SOREL, J.-P. TALPIN. *Clock-driven distributed real-time implementation of endochronous synchronous programs*, in "Embedded Software Conference", ACM Press, 2009.

[36] D. POTOP-BUTUCARU, R. D. SIMONE, Y. SOREL, J.-P. TALPIN. *From Concurrent Multiclock Programs to Deterministic Asynchronous Implementations*, in "Application of Concurrency to System Design", IEEE Press, 2009.

[37] S. SUHAIB, B. JOSE, S. SHUKLA, A. MATHAIKUTTY. *Formal transformation of a KPN specification to a GALS implementation," Specification*, in "Forum on Design Languages", IEEE, 2009, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4641426&isnumber=4641405.

[38] E. VECCHIE, J.-P. TALPIN, K. SCHNEIDER. *Separate compilation and execution of imperative synchronous modules*, in "Design Analysis and Test in Europe", IEEE Press, 2009.

[39] B. XUE, S. SHUKLA. *Modeling and Analyzing the Implementation of Latency-Insensitive Protocols Using the Polychrony Framework*, in "International Workshop on the Application of Formal Methods for Globally Asynchronous and Locally Synchronous Design", Elsevier, 2009, http://www.sciencedirect.com/science/article/B75H1-4WXBHBT-2/2/77a33e1d0047fb19a1306fcf23338ddb.

National Peer-Reviewed Conference/Proceedings

[40] C. ANDRÉ, A. BELAUNDE, B. BERTHOMIEU, C. BRUNETTE, A. CANALS, H. GARAVEL, S. GRAF, F. LANG, V. MAHÉ, M. NAKHLÉ, R. SCHNEKENBURGER, R. DE SIMONE, J.-P. TALPIN, F. VERNADAT. *Présentation des résultats du projet OpenEmbeDD*, in "Neptune, France Paris", P. BAZEX, A. CANALS, T. MILLAN (editors), Revue Génie Logiciel - AFCET, 2009, http://hal.inria.fr/inria-00381639/en/.

Scientific Books (or Scientific Book chapters)

[41] B. JOSE, S. SHUKLA, J.-P. TALPIN. *Programming models for multi-core embedded systems*, Taylor and Francis, 2009.

[42] D. POTOP-BUTUCARU, R. D. SIMONE, J.-P. TALPIN. *The synchronous hypothesis and polychronous languages*, CRC Press, 2009.

Research Reports

[43] L. BESNARD, T. GAUTIER, M. MOY, J.-P. TALPIN, K. JOHNSON, F. MARANINCHI. *Automatic translation of C/C++ parallel code into synchronous formalism using an SSA intermediate form*, INRIA, 2009, http://hal.inria.fr/inria-00400272/en/, RR-6976, Rapport de recherche.

[44] L. BESNARD, T. GAUTIER, J.-P. TALPIN. *Code generation strategies in the Polychrony environment*, INRIA, 2009, http://hal.inria.fr/inria-00372412/en/, RR-6894, Rapport de recherche.

[45] J.-P. TALPIN, J. OUY, T. GAUTIER, L. BESNARD, A. CORTIER. *Modular interpretation of heterogeneous modeling diagrams into synchronous equations using static single assignment*, INRIA, 2009, http://hal.inria.fr/inria-00417682/en/, RR-7036, Rapport de recherche.

References in notes

[46] *OpenEmbeDD website*, 2009, http://openembedd.org.

[47] *Polychrony Update Site for Eclipse plug-ins*, 2009, http://www.irisa.fr/espresso/Polychrony/update/.

[48] *TopCased website*, 2009, http://www.topcased.org.

[49] INRIA AOSTE TEAM (editor). *TimeSquare*, 2009, http://www-sop.inria.fr/aoste/dev/time_square/.

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

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

[52] C. ANDRÉ, F. MALLET, R. DE SIMONE. *Modeling Time(s)*, in "ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS/UML'07), TN, USA", LNCS 4735, Springer, October 2007, p. 559–573.

[53] 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.

[54] B. BERTHOMIEU, P.-O. RIBET, F. VERNADAT. *The tool TINA - construction of abstract state spaces for Petri Nets and Time Petri Nets*, in "International Journal of Production Research", vol. 42, n^o 14, 2004.

[55] L. BESNARD, T. GAUTIER, P. LE GUERNIC. *SIGNAL V4-INRIA version: Reference Manual*, 2009, http://www.irisa.fr/espresso/Polychrony.

[56] J. BUCK, S. HA, E. A. LEE, D. G. MESSERSCHMITT. *Ptolemy: A Framework for Simulating and Prototyping Heterogenous Systems*, in "Int. Journal in Computer Simulation", vol. 4, n^o 2, 1994, p. 155-182.

[57] 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.

[58] H. GARAVEL, F. LANG, R. MATEESCU, W. SERWE. *CADP 2006: A Toolbox for the construction and Analisys of Distributed Processes*, in "Proceedings of the 19th International Conference on Computer Aided Verification, Springer LNCS", vol. 4590, 2007.

[59] T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Polychronous Design of Real-Time Applications with SIGNAL*, in "ARTIST Survey of Programming Languages", A. BURNS (editor), 2008, http://www.artist-embedded.org/artist/ARTIST-Survey-of-Programming.html.

[60] K. L. MCMILLAN. *Symbolic Model Checking: An approach to the state explosion problem*, Carnegie Mellon University, May 1992, Ph. D. Thesis.

[61] OBJECT MANAGEMENT GROUP (OMG). *Modeling and Analysis of Real-time and Embedded systems (MARTE), Beta 2*, June 2008, http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf.

[62] 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.

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

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