IN PARTNERSHIP WITH:
CNRS

Université Rennes 1

Activity Report 2012

Project-Team ESPRESSO

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

IN COLLABORATION WITH: Institut de recherche en informatique et systèmes aléatoires (IRISA)

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Embedded and Real Time Systems

Table of contents

Project-Team ESPRESSO

Keywords: Synchronous Languages, Embedded Systems, Formal Methods, Model-Driven Engineering, Software Engineering, Compiling

Creation of the Project-Team: January 01, 2002 , *Updated into Team:* January 01, 2013 .

1. Members

Research Scientists

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

Engineer

Loïc Besnard [Research Engineer, CNRS]

PhD Students

Adnan Bouakaz [University of Rennes 1]
Van-Chan Ngo [Inria]
Ke Sun [Inria]

Post-Doctoral Fellows

Christophe Junke [Expert Engineer, Inria]
Yue Ma [Expert Engineer, Inria]
An Phung-Khac [Expert Engineer, Inria, until Sep. 30th.]
Huafeng Yu [Expert Engineer, Inria]

Administrative Assistant

Stéphanie Lemaile [Secretary, Inria]

2. Overall Objectives

2.1. Introduction

The ESPRESSO project-team is interested in the model-based computer-aided design of embedded-software architectures using formal methods provided with the polychronous model of computation [8]. ESPRESSO focuses on the system-level modeling and validation of software architecture, during which formal design and validation technologies can be most benefitial to users in helping to explore key design choices and validate preliminary user requirements. The research carried out in the project team covers all the necessary aspects of system-level design by providing a framework called Polychrony. The company Geensoft (now part of Dassault Systems), with which we had many collaborations, has supplied a commercial implementation of Polychrony, RT-Builder (see http://www.geensoft.com), which has been deployed on large-scale applications with the avionics and automotive industries.

Polychrony is a computer-aided design toolset that implements the best-suited GALS (globally asynchronous and locally synchronous) model of computation and communication to semantically capture embedded architectures. It provides a representation of this model of computation through an Eclipse environment to facilitate its use and inter-operation with the heterogeneity of languages and diagrams commonly used in the targeted application domains: aerospace and automotive. The core of Polychrony provides a wide range of analysis, transformation, verification and synthesis services to assist the engineer with the necessary tasks leading to the simulation, test, verification and code-generation for software architectures, while providing guaranteed assurance of traceability and formal correctness. The Polychrony toolset is available under EPL and GPL v2.0 license by Inria.

2.2. Context and motivations

The design of embedded software from multiple views and with heterogeneous formalisms is an ubiquitous practice in the avionics and automotive domains. It is more than common to utilize different high-level modeling standards for specifying the structure, the hardware and the software components of an embedded system.

Providing a high-level view of the system (a system-level view) from its composite models is a necessary but difficult task, allowing to analyze and validate global design choices as early as possible in the system design flow. Using formal methods at this stage of design requires one to define the suited system-level view in a model of computation and communication (MoCC) which has the mathematical capability to cross (abstract or refine) the algebraic boundaries of the specific MoCCs used by each of its constituents: synchronous and asynchronous models of communication; discrete and continuous models of time.

We believe these requirements to be met with the polychronous model of computation. Historically related to the synchronous programming paradigm (Esterel, Lustre), the polychronous model of computation, implemented with the data-flow language Signal and its Eclipse environment Polychrony, stands apart by the capability to model multi-clocked systems. This feature has, in turn, been proved and developed as one ability to compositionally describe high-level abstractions of GALS architectures.

The research and development performed in the team aim at completely exploiting this singularity and to implement its practical implications in order to provide the community with all benefits gained from this property of compositionality.

Our main research results are, first and foremost, to consolidate the unique capability of the polychronous model of computation to provide a compositional design mathematical framework with formal analysis and modular code generation techniques implementing true compositionality (i.e., without a global synchronization artifact as with most synchronous modeling environments) [42], [2], [13].

The most effective demonstrations of these features are found in our recent collaborative projects Spacify, Opees and Cesar to equip industrial toolsets with architecture/functions co-modeling services and provide flexible and modular code generation services.

Our research perspectives aim at pursuing the research, dissemination, collaboration and technology transfer results obtained by the team over the past years and, in doing so, further exploit the singularity and benefits of our model of computation and maximize its impact on the academic and industrial community.

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 [1]. This affinity explains the ease of generating systems and architectures and verifying 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 of the Year

Polarsys is an Industry Working Group focusing on open source tools for the development of embedded systems. Polychrony was used to define the Tool Quality Assurance Plan of the Polarsys platform according to the DO-178B and DO-178C certification standards. Polychrony has been integrated in the experimental Polarsys platform.

Jean-Pierre Talpin received the ACM/IEEE LICS Test of Time Award for his paper “A type and effect discipline”, with his co-author Pierre Jouvelot.

3. Scientific Foundations

3.1. Introduction

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 [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 [32] can be superimposed on this model by considering a flow equivalence relation as well as heterogeneous systems [33] by parameterizing composition with arbitrary timing relations.

3.2. Polychronous 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 domain 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.

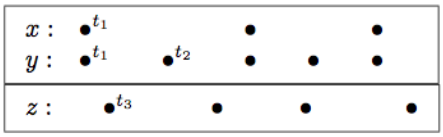

Figure 1. Behavior b over three signals x, y and z in two clock domains

3.2.1. 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.

3.2.2. 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 $t' \neg < t$.

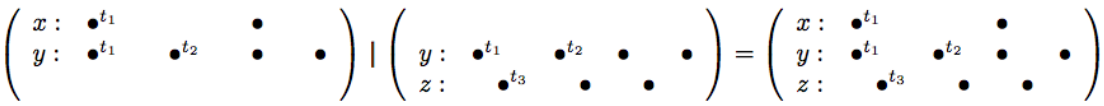

Figure 2. Synchronous composition of $b \in p$ and $c \in q$

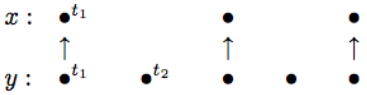

Figure 3. Scheduling relations between simultaneous events

3.2.3. 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 $d \leq c$.

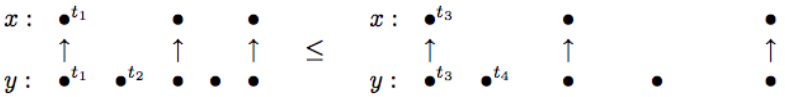

Figure 4. Relating synchronous behaviors by stretching.

3.3. A declarative design language

Signal [34], [48], [49], [41] 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 $y_{t_2} \rightarrow x_{t_2} \longleftarrow z_{t_2}$.
- 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).

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

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 ^= value`.

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

3.4. 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.

3.4.1. 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 $\hat{}0$ 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).

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

3.4.2. 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

3.4.3. Hierarchization

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

To determine the order $x \preceq y$ in which signals are processed during the period of a reaction, clock relations E play an essential role. The process of determining this order is called hierarchization and consists of an insertion algorithm which hooks elementary control flow graphs (in the form of if-then-else structures) one to the others. Figure 6, right, let h3 be a clock computed using h1 and h2. Let h be the head of a tree from which h1 and h2 are computed (an if-then-else), h3 is computed after h1 and h2 and placed under h [28].

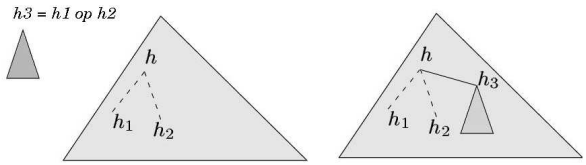

Figure 6. Hierarchization of clocks

4. Application Domains

4.1. Embedded systems

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.

Along the way, the project adopted the policy proposed with project Topcased and continued with OpenEmbeDD to make its developments available to a large community in open-source. The Polychrony environment is now integrated in the OPEES/Polarsys platform and distributed under EPL and GPL v2.0 license for the benefits of a growing community of users and contributors, among which the most active are Virginia Tech's Fermat laboratory and Inria's project-teams Aoste, Dart.

5. Software

5.1. The Polychrony toolset and its hypertext source documentation

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

The Polychrony toolset is an Open Source development environment for critical/embedded systems. It is based on Signal, a real-time polychronous data-flow language. It provides a unified model-driven environment to perform 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. It can be included in heterogeneous design systems with various input formalisms and output languages.

The Polychrony toolset provides a formal framework:

- to validate a design at different levels, by the way of formal verification and/or simulation,
- to refine descriptions in a top-down approach,
- to abstract properties needed for black-box composition,
- to assemble heterogeneous predefined components (bottom-up with COTS),
- to generate executable code for various architectures.

The Polychrony toolset contains three main components and an experimental interface to GNU Compiler Collection (GCC):

- The Signal toolbox, a batch compiler for the Signal language, and a structured API that provides a set of program transformations. The Signal toolbox can be installed without other components. The Signal toolbox is distributed under GPL V2 license.
- The Signal GUI, a Graphical User Interface to the Signal toolbox (editor + interactive access to compiling functionalities). The Signal GUI is distributed under GPL V2 license.
- The SME/SSME platform, a front-end to the Signal toolbox in the Eclipse environment. The SME/SSME platform is distributed under EPL license.
- GCCst, a back-end to GCC that generates Signal programs (not yet available for download).

The Polychrony toolset also provides:

- libraries of Signal programs,
- a set of Signal program examples,
- user oriented and implementation documentations,
- facilities to generate new versions.

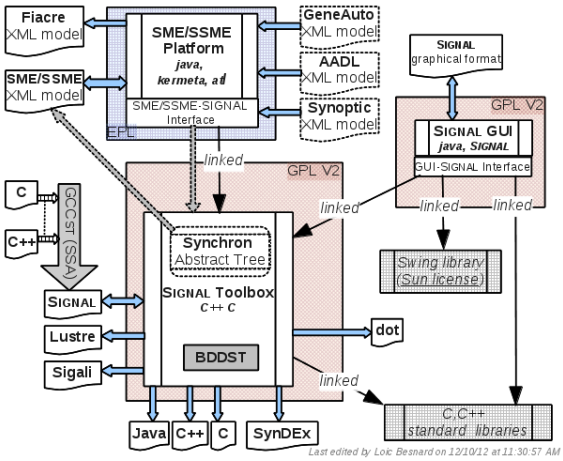

Figure 7. The Polychrony toolset high-level architecture

The Polychrony toolset can be freely downloaded on the following web sites:

- The Polychrony toolset public web site: http://www.irisa.fr/espresso/Polychrony. This site, intended for users and for developers, contains downloadable executable and source versions of the software for differents platforms, user documentation, examples, libraries, scientific publications and implementation documentation. In particular, this is the site for the new open-source distribution of Polychrony.
- The Inria GForge: https://gforge.inria.fr. This site, intended for internal developers, contains the whole sources of the environment and their documentation.
- The TOPCASED distribution site: http://www.topcased.org. This site provides the current reference version of the SSME platform, including the executable of the Signal toolbox.

The Polychrony toolset currently runs on Linux, MacOS and Windows systems.

The Geensoft company, now part of Dassault Systèmes, supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects (see http://www.geensoft.com).

As part of its open-source release, the Polychrony toolset not only comprises source code libraries but also an important corpus of structured documentation, whose aim is not only to document each functionality and service, but also to help a potential developer to package a subset of these functionalities and services, and adapt them to developing a new application-specific tool: a new language front-end, a new back-end compiler. This multi-scale, multi-purpose documentation aims to provide different views of the software, from a high-level structural view to low-level descriptions of basic modules. It supports a distribution of the software "by apartment" (a functionality or a set of functionalities) intended for developers who would only be interested by part of the services of the toolset.

A high-level architectural view of the Polychrony toolset is given in Figure 7.

5.2. The Eclipse interface

Participants: Loïc Besnard, Yue Ma, Huafeng Yu.

Meta-modeling, Eclipse, Ecore, Signal, Model transformation

We have developed a meta-model and interactive editor of Polychrony in Eclipse. Signal-Meta is the meta-model of the Signal language implemented with Eclipse/Ecore. It describes all syntactic elements specified in [35]: all Signal operators (e.g. arithmetic, clock synchronization), model (e.g. process frame, module), and construction (e.g. iteration, type declaration).

The meta-model primarily aims at making the language and services of the Polychrony environment available to inter-operation and composition with other components (e.g. AADL, Simulink, GeneAuto) within an Eclipse-based development toolchain. Polychrony now comprises the capability to directly import and export Ecore models instead of textual Signal programs, in order to facilitate interaction between components within such a toolchain.

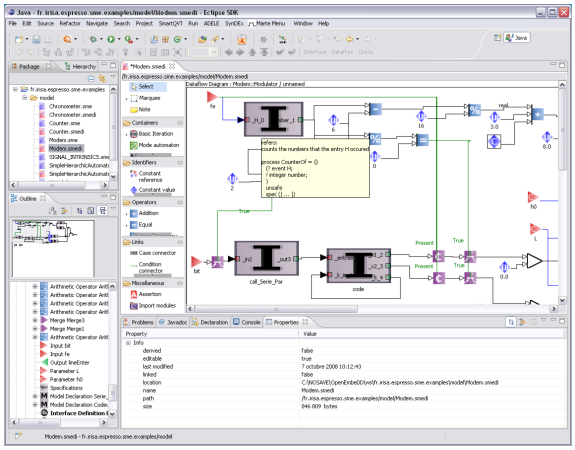

Figure 8. Eclipse SME Environment.

It also provides a graphical modeling framework allowing to design applications using a component-based approach. Application architectures can be easily described by just selecting 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 8) called SME (Signal-Meta under Eclipse). To highlight 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 data-flow) part, and one to model all explicit clock relations and dependences. The SME environment is available through the ESPRESSO update site [23], in the current OpenEmbeDD distribution [22], or in the TopCased distribution [25]. Note that a new meta-model of Signal, called SSME (Syntactic Signal-Meta under Eclipse), closer to the Signal abstract syntax, has been defined and integrated in the Polychrony toolset.

5.3. Integrated Modular Avionics design using Polychrony

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

The Apex interface, defined in the ARINC standard [26], 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 [27]). 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 [5] 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.

6. New Results

6.1. Extensions of the Signal language and the Polychrony formal model

Participants: Thierry Gautier, Paul Le Guernic.

The different works related to the use of the polychronous model as semantic median model (which has also a syntactic instance) for different effective models (AADL [15], Simulink via GeneAuto, UML via CCSL...) lead us to study various possible extensions of the semantic model as well as the syntactic one.

Thus, we are defining a new version, V5, of Signal, that will be a deep evolution from the current V4 version.

In particular, we are investigating the way state diagrams are best represented in the polychronous model of computation, maintaining the multi-clock characteristic property of the representation. We propose a semantic model for these automata, that relies on the Boolean algebra of clocks. A special case of automata is those that may be represented as regular clock expressions, for which we develop a specific formal calculus. These regular expressions may be used as a powerful manner to express regular dynamic properties of polychronous processes. In correspondence with these models, we are defining syntactic structures to represent these Signal State Diagrams.

Moreover, an important challenge we want to address in the next few years is that of providing design automation techniques and tools for engineering heterogeneous cyber-physical systems (CPS). This leads in particular to new requirements related to the language itself in which we want to describe such software architectures. With respect to the current V4 version of Signal, the basic idea is to extend Signal with a syntactic structure that encapsulates a polychronous process P in a *system*, S, that could have a continuous temporal domain providing a real-time clock presented in some time unit (*fs*, ..., *ms*, ..., *sec*, *mn*, ...). Such a real-time clock can be used as a usual signal in the process P encapsulated in S. Systems S_1, ..., S_n may be composed (with the standard composition of Signal) in a same system S, but the *ms* of a given system S_i is a priori not synchronous with the *ms* of another system S_j. Then it is possible to specify constraints in the system S on these different signals, to express for instance some variation limits of different clocks.

For that purpose, we have defined a new taxonomy of polychronous processes to characterize precisely the following classes: system, task, (logical) process, function, reaction, diagram, observer, controller... This characterization is based on properties such as time reference, input-output clock relations, input-output dependences, determinism, exo/endochrony. For example, a system is either a physical system abstraction, or a basic system, or a system of systems. A basic system has a unique continuous time reference; it provides an internal actual discrete time unit subset of its external continuous time, shared by all its components. As another example, for a subclass of logical processes: a function is a deterministic, inout clocked, endochronous and atomic process that denotes a mathematical flow function. All these different semantic classes are provided syntactic counterparts in the new Signal V5.

6.2. Experimental Polarsys platform

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

In the context of the OPEES project (http://www.opees.org/), we have experimented the IWG Eclipse platform Polarsys (http://www.eclipse.org/org/press-release/20111102_polarsys.php). Polarsys is a new industry collaboration to build open source tools for safety-critical software development. The integration of Polychrony into this platform has been realized in collaboration with the CS company. CS and Inria have produced the Polychrony experimentation report which is included in the global experimentation report. This document gathers the experiments performed by the several partners involved in the OPEES project on the Polarsys platform. An experiment is defined as the way one partner takes his component and uses it to check any of the services within the Polarsys environment. The services are functions the partners want the Polarsys environment to offer.

For the qualification of the Polychrony component on the Polarsys platform, CS and Inria provide the following documents:

- The Tool Quality Assurance Plan Template (TQAP). This document defines the OPEES quality assurance arrangements and gives some guidance to satisfy them. It focuses on qualification aspects and gives in appendices guidance for some criteria tool qualification with an example for Polychrony Tool.
- The Tool Verification Cases and Procedures (TVCP) document. It presents the test cases to be performed for the qualification of Polarsys Polychrony Verifier component as described in the TQAP.
- The Tool Verification Results (TVR). It presents the results of tests performed for the qualification of Polychrony Verifier on several operating systems, as described in the TQAP.

6.3. Translation validation of Polychronous Equations with an iLTS Model-checker

Participants: Van-Chan Ngo, Jean-Pierre Talpin, Thierry Gautier, Paul Le Guernic, Loïc Besnard.

This work [16], [18], which is part of the VeriSync project, focuses on verifying the correctness of transformations on abstract clocks in the Signal compiler [8]. We propose to use model checking technique over Polynomial Dynamical Systems (PDS) with the Sigali model checker [39].

Adopting the *translation validation* approach of [55], [54], we present an automated verification process to prove the correctness of a multi-clocked synchronous language compiler. Due to the very important role of abstract clocks and clock relations, we are interested in proving that abstract clocks and clock relations semantics of source programs are preserved during the compilation phases. Each individual transformation or optimization step of the compiler is followed by our verification process which proves the correctness of this step. The compiler will continue its work if and only if the correctness is proved positively. This approach avoids the disadvantage of proving in advance that the compiler always behaves correctly since every small change to the compiler requires reproving it.

Our verification framework uses polynomial dynamical systems (PDS) over a finite field, as common semantics for both source and transformed programs. We formalize the abstract clocks semantics of *polychronous equations* with the finite field modulo $p = 3$ as a PDS [16]. For a signal x, if x is boolean, we use the values $-1, 0, +1$ to encode (respectively) the fact that it is present and false, absent, or present and true. Then, the abstract clock can be represented by x^2. If x is non-boolean, we only encode the abstract clock by x^2, meaning that $x^2 = 0$ encodes x is absent, $x^2 = 1$ encodes x is present. An appropriate relation called *refinement* for PDSs is proposed to represent the correct transformation relation between the source and transformed programs. Then a dedicated checking procedure is proposed within Sigali to check the correct transformation relation. The checking procedure is based on the simulation techniques [30]. It is implemented as extension function of the Sigali model checker within the Polychrony toolset.

We have proposed an approach to prove the clock semantic preservation of the Signal compiler transformations until the generation code phase as well. The verification method applied to code generation phase addresses the formal verification of the generated C-code from a refined and optimized intermediate specification in which the compiler enforces logical timing constraints and in which the execution order of data-flow equations is completely scheduled. As a result, all individual transformations, optimizations, and code generation phases of the compiler are followed by a verification step which proves the correctness of transformations. The compiler continues if and only if correctness is proved and returns an error and a trace otherwise. The main idea is that the sequential C code is translated into the target synchronous program thanks to the intermediate SSA form, which is based on the work in [3]. In addition, if a refinement relation between two PDSs does not exist, our validator will find the set of states along with their associated events, which can be used to construct counterexamples in the transformed program [18].

6.4. Formal Verification of Transformations on Abstract Clock in Synchronous Compilers

Participants: Van-Chan Ngo, Jean-Pierre Talpin, Paul Le Guernic.

Translation validation was introduced in the 90's by Pnueli et al. as a technique to formally verify the correctness of code generated from the data-flow synchronous language Signal using model checking. Rather than certifying the code generator (by writing it entirely using a theorem prover) or qualifying it (by obeying to the 27 documentations required as per the DO-178C), translation validation provides a scalable approach to assessing the functional correctness of automatically generated code. By revisiting translation validation, which in the 90's suffered from the limitations of theorem proving and model checking techniques available then, we aim at developing a scalable and flexible approach that applies to the existing 500k-lines implementation of Signal, Polychrony, and is capable of handling large-scale, possibly automatically generated, Signal programs, while using of-the-shelf, efficient, model-checkers and SAT/SMT-solving libraries [36], [63].

The abstract clock semantics of polychronous equations is formalized as a first-order logic formula over boolean variables. For a signal x, if x is boolean then we use two boolean variables x, and $\widehat{x}$ to represent the value of signal x and it abstract clock, respectively. If x is non-boolean signal, we only need to capture its abstract clock by a boolean variable $\widehat{x}$. The boolean variable $\widehat{x}$ is true when the signal x is present and otherwise it is absent. The equational structure of a synchronous data-flow language makes that it is natural to represent the relations between abstract clocks described implicitly or explicitly by equations in terms of

first-order logic formulas. And then the combination of equations can be represented by the conjunction of the corresponding first-order logic formulas. We assume that all considered programs are supposed to be written with the primitive operators, meaning that derived operators are replaced by their corresponding primitive ones, and there is no nested operators such as $z := x$ default $(y$ when $b)$. The nested operators are broken by using fresh variables. These formulas use the usual logic operators and numerical comparison functions. Consider a general equation $y := R(x_1, x_2, ..., x_n)$, where R is an operator defined in a synchronous language (e.g. `suspend` in Esterel, `when` in Signal...), or it can be a usual boolean or numerical operator, then the abstract clock semantics of this equation can be represented as a first-order logic formula over the clocks and/or the boolean value of the involved signals $\Phi(\widehat{y}, \widehat{x_1}, \widehat{x_2}, ..., \widehat{x_n}, x_1, ...)$. For a boolean expression defined by numerical comparison functions and numerical expressions, to ensure the result formulas are boolean, we only encode the fact that the clocks of boolean and numerical expressions are synchronized, and we avoid encoding the numerical comparison function which defines the value of the boolean expression and the numerical expressions. For each i^{th} equation in program P, we denote by Φ_{eq_i} its abstract clock semantics, then the abstract clock semantics of P can be represented by a first-order logic formula, called its *clock model*, denoted as:

$$\Phi_P = \bigwedge_i^n \Phi_{eq_i} \tag{1}$$

where n denotes the number of equations composed in P. The detailed encoding scheme of the Signal language can be found in [19].

Given two clock models P_1 and P_2, we say that P_2 is a *correct transformation* on abstract clocks of P_1, or P_2 *refines* P_1 w.r.t the clock semantics, if it satisfies:

$$\forall I.(I \models \Phi_{P_2} \rightarrow I \models \Phi_{P_1}) \tag{2}$$

We write $P_2 \sqsubseteq_{clock} P_1$ to denote the fact that P_2 refines P_1. We also provide an approach to check the existence of refinement by using a SMT-solver that is based on the following theorem:
Theorem. Given a source program P_1 and its transformed program P_2, P_2 is a correct transformation of P_1 on abstract clocks if it satisfies that the formula Φ_{P_1} is a logical consequence of the formula Φ_{P_2}, or

$$\models (\Phi_{P_2} \rightarrow \Phi_{P_1}) \text{ if and only if } P_2 \sqsubseteq_{clock} P_1 \tag{3}$$

Here, we delegate the checking of the above formula against the clock models to a SMT-solver that efficiently deals with first-order logic formulas over boolean and numeric expressions. The checking formulas belong to decidable theories, this solver gives two types of answers: *sat* when the formula has a model (there exists an interpretation that satisfies it); or *unsat* otherwise. Our implementation uses the SMTLIB common format [31] to encode the formulas obtained from the previous step as input of SMT solvers. For our implementation, we consider the Yices solver [38], which is one of the best two solvers at the last SMTCOMP competition [59].

6.5. Formal Verification of Transformations on Data Dependency in Synchronous Compilers

Participants: Van-Chan Ngo, Jean-Pierre Talpin, Paul Le Guernic.

We propose an approach to prove the data dependency semantic preservation of transformations in a synchronous compiler (such as that of Signal). In the Signal language, the scheduling or data dependency is expressed implicitly through polychronous equations. We use *Synchronous Data-flow Dependence Graphs* (SDD Graphs) [46], [50] to formalize the data dependency semantics of polychronous equations. A tuple $< G, C, fE >$ is a SDD graph if and only if:

- $G =< N, E, I, O >$ is a dependence graph $< N, E >$ with I/O nodes: the inputs I and the output O such that I, O are subsets of N and I and O are disjoint.
- C is a set of constraints, called clocks.
- $fE : E \longrightarrow C$ is a mapping labeling each edge with a clock; it specifies the existence condition of the edges.

For instance, for the *counter* example:
$zv := v\$1|v := (1 \text{ when } rst) \text{ default } zv + 1$
we get a SDD graph with:

- $N = \{1, v, zv + 1\}$
- $E = \{(1, v), (zv + 1, v)\}$
- $C = \{\widehat{rst}, \widehat{v} \wedge \neg\widehat{rst}\}$
- $fE((1, v)) = \widehat{rst}, fE((zv + 1, v)) = \widehat{v} \wedge \neg\widehat{rst}$

Let $SDD_1 =< G_1, C_1, fE_1 >$ and $SDD_2 =< G_2, C_2, fE_2 >$ to be two SDD graphs which represent the data dependency semantics of source and transformed programs, we say that SDD_2 is a *correct transformation* of SDD_1 on data dependency, or SDD_2 *refines* SDD_1 w.r.t the data dependency semantics, if it satisfies that for any pair of nodes $x, y \in G_1 \cap G_2$ with $(x, y) \in E_1$:

- $fE_1(x, y)) \Rightarrow((x, y) \in E_2 \wedge fE_2(x, y))$ (reinforcement)
- $(fE_2(x, y) \wedge fE_2(y, x) \Leftrightarrow \text{ false})$ (deadlock consistency)

6.6. Experiment with constraint-based testing

Participants: Christophe Junke, Loïc Besnard, Jean-Pierre Talpin.

Based on past experiences with contraint-based testing of Lustre programs, we investigated automatic test *sequences* generation for Signal: from a given test objective expressed as a boolean flow (or an event), we try to generate a sequence of inputs over discrete time which lead to an observation of the test objective. Our approach was based on an existing tool named GATeL, from CEA LIST, with the kind permission of its authors. This tool targets the Lustre language, so we reused Polychrony's Lustre generator to export Signal programs as Lustre nodes and use the result with GATeL to generate test sequences. The resulting test sequences were in turn formatted in a way suitable for simulation according to the original compilation of Signal to C: in other words, the generated sequences were tested on the actual program resulting from compilation of considered Signal specifications. During this experiment, we corrected Signal's Lustre generator tool which suffered from some several bugs that made it emit consistently incorrect Lustre programs. After some work, we could translate faithfully a little more than sixty existing Signal programs of simple to moderate complexity.

Our contribution is an example of how Signal can benefit from the pool of existing tools applicable to Lustre and why having a correct Signal-to-Lustre translator can be useful for Signal programs. This approach has its limits because it is not always possible nor adequate to fully translate a Signal program to Lustre: (1) By requiring the existence of one root clock and changing a program's input/output interface, it may be possible to simulate a Signal program in Lustre, but with loss of information (like user-defined flow dependencies); hence, some results based on the one Lustre implementation of a model may not easily be generalized to every possible execution of the original Signal program; (2) the complexity of Signal's semantics is mainly motivated by the power it gives to handle partial system specifications during the development process, whereas most Lustre tools expect fully defined executable programs; as such, they are of little help when dealing with most Signal programs. For those main reasons, it might be better to study and implement verification techniques around the Signal language and extend the set of formal tools that can reason about Signal programs.

More generally, our experiment can lead us to consider the use of constraint solving techniques with Signal, not only for verification but also compilation and simulation.

6.7. Polychronous modeling, analysis and validation for timed software architectures in AADL

Participants: Yue Ma, Huafeng Yu, Paul Le Guernic, Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin.

High-level architecture modeling languages, such as AADL (Architecture Analysis and Design Language), are gradually adopted in the design of embedded systems so that design choice verification, architecture exploration and system property checking are carried out as early as possible. We are interested in the clock-based timing analysis, modeling and validation of software architectures specified in AADL [15]. In our approach, we first analyze the timing semantics of AADL, from which the formal polychronous/multiclock semantics is derived thanks to the multiclock nature of AADL specifications. Thus users are not suffered to find and/or build the fastest clock in the system. This distinguishes from [45], [37], where synchronous semantics is a prerequisite. This polychronous semantics is then expressed via a polychronous model of computation (MoC) [8] covering both AADL software, execution platform, and their binding. In addition, AADL thread-level scheduling is also explored and integrated according to affine clock relations [58]. In the framework of Polychrony, C or Java code is generated from the polychronous MoC. Simulation can then be carried out for the purpose of performance evaluation and verification.

Polychrony provides the back-end semantic-preserving transformation, scheduling, code generation, formal analysis and verification, architecture exploitation, and distribution [2]. With the scheduler synthesis, the translated AADL model is complete and executable, and can be used for the following analysis and validation [15]: 1) static analysis, including determinism identification and deadlock detection; 2) profiling-based analysis of real-time characteristics of a system [47]; 3) affine clock calculus to analyze the affine relations between clocks [58]; 4) co-simulation of AADL specifications and demonstration using the VCD technique [60]; 5) real-time scheduling and software/hardware allocation through the SynDEx tool [43].

An automatic toolchain dedicated to AADL modeling, scheduling, time analysis, verification, and simulation has been implemented and also integrated as plug-ins in the Eclipse framework. This toolchain (referred to as ASME2SSME) has been migrated from AADL V1.5.8 to AADL V2.0, together with OSATE V2. An experiment of interpretation of AADL Behavior Annex (BA) is initially performed, so that the Behavior Annex plugin is integrated in the modeling and transformation.

The whole model transformation and simulation chain has been migrated to Eclipse Indigo and attached to Polarsys as an Eclipse RCP. A tutorial case study, developed in the framework of the OPEES project [21], is adopted to illustrate the effectiveness of our contribution.

6.8. Static affine clocked-based scheduling and its seamless integration to ASME2SSME

Participants: Huafeng Yu, Yue Ma, Loïc Besnard, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.

An AADL model is not complete and executable if the thread-level scheduling is not resolved. Some scheduling tools, such as Cheddar [57], are well connected to AADL for schedulability analysis, scheduler synthesis and simulation inside these tools. However, they do not completely satisfy our demands for the following reasons: 1) logical and chronometric clocks are easily transformed into each other for formal and real-time analysis; 2) more events, such as input/output frozen events are also involved in the analysis; 3) static and periodic scheduling rather than stochastic/dynamic scheduling is expected due to predictability and formal verification; 4) the scheduling is easily and seamlessly connected to affine clock systems [58] so that formal analysis can be performed in Polychrony. Affine clock relations yield an expressive calculus for the specification and the analysis of time-triggered systems. A particular case of affine relations is the case of affine sampling relation expressed as $y = \{d \cdot t + \phi \mid t \in x\}$ of a reference discrete time x (d, t, ϕ are integers): y is a subsampling of positive phase ϕ and strictly positive period d on x.

We therefore propose a static affine-clocked-based scheduler synthesis process in the transformation from AADL to Signal, which includes the following subprocesses: 1) *calculate hyper-period* from the periods of all the threads according to the least common multiple principle; 2) *perform the scheduling* based on the hyper-period, and valid schedules are calculated according to a static, non-preemptive, and single-processor scheduling policy. More precisely, discrete events of each thread, such as dispatch, input/output frozen time, start and complete, are allocated in the hyper-period on condition that all their timing properties are satisfied. Affine clock relations of these events are ensured during the calculation. In the calculation process, different scheduling policies are considered, such as EDF and RM; 3) *export schedules to Signal affine clocks in a direct way*. This process, implemented as an independent Eclipse plugin, has been seamlessly integrated into the ASME2SSME toolchain.

6.9. Code distribution and architecture exploration via Polychrony and SynDEx

Participants: Huafeng Yu, Yue Ma, Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin, Paul Le Guernic.

We propose an approach to address code distribution and multi-processor architecture exploration via the Polychrony and SynDEx toolchains. We consider high-level AADL models for the specification of multi-processor architecture. This architecture generally has a multiclock nature, thus it is modeled with a polychronous MoC. In this way, users are not suffered to find and/or build the fastest clock for a multi-processor architecture. According to this principle, AADL models are transformed into Signal models. To bridge between the polychronous semantics of Signal and synchronous semantics of SynDEx, clock synthesis in Polychrony [24] is applied. The translation from Signal to SynDEx is integrated in Polychrony. Finally, SynDEx models are used to perform distribution, scheduling, and eventually executive code generation for a specific architecture.

The main advantages of our approach are: 1) a formal model is adopted to connect the three languages, and it helps to preserve the semantic coherence and correct code generation in the transformations; 2) the formal model and methods used in the transformation are transparent to AADL practitioners, and it is fast and efficient to have the illustrative results for architecture exploration; 3) it provides the possibility for one of the three languages to take advantage of the functionalities provided by the other two languages. A toolchain has been developed, which includes model transformations between the three languages, considering both semantic and syntactic aspects. A tutorial case study, developed in the framework of the CESAR project [20], was adopted to demonstrate our contribution.

6.10. Design of safety-critical Java applications using affine abstract clocks

Participants: Adnan Bouakaz, Jean-Pierre Talpin.

Safety-critical Java (SCJ) is a domain specific API of Java that aims at the development of qualified and certified embedded systems. Despite its simplified memory and concurrency models, it is always difficult to ensure functional determinism and schedule feasibility when using shared-memory and traditional lock-based mutual exclusion protocols. Automated code generation techniques from data-flow specifications allow waiving part of the difficult and error-prone tasks of programming real-time schedules for computations and communications from the engineering process. Our ADFG tool aims at automatic SCJ code generation from data-flow specifications for a multitask implementation with an earliest-deadline first scheduler. The tool integrates the necessary formal analyses, model transformations, and the SCJ annotation checker as well.

The underlying data-flow model, called the affine data-flow (ADF) model of computation [14], is similar to cyclo-static data-flow graphs; it has however ultimately periodic production and consumption rates and a time-triggered operational semantics. We have also proposed a scheduling analysis of ADF specifications that consists of two major steps:

- The construction of abstract affine schedules for computations that minimize buffering requirements under the assumption of read-write precedences and exclude overflow and underflow exceptions over communication channels. Affine transformations of clocks were first introduced in the context of Signal programs [58] and used in the ADF model to relate the activation rates of connected actors.

- The concretization of the affine schedules using an earliest-deadline first (EDF) symbolic schedulability analysis in a way that read-write precedences is ensured without the need for lock-based mechanisms and the processor utilization factor is maximized.

6.11. Polychronous controller synthesis from MARTE CCSL timing specifications

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

CCSL (Clock Constraint Specification Language) [29] is defined in an annex of the UML MARTE profile [53]. We are interested in the analysis, synthesis, code generation and simulation of polychronous systems specified in CCSL. Timed systems subject to clock expressions or relations can be modeled, specified, analyzed, and simulated within software environments such as SCADE [40], TimeSquare [44] and Polychrony. The motivation of our work, to address the simulation and code generation of polychronous systems, is to take advantage of the formal framework of Polychrony in the context of a high-level specification formalism, MARTE CCSL [62]. Yet, our work considers a novel approach with regard to previous approaches: to generate executable specifications by considering discrete controller synthesis (DCS) [56], [51], [52].

Based on our previous work on clock hierarchization [61] and the general clock synthesis approach [62], our current work concentrates on the study of interface-oriented clock synthesis in the context of distributed components. In this work, CCSL clock constraints are specified on the clocked signals that pass through the interface, and the controller to synthesize is used to ensure the constraints. Interface-level synthesis helps to reduce the synthesis complexity since communication concerns and internal component behavior are isolated from the synthesis. The controllability analysis of signals and clock relations are studied with regard to endochronous, polychronous, and reactive components. This analysis leads to the separation of controllable and uncontrollable signals in the synthesis. Observers of CCSL clock constraints have been proposed in order to specify control objectives. In addition, properties of local components and the global system, such as determinism and deadlock, are also initially studied.

6.12. An integration language for Averest/Quartz and Polychrony/Signal

Participants: Ke Sun, Jean-Pierre Talpin.

As typical synchronous languages, Quartz and Signal possess their own respective characteristics [11]. In particular, Quartz, an imperative synchronous language, mainly focuses on the description of the control-flow. The Quartz model is always reactive and synchronously deterministic. Different from Quartz models which can only be monochronous, a process in Signal may be polychronous, meaning that its clock hierarchy can form a forest. Therefore, the potential integration between Averest, a framework for Quartz, and Polychrony, a toolset for Signal, may offer a practical mode to develop globally asynchronous locally synchronous (GALS) systems: program imperative and reactive modules in Quartz, then synthesize the scheduler from their Signal network specification.

To maximally benefit from the existing achievements for the two languages [12], the main idea is to communicate the Quartz modules with each other via asynchronous wrappers without changing the original code. Considering that the Quartz modules should be still deterministic in asynchronous environment, the wrapper should be capable of controlling the IO streams. On the other hand, the wrapper, as a module interface, will make sense for automatic scheduler synthesis, the next step.

We will propose a new, easy to use, domain-specific language to help the user specify the input traces as requirements to the environment and define the IO traces as guarantee of the module. From the user-defined specification, a series of clock constraints, assertions, etc. may be synthesized in the form of Signal specification. Thus, this language may bridge the gap between Polychrony/Signal and Averest/Quartz.

7. Partnerships and Cooperations

7.1. National Initiatives

7.1.1. ANR

Program: ANR

Project acronym: VeriSync

Project title: Vérification formelle d'un générateur de code pour un langage synchrone

Duration: Nov. 2010 - Oct. 2013

Coordinator: IRIT

Other partners: IRIT

URL: http://www.irit.fr/Verisync/

Abstract:

The VeriSync project aims at improving the safety and reliability assessment of code produced for embedded software using synchronous programming environments developed under the paradigm of Model Driven Engineering. This is achieved by formally proving the correctness of essential transformations that a source model undergoes during its compilation into executable code.

Our contribution to VeriSync consists of revisiting the seminal work of Pnueli et al. on translation validation and equip the Polychrony environment with updated verification techniques to scale it to possibly large, sequential or distributed, C programs generated from the Signal compiler. Our study covers the definition of simulation and bisimulation equivalence relations capable of assessing the correspondence between a source Signal specification and the sequential or concurrent code generated from it, as well as both specific abstract model-checking techniques allowing to accelerate verification and counter-example search techniques, to filter spurious verification failures obtained from excessive abstracted exploration.

7.1.2. Competitivity Clusters

Program: FUI

Project acronym: P

Project title: Project P

Duration: March 2011 - Feb. 2014

Coordinator: Continental Automotive France

Other partners: 19 partners (Airbus, Astrium, Rockwell Collins, Safran, Thales Alenia Space, Thales Avionics...)

URL: http://www.open-do.org/projects/p/

Abstract:

The aim of project P is 1/ to aid industrials to deploy model-driven engineering technology for the development of safety-critical embedded applications, 2/ to contribute on initiatives such as OPEES and CESAR to develop support for tools inter-operability and 3/ to provide state-of-the-art automated code generation techniques from multiple, heterogeneous, system-levels models. The focus of project P is the development of a code generation toolchain starting from domain-specific modeling languages for embedded software design and to deliver the outcome of this development as an open-source distribution, in the aim of gaining an impact similar to GCC for general-purpose programming, as well as a kit to aid with the qualification of that code generation toolchain.

The contribution of project-team ESPRESSO in project P is to bring the necessary open-source technology of the Polychrony environment to allow for the synthesis of symbolic schedulers for

software architectures modeled with P in a manner ensuring global asynchronous deterministic execution.

The current activities in the project consist in gathering and writing detailed documentation about the project context, requirements and constraints. We are now familiar with the technologies involved in the project and started refining high-level requirements so as to to express technical objectives and solutions. The P formalism is still in the process of being defined and some aspects of the language are unknown (namely the sofwtare architecture formalism). For the subset of P that is sufficiently known and stable, we are investigating the semantical mapping between P and Signal with respect to controller synthesis.

7.1.3. CORAC

Program: CORAC

Project acronym: CORAIL

Project title: Composants pour l'Avionique Modulaire Étendue

Duration: Sep. 2011 - Dec. 2016

Coordinator: Thales Avionics

Other partners: Airbus, Dassault Aviation, Eurocopter, Sagem...

URL: http://www.corac-ame.com/

Abstract:

The CORAIL project aims at defining components for Extended Modular Avionics. The contribution of project-team ESPRESSO is to define a specification method and to provide a generator of multi-task applications.

7.2. European Initiatives

7.2.1. Collaborations in European Programs, except FP7

Program: ARTEMIS

Project acronym: CESAR

Project title: Cost-efficient methods and processes for safety relevant embedded systems

Duration: March 2009 - June 2012

Coordinator: AVL List GmbH

Other partners: 59 project partners (main partners for us: AIRBUS, IRIT (CNRS)...)

URL: http://www.cesarproject.eu/

Abstract:

In the context of CESAR, we have participated to the sub-project 3 demonstrator in order to demonstrate the usability of Polychrony as a co-simulation tool within the reference technology platform of the project, to which its open-source release has been integrated. The case-study, implemented in collaborateion with Airbus and IRIT, consists of co-modeling the doors management system of an Airbus A350 by merging its architecture description, specified with AADL, with its behavioral description, specified with Simulink.

In this case-study, we demonstrate that the Polychrony toolset can effectively serve as a modeling infrastructure to compositionally assemble, compile and verify heterogeneous specifications (AADL and Simulink). Our case study covers code generation for real-time simulation and test as well as formal verification both at system-level and in a GALS framework. Based on that case study, we are developing further modular code-generation services, real-time simulation, test and performance evaluation, formal verification as well as the validation of the generated concurrent and distributed code.

Program: ITEA2

Project acronym: OPEES

Project title: Open Platform for the Engineering of Embedded Systems

Duration: Feb. 2009 - Dec. 2012

Coordinator: Obeo

Other partners: 30 partners (main partners for us: Airbus, CS Communication & Systèmes, INDRA (Spain), INPT/IRIT...)

URL: http://www.opees.org/

Abstract: The ITEA2 project OPEES is the continuation of the ANR project OPENEMBEDD to provide an open-source platform for embedded software design. Its outcome will outlive the duration of the project as it has given rise to an Industrial Working Group of the Eclipse consortium, Polarsys, whose goal is to host and maintain the proposed open-source platform and guarantee its long-term availability.

The mission of OPEES is to build a community able to ensure durability of innovative engineering technologies in the domain of critical software-intensive embedded systems. Its main objectives are to secure the industrial strategy, improve their competitiveness and develop the European software industry.

Our goal in the OPEES project was to deliver the Polychrony toolset on the Polarsys platform as an infrastructure for the co-simulation and co-verification of embedded architectures. To this end, Polychrony has been under a quality assessment process performed in collaboration with CS.

7.3. International Initiatives

7.3.1. Inria Associate Teams

7.3.1.1. POLYCORE

Title: Polychronous models

Inria principal investigator: Jean-Pierre Talpin

International Partner (Institution - Laboratory - Researcher):

Virginia Tech (United States) - Fermat Laboratory - Sandeep Shukla

Duration: 2011 - 2013

See also: http://www.irisa.fr/espresso/Polycore

Inria Associate Project POLYCORE starts from an observation that can be shared with anyone how experienced with multi-threaded programming, to acknowledge the difficulty of designing and implementing such software. Resolving concurrency, synchronization, and coordination issues, and tackling the non-determinism germane in multi-threaded software is extremely difficult. Ensuring correctness with respect to the specification and deterministic behavior is however necessary for safe execution of such code on embedded architectures. It is therefore desirable to synthesize multi-threaded code from formal specifications using a provably 'correct-by-construction' approach.

While time-triggered programming model simplifies code generation, our shared intuition is that multi-rate event driven execution models are much more efficiently adapted to tackle embedded software design challenges posed by forthcoming heterogeneous multi-core embedded architectures. To this aim, we develop formal models, methods, algorithms and techniques for generating provably correct multi-threaded reactive real-time embedded software for mission-critical applications. For scalable modeling of larger embedded software systems, the specification formalism has to be compositional and hierarchical.

Our proposed formalism entails a model of computation (MoC) based on a multi-rate synchronous data-flow paradigm: Polychrony. It aims at combining the capabilities of Esterel/Quartz

(ESG/TUKL) for correctly programming synchronous modules, with the capabilities of Polychrony (Inria), to give high-level abstractions of complex multi-clocked networks and yet provide powerful communication and scheduling code synthesis, all combined in an application-specific modeling and programming environment, design in collaboration with Virginia Tech and the AFRL [12], [11]. This year, we laid novel semantical foundations to designing our envisioned environment by defining a constructive semantic encompassing the polychronous data-flow model of Signal and the reactive synchronous imperative model of Quartz, and allowing to formulate the very first executable, small-step, structured operational semantics of Signal [17].

7.4. International Research Visitors

7.4.1. Visits of International Scientists

Pr. John Koo (SIAT, Shenzhen) visited ESPRESSO in summer 2012 with the support of the University of Rennes 1. During his stay, we elaborated a collaboration plan and project proposal on integrated discrete/continuous/hardware simulation with LIAMA.

In the context of the associate project Polycore, Jens Brandt (TU Kaiserslautern) visited ESPRESSO in June to share code generation techniques in Quartz and Signal. Loïc Besnard visited Virginia Tech in June to present the open-source release of Polychrony and explore possibles uses of Polychrony in the MRCDIF environment developed at the FLVT. Jean-Pierre Talpin visited Virginia Tech in May and October to prepare our work on Quartz and Signal and jointly draft a project proposal for the USAFRL.

7.4.2. Visits to International Teams

Jean-Pierre Talpin received a grant as invited scientist by the Chinese Academy of Science to visit the Shenzhen Institute for Advanced Technology in December 2012 and further ongoing collaborations with Pr. Koo and LIAMA.

8. Dissemination

8.1. Scientific Animation

8.1.1. Invited Lectures

Jean-Pierre Talpin gave a series of invited lectures on synchronous programming and on Polychrony at the Ecole Centrale de Pékin and at Beihang University, May 2012, in Beijing, China.

Jean-Pierre Talpin was invited at the 2012 ACM/IEEE Conference on Logic in Computer Science, in June 2012, Dubrovnik, to receive the ACM/IEEE LICS Test of Time Award. He gave a short presentation on the technical challenges and theoretical breakthroughs of his awarded paper: “A type and effect discipline”, with his co-author Pierre Jouvelot.

Jean-Pierre Talpin participated to the Dagstuhl meeting on Architecture-Driven Semantic Analysis of Embedded Systems in June 2012, and presented the tools developped by ESPRESSO for co-modeling embedded software architectures with Polychrony.

Jean-Pierre Talpin gave an invited presentation on extending AADL with a timed annex at the AADL standardisation committee and SAE conference, October 2012 in Phoenix, a followup of which is the definition by the ESPRESSO project-team of a synchronous annex for AADL.

8.1.2. Conferences

Jean-Pierre Talpin is a member of the steering committee of the ACM-IEEE conference on methods and models for co-design (MEMOCODE) and of the editorial board of the EURASIP Journal on Embedded Systems. He served as technical program committee member for the following conferences:

HLDVT’11 http://www.hldvt.com/11
SAC’11 http://www.acm.org/conferences/sac/sac2011
SIES’11 http://www.mrtc.mdh.se/sies2011

Jean-Pierre Talpin co-organized with Elisabeth Lebret the 19th. edition of the synchronous programming workshop, held in Le Croisic, November 2012 (http://synchron2012.inria.fr).

Thierry Gautier served as technical program commitee member for the conference ESLsyn 2012 (http://www.ecsi.org/eslsyn2012) and MEMOCODE 2012 (http://memocode.irisa.fr/2012/MEMOCODE%202012.html).

8.2. Teaching - Supervision - Juries

8.2.1. Juries

Jean-Pierre Talpin served as referee and president at the thesis defense of Mikel Cordovilla, entitled “Environnement de développement d’applications multi-périodiques sur plateforme multi-cœur: la boîte à outils SchedMCore”, held on April 2, 2012 at ONERA.

Paul Le Guernic served as examiner at the Habilitation thesis defense of Abdoulaye Gamatié, entitled “Design and analysis for multi-clock and data-intensive applications on multiprocessor systems-on-chip”, held on November 15, 2012 at Université de Lille 1. He served also as referee at the thesis defense of Michaël Lafaye, entitled “Modélisation de plate-forme avionique pour exploration de performance en avance de phase”, held on November 19, 2012 at Télécom ParisTech.

Jean-Pierre Talpin served on the Professor selection committee of ENSEEIHT in May 2012.

Thierry Gautier served on an Associate Professor selection committee of Université de Rennes 1 in May 2012.

Loïc Besnard served as president in a jury for the competitive selection of engineers at CNRS in October 2012.

Paul Le Guernic served as “rapporteur” of Inria Research Center Rennes - Bretagne-Atlantique at the Regional symposium of Higher Education and Research (Assises territoriales de l’enseignement supŕieur et de la recherche) in Nantes and Rennes.

9. Bibliography

Major publications by the team in recent years

[1] 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", 2003, vol. 91, n^{o} 1, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.1117.

[2] L. BESNARD, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Compilation of Polychronous Data Flow Equations*, in "Synthesis of Embedded Software", S. K. SHUKLA, J.-P. TALPIN (editors), Springer, 2010, p. 1-40 [*DOI :* 10.1007/978-1-4419-6400-7_1], http://hal.inria.fr/inria-00540493.

[3] 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 "Electronic Communication of the European Association of Software Science and Technology", 2009, vol. 23, Automated Verification of Critical Systems 2009, http://journal.ub.tu-berlin.de/eceasst/article/view/312/301.

[4] C. BRUNETTE, J.-P. TALPIN, A. GAMATIÉ, T. GAUTIER. *A metamodel for the design of polychronous systems*, in "The Journal of Logic and Algebraic Programming", 2009, vol. 78, n^{o} 4, p. 233 - 259, IFIP WG1.8 Workshop on Applying Concurrency Research in Industry [*DOI :* 10.1016/J.JLAP.2008.11.005], http://www.sciencedirect.com/science/article/pii/S1567832608000957.

[5] 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)", April 2007, vol. 16, n° 2, http://doi.acm.org/10.1145/1217295.1217298.

[6] A. GAMATIÉ, T. GAUTIER. *The Signal Synchronous Multiclock Approach to the Design of Distributed Embedded Systems*, in "IEEE Transactions on Parallel and Distributed Systems", 2010, vol. 21, n° 5, p. 641-657 [*DOI :* 10.1109/TPDS.2009.125], http://hal.inria.fr/inria-00522794.

[7] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Synchronous design of avionic applications based on model refinements*, in "Journal of Embedded Computing (IOS Press)", 2006, vol. 2, n° 3-4, p. 273-289, http://hal.archives-ouvertes.fr/hal-00541523.

[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", June 2003, vol. 12, n° 03, http://hal.inria.fr/docs/00/07/18/71/PDF/RR-4715.pdf.

[9] D. POTOP-BUTUCARU, Y. SOREL, R. DE SIMONE, J.-P. TALPIN. *From Concurrent Multi-clock Programs to Deterministic Asynchronous Implementations*, in "Fundamenta Informaticae", January 2011, vol. 108, n° 1-2, p. 91–118, http://dl.acm.org/citation.cfm?id=2362088.2362094.

[10] J.-P. TALPIN, P. LE GUERNIC, S. SHUKLA, R. GUPTA. *A compositional behavioral modeling framework for embedded system design and conformance checking*, in "International Journal of Parallel Programming", December 2005, vol. 33, n° 6, p. 613-643 [*DOI :* 10.1007/s10766-005-8907-y], http://hal.archives-ouvertes.fr/hal-00541986.

Publications of the year

Articles in International Peer-Reviewed Journals

[11] J. BRANDT, M. GEMÜNDE, K. SCHNEIDER, S. SHUKLA, J.-P. TALPIN. *Representation of synchronous, asynchronous, and polychronous components by clocked guarded actions*, in "Design Automation for Embedded Systems", 2012 [*DOI :* 10.1007/s10617-012-9087-9], http://hal.inria.fr/hal-00763334.

[12] J. BRANDT, M. GEMÜNDE, K. SCHNEIDER, S. SHUKLA, J.-P. TALPIN. *Embedding Polychrony into Synchrony*, in "IEEE Transactions on Software Engineering", 2013, http://hal.inria.fr/hal-00763317.

[13] J.-P. TALPIN, J. OUY, T. GAUTIER, L. BESNARD, P. LE GUERNIC. *Compositional design of isochronous systems*, in "Science of Computer Programming", February 2012, vol. 77, n° 2, p. 113-128 [*DOI :* 10.1016/j.scico.2010.06.006], http://hal.archives-ouvertes.fr/hal-00768341.

International Conferences with Proceedings

[14] A. BOUAKAZ, J.-P. TALPIN, J. VITEK. *Affine Data-Flow Graphs for the Synthesis of Hard Real-Time Applications*, in "Proceedings of the 2012 12th International Conference on Application of Concurrency to System Design", Hamburg, Germany, ACM, 2012, p. 183-192 [*DOI :* 10.1109/ACSD.2012.16], http://hal.inria.fr/hal-00763387.

[15] Y. MA, H. YU, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN, L. BESNARD, M. HEITZ. *Toward Polychronous Analysis and Validation for Timed Software Architectures in AADL*, in "The Design, Automation, and Test in Europe (DATE) conference", Grenoble, France, March 2013, 6, http://hal.inria.fr/hal-00763379.

[16] V. C. NGO, L. BESNARD, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Formal Verification of Compiler Transformations on Polychronous Equations*, in "International Conference on Integrated Formal Methods", Pisa, Italy, CNR/ISTI Italy, June 2012, http://hal.inria.fr/hal-00730393.

[17] J.-P. TALPIN, J. BRANDT, M. GEMÜNDE, K. SCHNEIDER, S. SHUKLA. *Constructive Polychronous Systems*, in "Logical Foundations of Computer Science", San Diego, CA, United States, S. ARTEMOV, A. NERODE (editors), Lecture Notes in Computer Science, Springer, 2013, vol. 7734, http://hal.inria.fr/hal-00763371.

Research Reports

[18] V. C. NGO, J.-P. TALPIN, T. GAUTIER, P. LE GUERNIC, L. BESNARD. *Formal Verification of Synchronous Data-flow Compilers*, Inria, 2012, n^{o} RR-7921, http://hal.inria.fr/hal-00685633.

[19] V. C. NGO, J.-P. TALPIN, P. LE GUERNIC. *Formal Verification of Transformations on Abstract Clocks in Synchronous Compilers*, Inria, September 2012, n^{o} RR-8064, http://hal.inria.fr/hal-00730926.

References in notes

[20] *Cost-efficient methods and processes for safety relevant embedded systems (CESAR project)*, http://www.cesarproject.eu/.

[21] *Open Platform for the Engineering of Embedded Systems (OPEES Project)*, http://www.opees.org/.

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

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

[24] *The Polychrony Toolset*, http://www.irisa.fr/espresso/Polychrony/.

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

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

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

[28] P. AMAGBEGNON, L. BESNARD, P. LE GUERNIC. *Implementation of the data-flow synchronous language SIGNAL*, in "ACM SIGPLAN Notices", June 1995, vol. 30, n^{o} 6, p. 163-173 [*DOI :* 10.1145/223428.207134], http://hal.archives-ouvertes.fr/hal-00544128.

[29] 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.

[30] C. BAIER, J.-P. KATOEN. *Principles of model checking*, in "The MIT Press", 2008.

[31] C. BARRETT, S. RANISE, A. STUMP, C. TINELLI. *The Satisfiability Modulo Theories Library (SMT-LIB)*, 2008, http://www.SMT-LIB.org.

[32] 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, Springer, August 1999, vol. 1664, p. 162–177, http://hal.inria.fr/inria-00073032.

[33] 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.

[34] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT. *Synchronous programming with events and relations: the Signal language and its semantics*, in "Science of Computer Programming", September 1991, vol. 16, n° 2, p. 103-149, http://dx.doi.org/10.1016/0167-6423(91)90001-E.

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

[36] A. BIERE, M. HEULE, H. VAN MAAREN, T. WALSH. *Handbook of satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, in "ISBN 978-1-5860-3929-5", 2009.

[37] M. BOZZANO, A. CIMATTI, J.-P. KATOEN, V. NGUYEN, T. NOLL, M. ROVERI. *Safety, Dependability, and Performance Analysis of Extended AADL Models*, in "The Computer Journal", 2011, vol. 54, n° 5, p. 754–775.

[38] B. DUTERTRE, L. DE MOURA. *Yices sat-solver*, 2009, http://yices.csl.sri.com.

[39] B. DUTERTRE, M. LE BORGNE, H. MARCHAND. *SIGALI : un système de calcul formel pour la vérification de programmes SIGNAL*, 1998, Manuel d'utilisation.

[40] ESTEREL TECHNOLOGIES. *SCADE Suite*, http://www.esterel-technologies.com/products/scade-suite/.

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

[42] 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, Royaume-Uni, Springer, 1999, p. 127-149, http://hal.archives-ouvertes.fr/hal-00543824.

[43] INRIA AOSTE TEAM. *SynDEx*, http://www-rocq.inria.fr/syndex/.

[44] INRIA AOSTE TEAM. *TimeSquare*, http://timesquare.inria.fr/.

[45] E. JAHIER, N. HALBWACHS, P. RAYMOND, X. NICOLLIN, D. LESENS. *Virtual execution of AADL models via a translation into synchronous programs*, in "ACM & IEEE international conference on Embedded software (EMSOFT)", 2007.

[46] G. KAHN. *The Semantics of a Simple Language for Parallel Programming*, in "IFIP Congress", 1974, p. 471-475.

[47] A. KOUNTOURIS, P. LE GUERNIC. *Profiling of SIGNAL programs and its application in the timing evaluation of design implementations*, in "IEE Colloquium on the Hardware-Software Cosynthesis for Reconfigurable", Bristol, Royaume-Uni, The Institution of Electrical Engineers, 1996, p. 6/1-6/9 [*DOI :* 10.1049/IC:19960225], http://hal.archives-ouvertes.fr/hal-00544253.

[48] P. LE GUERNIC, T. GAUTIER, M. LE BORGNE, C. LE MAIRE. *Programming Real-Time Applications with Signal*, in "Proceedings of the IEEE", 1991, vol. 79, n^{o} 9, p. 1321-1336 [*DOI :* 10.1109/5.97301], http://hal.inria.fr/inria-00540460.

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

[50] O. MAFFEIS, P. LE GUERNIC. *Distributed Implementation of SIGNAL: Scheduling & Graph Clustering*, in "Formal Techniques in Real-Time and Fault-Tolerant Systems", Lübeck, Allemagne, LNCS vol. 863, Springer-Verlag, 1994, p. 547-566, http://hal.archives-ouvertes.fr/hal-00544101.

[51] O. MALER, A. PNUELI, J. SIFAKIS. *On the Synthesis of Discrete Controllers for timed Systems*, in "Proceedings STACS'95", Lecture Notes in Computer Science, 1995, vol. 900, p. 229–242.

[52] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment*, in "Discrete Event Dynamic Systems", October 2000, vol. 10, n^{o} 4, p. 325-346 [*DOI :* 10.1023/A:1008311720696], http://hal.archives-ouvertes.fr/hal-00546147.

[53] OBJECT MANAGEMENT GROUP (OMG). *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, v1.1*, June 2011, http://www.omg.org/spec/MARTE/1.1/PDF/.

[54] A. PNUELI, M. SIEGEL, E. SINGERMAN. *Translation validation: From SIGNAL to C*, in "Correct Sytem Design Recent Insights and Advances", 2000, p. 231-255.

[55] A. PNUELI, M. SIEGEL, E. SINGERMAN. *Translation validation*, in "Proceedings of TACAS'98", 1998, p. 151-166.

[56] P. RAMADGE, W. WONHAM. *The Control of Discrete Event Systems*, in "Proceedings of the IEEE, Special issue on Dynamics of Discrete Event Systems", 1989, vol. 77, n^{o} 1, p. 81–98.

[57] F. SINGHOFF, J. LEGRAND, L. NANA, L. MARCÉ. *Scheduling and memory requirements analysis with AADL*, in "ACM SIGAda international conference on ADA (SigAda'05)", 2005, http://doi.acm.org/10.1145/1103846.1103847.

[58] I. SMARANDACHE, T. GAUTIER, P. LE GUERNIC. *Validation of Mixed Signal-Alpha Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints*, in "FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems", Toulouse, France, LNCS vol. 1709, Springer, 1999, p. 1364-1383 [*DOI :* 10.1007/3-540-48118-4_22], http://hal.archives-ouvertes.fr/hal-00548887.

[59] A. STUMP, M. DETERS. *http://www.smtcomp.org/2009*, 2009.

[60] H. YU, Y. MA, Y. GLOUCHE, J.-P. TALPIN, L. BESNARD, T. GAUTIER, P. LE GUERNIC, A. TOOM, O. LAURENT. *System-level Co-simulation of Integrated Avionics Using Polychrony*, in "ACM Symposium On Applied Computing", TaiChung, Taïwan, March 2011, http://hal.inria.fr/inria-00536907/en/.

[61] H. YU, J.-P. TALPIN, L. BESNARD, T. GAUTIER, F. MALLET, C. ANDRÉ, R. DE SIMONE. *Polychronous Analysis of Timing Constraints in UML MARTE*, in "IEEE International Workshop on Model-Based Engineering for Real-Time Embedded Systems Design", Parador of Carmona, Espagne, 2010, 7 p., http://hal.inria.fr/inria-00497249.

[62] H. YU, J.-P. TALPIN, L. BESNARD, T. GAUTIER, H. MARCHAND, P. LE GUERNIC. *Polychronous Controller Synthesis from MARTE CCSL Timing Specifications*, in "ACM/IEEE Ninth International Conference on Formal Methods and Models for Codesign (MEMOCODE)", Cambridge, Royaume-Uni, July 2011, http://hal.inria.fr/inria-00594942/en/.

[63] L. DE MOURA, N. BJORNER. *Satisfiability Modulo Theories: An appetizer*, in "Brazilian Symposium on Formal Methods", 2009.