IN PARTNERSHIP WITH:
CNRS

Université Rennes 1

Activity Report 2013

Project-Team ESPRESSO

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

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Embedded and Real Time Systems

Table of contents

1. Members **1**
2. Overall Objectives **1**
  2.1. Introduction 1
  2.2. Context and motivations 2
  2.3. The polychronous approach 2
  2.4. Highlights of the Year 3
3. Research Program **3**
  3.1. Introduction 3
  3.2. Polychronous model of computation 4
    3.2.1. Composition 4
    3.2.2. Scheduling 5
    3.2.3. Structure 5
  3.3. A declarative design language 5
  3.4. Compilation of Signal 7
    3.4.1. Synchronization and scheduling specifications 7
    3.4.2. Synchronization and scheduling analysis 7
    3.4.3. Hierarchization 7
4. Application Domains **8**
5. Software and Platforms **8**
  5.1. The Polychrony toolset and its hypertext source documentation 8
  5.2. The Eclipse interface 10
  5.3. Integrated Modular Avionics design using Polychrony 11
  5.4. Safety-Critical Java Level 1 Code generation from Dataflow Graph Specifications 11
6. New Results **11**
  6.1. A pivot in between synchrony and asynchrony 11
  6.2. New functionalities of Polychrony 12
  6.3. Formal Verification of Synchronous Dataflow Program Transformations Toward Certified Compilers 12
  6.4. Exploring system architectures in AADL via Polychrony and SynDEx 14
  6.5. A synchronous annex for the AADL 14
  6.6. Ongoing activities and results for integration of Polychrony with the P toolset 14
  6.7. Real-Time Scheduling of Dataflow Graphs 16
  6.8. Structure-Preserved Distribution of Synchronous Programs 16
7. Partnerships and Cooperations **17**
  7.1. National Initiatives 17
    7.1.1. ANR 17
    7.1.2. Competitivity Clusters 17
    7.1.3. CORAC 18
  7.2. International Initiatives 18
    7.2.1. Inria Associate Teams 18
    7.2.2. Inria International Partners 19
      7.2.2.1. The University of Hong Kong, Emerging Technologies Institute 19
      7.2.2.2. Beihang University, Institute of Computer Architectures 19
    7.2.3. Participation In other International Programs 20
  7.3. International Research Visitors 20
8. Dissemination **20**
  8.1. Scientific Animation 20
  8.2. Teaching - Supervision - Juries 21
    8.2.1. Teaching 21

8.2.2. Juries 21
9. Bibliography **21**

Project-Team ESPRESSO

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

Creation of the Project-Team: 2002 January 01*, updated into Team:* 2013 January 01, end of the Project-Team: 2013 December 31.

1. Members

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

Faculty Member
Adnan Bouakaz [Univ. Rennes I]

Engineers
Loïc Besnard [CNRS, Senior Engineer]
Christophe Junke [Inria, granted by OSEO Innovation]
Huafeng Yu [Inria, granted by OSEO Innovation, until Feb 2013]

PhD Students
Van-Chan Ngo [Inria, granted by ANR VeriSync project]
Ke Sun [Inria, granted by OSEO Innovation]

Administrative Assistant
Stephanie Lemaile [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 Systèmes), with which we had many collaborations, has supplied a commercial implementation of Polychrony, RT-Builder, which has been deployed on large-scale applications with the avionics and automotive industries (contact: http://www.geensoft.com).

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 forces 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 dataflow 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) [40], [2], [11].

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 call for unambiguous design models based on models and methods to translate a high-level system specification into a distribution of purely sequential programs together with semantics-preserving transformations and high-level optimizations such as hierarchization (sequentialization) or desynchronization (protocol synthesis).

System design based on the so-called “synchronous hypothesis” has in this respect focused the attention of many academic and industrial actors. The synchronous paradigm abstracts the non-functional implementation details of a system and focuses on the logics behind the instants at which the system functionalities should be secured.

Synchronous design models and languages provide intuitive models for embedded systems that ease the generation of systems and architectures and the verification of their functionalities [1].

In the relational mathematical model behind the design language Signal, the supportive dataflow 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 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 Eclipse Industry Working Group focusing on open source tools for the development of embedded systems. After previous years experimentation, POP, A Polychronous Modeling Environment on Polarsys, has been approved as open source project under the Polarsys Top-Level Project, which is operating under the auspices of the Polarsys Industry Working Group.

3. Research Program

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.

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 [29] can be superimposed on this model by considering a flow equivalence relation as well as heterogeneous systems [30] by parameterizing composition with arbitrary timing relations.

3.2. 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.

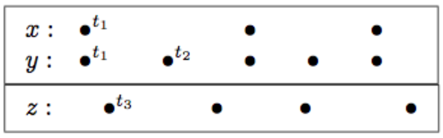

Figure 1. Behavior b over three signals x, y and z in two clock domains

3.2.1. Composition

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

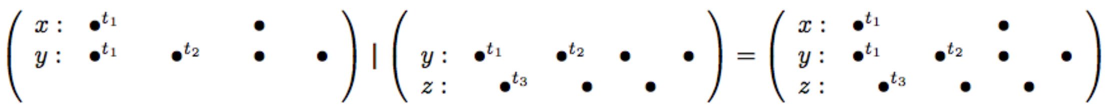

Figure 2. Synchronous composition of $b \in p$ and $c \in q$

3.2.2. 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 $\neg(t' < t)$.

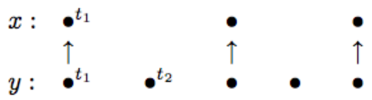

Figure 3. Scheduling relations between simultaneous events

3.2.3. Structure

A synchronous structure is a semi-lattice representing behaviors with the same timing structure. We interpret 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 $d \leq c$.

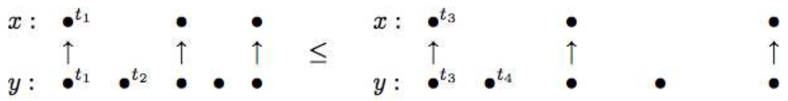

Figure 4. Relating synchronous behaviors by stretching.

3.3. A declarative design language

Signal [32], [45], [39] is a declarative design language using the polychronous model of computation. 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 $x := y\ f\ z$ 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}.

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

- 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).

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

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 dataflow 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 $\texttt{when}\,x$ (resp. $\texttt{when}\,\texttt{not}\,x$) 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 $x \to y\,\texttt{when}\,e$ allow to schedule the calculation of signals (e.g. x after y at the clock e).

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

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

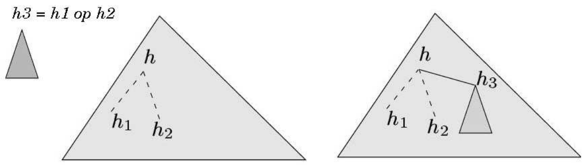

Figure 6. Hierarchization of clocks

To determine the order $x \preceq y$ in which signals are processed during the period of a reaction, clock relations E play an essential role. The process of determining this order is called hierarchization and consists of an insertion algorithm which hooks elementary control flow graphs (in the form of if-then-else structures) one to the others. Figure 6, right, let h3 be a clock computed using h1 and h2. Let h be the head of a tree from which h1 and h2 are computed (an if-then-else), h3 is computed after h1 and h2 and placed under h [27].

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 and Platforms

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 dataflow 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).

In 2013, to be able to use the Signal GUI both as a specific tool and as a graphical view under Eclipse, the code of the Signal GUI has been restructured in three parts: a common part used by both tools (28 classes), a specific part for the Signal GUI (2 classes), a specific part for Eclipse (2 classes). Such a structuration facilitates the maintenance of the products.

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.

The building of the Signal toolbox is managed using the CMake utility (http://www.cmake.org). It is used to ensure the portability on the different operating systems (CMake generates native makefiles). For the same reason, in 2013, we have integrated the management of the tests of the Signal batch compiler under CMake (using CTest).

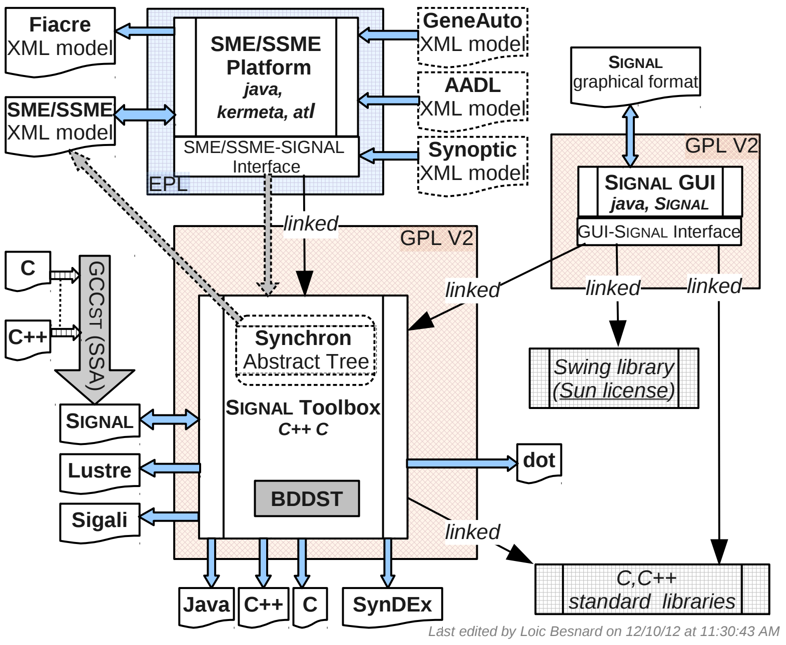

Figure 7. The Polychrony toolset high-level architecture

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

- The Polychrony toolset public web site: http://www.irisa.fr/espresso/Polychrony/index.php. 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.

In 2012, during the OPEES project, we have integrated Polychrony on the Polarsys Experimental Eclipse platform, a new industry collaboration to build open source tools for safety-critical software development. In 2013, the integration to the actual Eclipse Polarsys platform (http://www.polarsys.org) has been started. The proposal project is available at http://www.eclipse.org/proposals/polarsys.polychrony. The creation of a new Polarsys project (as defined in the Eclipse Development Process) is decomposed into several steps (before the first release): project proposal, community review, trademark review, creation review, provisioning, initial contribution. For trademark reasons, Polychrony is called POP as Polarsys Project (POP, a polychronous modeling environment on Polarsys). At the end of 2013, we are at the "provisioning" step. It will be provisioned as soon as the employer consent (Inria, CNRS) form will be signed.

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

Dassault Systèmes, supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects.

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

Participant: Loïc Besnard.

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 [33]: 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.

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 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 dataflow) part, and one to model all explicit clock relations and dependences. The SME environment is available through the ESPRESSO update site [24]. 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.

It should be noted that the Eclipse Foudation does not host code under GPL license. So, the Signal toolbox useful to compile Signal code from Eclipse is hosted on our web server. For this reason, the building of the Signal toolbox, previously managed under Eclipse, has now been exported. The interface of the Signal toolbox for Eclipse is now managed using the CMake tool like the Signal toolbox and the Signal GUI.

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 [25], 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 [26]). 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.

5.4. Safety-Critical Java Level 1 Code generation from Dataflow Graph Specifications

Participants: Adnan Bouakaz, Jean-Pierre Talpin.

We have proposed a dataflow design model [19] of SCJ/L1 applications [43] in which handlers (periodic and aperiodic actors) communicate only through lock-free channels. Hence, each mission is modeled as a dataflow graph. The presented dataflow design model comes with a development tool integrated in the Eclipse IDE for easing the development of SCJ/L1 applications and enforcing the restrictions imposed by the design model. It consists of a GMF editor where applications are designed graphically and timing and buffering parameters can be synthesized. Indeed, abstract affine scheduling is first applied on the dataflow subgraph, that consists only of periodic actors, to compute timeless scheduling constraints (e.g. relation between the speeds of two actors) and buffering parameters. Then, symbolic fixed-priority schedulability analysis (i.e., synthesis of timing and scheduling parameters of actors) considers both periodic and aperiodic actors.

Through a model-to-text transformation, using Acceleo, the SCJ code for missions, interfaces of handlers, and the mission sequencer is automatically generated in addition to the annotations needed by the memory checker. Channels are implemented as cyclic arrays or cyclical asynchronous buffers; and a fixed amount of memory is hence reused to store the infinite streams of tokens. The user must provide the SCJ code of all the `handleAsyncEvent()` methods. We have integrated the SCJ memory checker [52] in our tool so that potential dangling pointers can be highlighted at compile-time. To enhance functional determinism, we would like to develop an ownership type system to ensure that actors are strongly isolated and communicate only through buffers.

6. New Results

6.1. A pivot in between synchrony and asynchrony

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

Our time modeling framework requires a pivot specification paradigm to materialise a spectrum of models of computation and communication ranging from synchrony to asynchrony, from software to hardware, and accommodate with (abstractions of) software behaviors (software, functional blocks, tasks) and requirements (temporal properties, contracts, regular expressions) through logical, periodic, multi-periodic or affine time. We aim at developing a framework comprising dataflow networks (communications) and synchronous automata (computations) controlled by synthesised wrapper enforcing abstractions of specified constraints from the software viewpoint (timing requirements).

Relations between Kahn networks and classes of synchronous dataflow graphs (SDF) as well as synchronous languages have been studied in the past (e.g. Lustre), yet never in the full generality of relating the domain-theoretic model of Kahn networks to it most general synchronous incarnation (one that at least allows to express several clock domains) [17]. We are currently elaborating such a model to characterise morphisms between untimed asynchronous networks and multi-clocked, synchronous, dataflow networks. In this prospect, we developed the first constructive operational semantics of Signal [21], which opens to further investigations on its full abstraction relation with a denotational characterisation using Kahn networks over a polychronous domain.

6.2. New functionalities of Polychrony

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

We have developed and integrated in the Signal toolbox some clock computations useful for optimizations: *assignment clocks* and *utility clocks*. These information may be used to reduce the frequency of the computations and the communications for distributed code generation.

Assignment clock. A given signal is supposed to be computed at the instants of its clock, defined by the clock of the expression of its definition. For a signal x, the expression of its definition can always be rewritten as $x := (E_1\ \texttt{when}\ h_1)\ \texttt{default} \ldots \texttt{default}\ (E_{n-1}\ \texttt{when}\ h_{n-1})\ \texttt{default}\ (x\,\$\ \texttt{when}\ k)$. If we assume that the signal keeps, between two consecutive instants, the last computed value, the assignment of $(x\,\$)$ to x is unnecessary. Then, the assignment clock is then defined by $h_1\,\hat{}{+} \ldots \hat{}{+}\,h_{n-1}$, smaller than the clock of x defined by $(h_1\,\hat{}{+} \ldots \hat{}{+}\,h_{n-1})\,\hat{}{+}\,k$.

Utility clock. The utility clock defines the instants at which a signal is necessary. For a signal x, the utility clock, $hu(x)$ is defined by:

- the clock of x if x is an input, an output, a memory, or if it is used to define an undersampling clock ($\texttt{when}\ f(x)$);
- otherwise, it is defined, for $x \to y_1\ \texttt{when}\ h_1$, ..., $x \to y_n\ \texttt{when}\ h_n$, by $\sum_{i=1,n}(hu(y_i)\,\hat{}{*}\,h_i)$.

If we rewrite the Signal program by sampling the signals (except for inputs/outputs) by their utility clock, the new Signal program is equivalent to the previous one, with respect to its behavior with the external world. Note that the utility clock can be used only when this transformation does not introduce cycles in the graph.

6.3. Formal Verification of Synchronous Dataflow Program Transformations Toward Certified Compilers

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

Translation validation [49], [48] is a technique that attempts to verify that program transformations preserve the program semantics. A compiler generally involves several phases during its compilation process. For instance, the Signal compiler [2], [8], in its first two phases, *calculates the clock information*, makes *Boolean abstraction*, and makes *static scheduling*. The final phase is the executable code generation. One can try to prove globally that the input program and its final transformed program have the same semantics. However, we believe that a better approach consists in separating the concerns and proving for each phase the preservation of different kinds of semantic properties. In the case of the Signal compiler, the preservation of the semantics can be decomposed into the preservation of clock semantics, data dependence, and value-equivalence of variables.

Translation Validation for Clock and SDGs Transformations. This work focuses on proving the preservation of clock semantics in the first two phases of the Signal compiler. In order to do that we encode the clock semantics and data dependence as *clock models* and *synchronous dependence graphs* (SDGs). Then we show that a transformation is correct if and only if there exist *refinements* between clock models, and between SDGs, written as $\Phi(P_2) \sqsubseteq_{clk} \Phi(P_1)$ and $SDG(P_2) \sqsubseteq_{dep} SDG(P_1)$ [15]. We delegate the checking of the preservation to a SMT-solver [38], [54].

Translation Validation of Polychronous Dataflow Specifications: from Signal to C using Synchronous Dataflow Value-Graphs. In this work, we build a validator for the synchronous dataflow compiler of Signal. This validator tries to match the value-graph [53] of each output of the original program and its transformed counterpart. That ensures that every output of the original program and its counterpart in the transformed program have the same value whenever they are present. Our validator does not require any instrumentation and modification of the compiler, nor any rewriting of the source program.

The Signal program and its generated C program have been represented in the same shared synchronous dataflow value-graph (SDVG), in which the nodes for the same structures (variables, constants, operators) have been shared. For instance, the values of input signals and their corresponding variables in the generated C code are represented by the same nodes in the shared graph. Then, the shared graph is transformed following *predefined rules* to show that all output signal values in the Signal program and their counterparts in the generated C code are rooted at the same subgraph.

Consider the following process, where $IR(P)$ is the compiled code of the program P and TV(SDVG(P,IR(P))) is *true* when all output signal values in P and their counterparts in $IR(P)$ are the same:
if (Cp(P) is Error) then output Error; else
if ((Φ(IR(P)) $\sqsubseteq_{clk}$ Φ(P)) and (SDG(IR(P)) $\sqsubseteq_{dep}$ SDG(P)) and (TV(SDVG(P,IR(P))))) then output IR(P); else output Error.
This will provide formal guarantee as strong as that provided by a formally certified compiler w.r.t. the clock semantics and the data dependence in case the validator is certified formally.

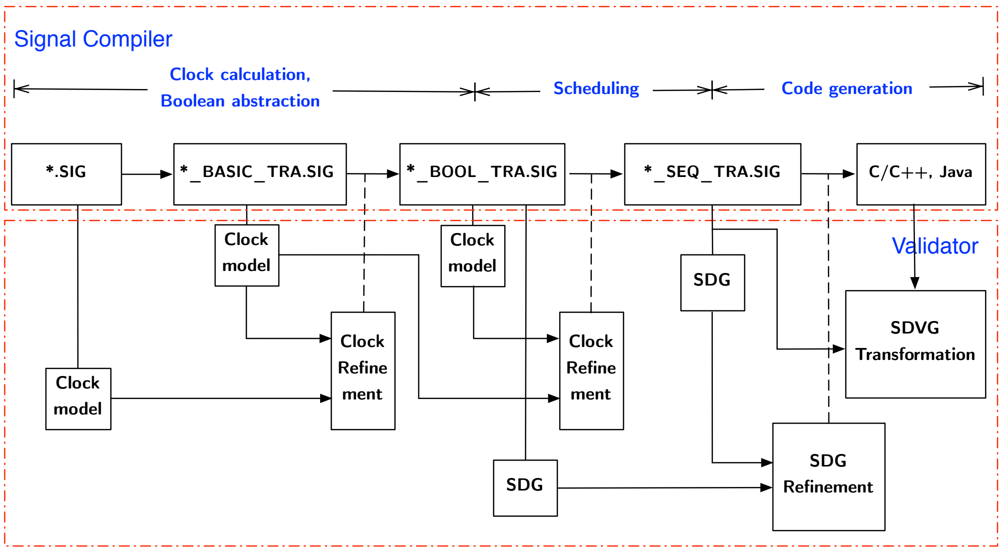

Figure 8. An overview of our integration within Polychrony toolset.

Implementation and Experiments. At a high level, our tool *SigCert* [47] developed in OCaml checks the correctness of the compilation of the Polychrony Signal compiler w.r.t clock semantics, data dependence, and value-equivalence as shown in Figure 8.

6.4. Exploring system architectures in AADL via Polychrony and SynDEx

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

Architecture analysis & design language (AADL) has been increasingly adopted in the design of embedded systems, and corresponding scheduling and formal verification have been well studied. However, little work takes code distribution and architecture exploration into account, particularly considering clock constraints, for distributed multi-processor systems. Our approach [20], [16], [17] handles these concerns within the toolchain AADL-Polychrony-SynDEx. First, in order to avoid semantic ambiguities of AADL, the polychronous/multiclock semantics of AADL, based on a polychronous model of computation, is considered. Clock synthesis is then carried out in Polychrony, which bridges the gap between the polychronous semantics and the synchronous semantics of SynDEx [42]. The same timing semantics is always preserved in order to ensure the correctness of the transformations between different formalisms. Code distribution and corresponding scheduling is carried out on the obtained SynDEx model in the last step, which enables the exploration of architectures originally specified in AADL. Our contribution provides a fast yet efficient architecture exploration approach for the design of distributed real-time and embedded systems. The approach has been illustrated using an avionic case study.

6.5. A synchronous annex for the AADL

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

We propose a synchronous timing annex for the SAE standard AADL. Our approach consists of building a synchronous model of computation and communication that best fits the semantics and expressive capability of the AADL and its behavioral annex and yet requires little to know (syntactic) extension to it, i.e. to identify a synchronous core of the AADL (which prerequisites a formal definition of synchrony at hand) and define a formal design methodology to use the AADL in a way that supports formal analysis, verification and synthesis.

Our approach first identifies the core AADL concepts from which time events can be described. Then, is considers the behavior annex (BA) as the mean to model synchronous signals and traces through automata. Finally, we consider elements of the constraint annex to reason about abstractions of these signals and traces by clocks and relations among them. To support the formal presentation of these elements, we define a model of automata that comprises a transition system to express explicit transitions and constraints, in the form of a boolean formula on time, to implicitly constraint its behavior. The implementation of such an automaton amounts to composing its explicit transition system with that of the controller synthesised from its specified constraints.

6.6. Ongoing activities and results for integration of Polychrony with the P toolset

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

Current state of P. The P language is still under definition, notably for the software/hardware architectural description of systems. In late october 2013, technical partners (headed by Adacore) released the first beta version of the toolset. The main activites of the ESPRESSO team can be splitted in analysis and development activities:

- The analysis activites consisted in understanding what tasks shall be performed ultimately by the P toolset w.r.t. code generation and architecture, and how Polychrony could be used in the proposed workflow.
- The development activities consisted in introducing a modified block sequencing algorithm in P and starting the development of the P to Signal converter.

Co-modeling in P. First, P should import functional behavior from Simulink, Stateflow and UML class, activity and state machine diagrams. Those represent a strictly sequential semantics: "the code generated from functional behaviour language will be strictly sequential and void of tasking features" (P specification [1]).

Second, imported architectural description languages are likely to be SysML, MARTE and AADL, which present concurrent semantics. Hence, "the code generated from architectural description languages may include concurrent semantics (thread, shared resources...)" (*ibid*). However, code generation from architectural description languages will consist of invocations to an underlying real-time API. The current target of code generation is the APEX ARINC-653 API, which provides real-time services like inter/intra-partition communication channels as well as task scheduling. Real-time properties of imported architectural elements, like task periods and scheduling policy, are used to configure those services.

Code distribution. Code generation should be able to distribute the functional blocks among architectural elements (processors/threads and buses/queues).

Polychrony offers a way to distribute Signal processes among different locations [31]. In general, such code distribution may lead to the synthesis of new input and output ports: when expressing synchronous communication with asynchronous protocols, some clock information might need to be added to resynchronize data-flows. Moreover, the computation model of Signal allows to order asynchronous read and write operations to avoid communication deadlock. The extended input/output interfaces of blocks could be reimported back to P in order to ensure the correctness of code distribution.

It appears however that the subset of Simulink that is imported in P, and the execution model of P functional models that is enforced by the P compiler, can be viewed as a composition of endochronous multi-rate nodes (all inputs of a node are computed before all of its outputs; this avoids deadlock problems when composing nodes). This model ends up being similar to a Lustre model of computation, where code distribution can be performed without adding communication flows and where read/write operations can be setup in a general way without introducing deadlocks [41].

Despite the above observations, it might be possible to extend the input/output interfaces of existing P models thanks to Polychrony. One approach is to ensure that block dependencies between Simulink blocks are effectively respected after code distribution. Indeed, functional blocks can be partially ordered thanks to user-defined priorities. If other partners see an interest with this approach, it could be possible to establish communication links between ordered blocks, so that the global execution order of blocks in a distributed setting is the same as the one modeled originally in the simulation environment.

Model clustering. Alternatively, it would be interesting from a Signal point of view to loosen the synchronization assumptions made by both Simulink and P so that only algebraic dependencies are taken into account (e.g. interpret all Simulink subsystems as virtual, ignore all non-strictly required dependencies...), while respecting clock constraints (e.g. sample time, controlled and enabled blocks...). In that case, the Signal compiler could perform code distribution for simulation purposes, or simply to provide another compilation scheme for P. Another step could be to apply an automatic code distribution mechanism into so-called *clusters*, and export those clusters back to P as architectural elements. The resulting P model would end-up being having possibly more tasks/threads and smaller functional blocks, which might be interesting. Those design decisions are still under consideration and must be discussed with other partners.

From P to Signal. The development activities in the P project currently consist in adding a P to Signal translator. It is being developed as a backend of the P toolset, which provides a number of facilities to access and perform computations on P models. The current prototype must be completed and refined according to what are the actual needs in the project, but can already be tested with input models.

[1] https://forge.open-do.org/plugins/moinmoin/p/

In order to validate the approach, the existing test models of the P projects are all checked with this exporter (over two hundreds small models, a couple of big ones). The resulting SSME files are then converted to Signal files: this step required to generate a command-line version of the Eclispe Polychrony product, as well a a batch converter from SSME to Signal (this converter is integrated in the Polychrony environment). In addition to convert SSME files to Signal, this converter also validates the model against Ecore constraints. Finally, the resulting Signal files are compiled with the original `C++` Signal compiler to check typing and clock relationships (those tests are not performed at the SSME level). The resulting test toolchain gives useful feedbacks for the iterative development of the translator.

Partial orders in P. The exporter also needs to export block dependencies from functional models. Since Polychrony is also able to infer a total order while taking into account code distribution, it was not satisfactory to export the existing total order computed by the P toolset: it is more sensible to export the subset that is strictly necessary (or desired). In agreement with technical partners, we modified the existing sequencer so that it could be parameterized with block ordering criteria (for example, we might want to take into account dataflow dependencies as well as user-defined priority in Polychrony, but nothing more). The outcome is a single package responsible for computing partial and total orders inside the P toolset. This prevents other tools, like the P to Signal exporter, to compute partial order by themselves.

The implementation of the sequencer is based on a dependency matrix that helps computing the transitive closure of dependencies (to quickly check whether two blocks are dependent on each other) while keeping track of their transitive reduction (in order to export only the minimal set of relationships). Now that the first version of the P toolset is released, the sequencer will hopefully be integrated in the P toolset.

6.7. Real-Time Scheduling of Dataflow Graphs

Participants: Adnan Bouakaz, Jean-Pierre Talpin.

The ever-increasing functional and nonfunctional requirements in real-time safety-critical embedded systems call for new design flows that solve the specification, validation, and synthesis problems. Ensuring key properties, such as functional determinism and temporal predictability, has been the main objective of many embedded system design models. Dataflow models of computation (such as KPN [44], SDF [46], CSDF [34], etc.) are widely used to model stream-based embedded systems due to their inherent functional determinism. Since the introduction of the (C)SDF model, a considerable effort has been made to solve the static-periodic scheduling problem [28]. Ensuring boundedness and liveness is the essence of the proposed algorithms in addition to optimizing some nonfunctional performance metrics (e.g. buffer minimization, throughput maximization, etc.). However, nowadays real-time embedded systems are so complex that real-time operating systems are used to manage hardware resources and host real-time tasks. Most of real-time operating systems rely on priority-driven scheduling algorithms [51], [37] (e.g. RM, EDF, etc.) instead of static schedules which are inflexible and difficult to maintain. Our work [12], [18], [19] [35] addresses the real-time scheduling problem of dataflow graph specifications; i.e., transformation of the dataflow specification to a set of independent real-time tasks w.r.t. a given priority-driven scheduling policy such that the following properties are satisfied: (1) channels are bounded and overflow/underflow-free; (2) the task set is schedulable on a given uniprocessor (or multiprocessor) architecture. This problem requires the synthesis of scheduling parameters (e.g. periods, priorities, processor allocation, etc.) and channel capacities. Furthermore, our work considers two performance optimization problems: buffer minimization and throughput maximization.

6.8. Structure-Preserved Distribution of Synchronous Programs

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

We propose an automatically structure-preserved distribution method, which is based on synchronous guarded actions [50] and component calls in an intermediate representation [36]. The guarded actions describe the local behavior. The component calls preserve the modular structure information of synchronous programs. Using this method, the designer can naturally blend the distribution design into the whole system design procedure: following the modular structure, a globally asynchronous locally synchronous (GALS) network over distributed nodes can be automatically constructed. Each node corresponds to a component and contains:

- a computing element, computing (as sender) or reacting to (as receiver) scheduling commands;
- a controlling element, called adaptor, adjusting the asynchronous communication between nodes.

The computing element focuses on the functional behaviors (i.e., value computation) in synchronous runs, which can be perfectly described by synchronous guarded actions. On the other hand, the controlling element mainly considers the temporal constraints (i.e., clock relation) under asynchronous communications. Guarded actions are not suitable for specifying clock relations, then we use polychronous specifications [8] to define the inter-node communications.

A perspective for future work would be the structure-preserved distribution of synchronous programs with multi-interaction. Multiple interactions in one logical instant are desynchronized and projected onto finer grained instants. Owing to this extension, it would provide more convenience for the designer to express the parallelism among components.

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 - Sept. 2015

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 [23] and CESAR [22] 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..

7.1.3. CORAC

Program: CORAC

Project acronym: CORAIL

Project title: Composants pour l'Avionique Modulaire Étendue

Duration: July 2013 - May 2017

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. International Initiatives

7.2.1. Inria Associate Teams

7.2.1.1. POLYCORE

Title: Models of computation for embedded software design of multi-core architectures

Inria principal investigator: Jean-Pierre Talpin

International Partner :

Virginia Tech Research Laboratories, Arlington (United States)

Embedded Systems Group, Teschnische Universität Kaiserslautern (Germany)

Duration: 2011 - 2013

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

Anyone experienced with multi-threaded programming would recognize 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 necessary for safe execution of such code. It is therefore desirable to synthesize multi-threaded code from formal specifications using a provably "correct-by-construction" approach. In Europe, it has been widely claimed that the embedded software for "fly-by-wire" was mostly automatically generated using French tools based on the synchronous programming models. Unfortunately, software generated in those contexts usually operate in a time-triggered execution model. Such models are simpler but less efficient than multi-threaded software on multi-core processors. Normally they run on multiple processors communicating over a time-triggered bus. Hence the execution is less efficient than it could be. While time-triggered programming model simplifies code generation, we feel that multi-rate event driven execution model is much more efficient. Code synthesis for such execution model must be thoroughly investigated. The multi-threaded software generation is inspired by a recent shift in the hardware design paradigms from single-core to multi-core processors. This shift has brought parallel and concurrent programming to the desktop and embedded arena. In the desktop market, most processors now being sold are multi-core, and very soon this trend might conquer the embedded world as well. We plan to 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 dataflow paradigm: Polychrony.

7.2.2. Inria International Partners

7.2.2.1. The University of Hong Kong, Emerging Technologies Institute

Title: Virtual prototyping of embedded software architectures

Inria principal investigator: Jean-Pierre Talpin

International Partner :

The University of Hong Kong - Emerging Technologies Institute - John Koo

Embedded software architectures are modeling objects at the crossing of several design viewpoints: the physical environment, the embedded software and the hardware architecture. These viewpoints present different perceptions of time: continuous and discrete, event-based and clock-based. They are further represented by high-level models that significantly alter this perception: in the model of the environment, evolution over time is represented by differential equations whose resolution alters discrete simulation time; in the model of the embedded software, hardware/operating-system events are sampled by periodic reaction loops; in the model of the hardware, instruction clock time is usually approximated by coarser periods or transactions. Providing a mathematical framework, verification and synthesis tools, to understand, compose and orchestrate them would prove invaluable to system architects. The architect operates from design focus point around which all components of the system under design—software, middleware, hardware and environment—need to be analyzed, profiled, composed, simulated, validated. It is the aim of our project to propose a formal design methodology to that purpose.

7.2.2.2. Beihang University, Institute of Computer Architectures

Title: Certifiable development of a synchronous compiler for multi-core platforms

Inria principal investigator: Jean-Pierre Talpin

International Partner :

Beihang University, China - Institute of Computer Architectures - Kai Hu

The synchronous paradigm is a widely accepted approach for the design of safety-critical applications, such as digital circuits or embedded software. The well-defined notions of time and causality at specification-level provide a simple way to model, analyze and verify systems. The synchronous programming paradigm is made popular because of its role at the joint point of 1) computer science and language design, 2) control theory and reactive systems, and 3) microelectronic (synchronous) circuit design. It provides a sound semantic background with a notion of discrete instants and successive reactions, together with high-level structuring primitives which help defining subthreads whose activations (defined by signals or clocks) model over/sub-sampling. Exploiting the semantic independence of various computations to allow the generation of concurrent, potentially distributed code from synchronous and polychronous specifications is a notoriously difficult subject. It amounts to determining which part of the system-wide synchronization specific to the synchronous model can be removed while preserving the specified functionality. In this context, the objective of the proposed project consists in the design of a certifiable compiler from a synchronous language to a multicore platform. However, even if the compilation of endochronous systems to a sequential architecture has been widely studied for twenty years, targeting multicore architectures is more recent and exploiting weak endochrony has not yet been deeply explored. Three main points will be addressed: the architecture of a compiler of weakly-endochronous programs to a virtual parallel machine; the formal verification of some of these compilation steps as well as the formal modeling of the target; the study of multicore platforms, of their synchronization primitives and the implementation of the virtual machine on such a platform.

7.2.3. Participation In other International Programs

7.2.3.1. USAF Office for Scientific Grant FA8655-13-1-3049

Title: Co-Modeling of Safety-Critical Multi-threaded Embedded Software for Multi-Core Embedded Platforms

Inria principal investigator: Jean-Pierre Talpin

International Partner :

Virginia Tech Research Laboratories, Arlington (United States)

Embedded Systems Group, Teschnische Universität Kaiserslautern (Germany)

Duration: 2013 - 2016

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

The aim of the USAF OSR Grant FA8655-13-1-3049 is to support collaborative research entitled “Co-Modeling of safety-critical multi-threaded embedded software for multi-core embedded platforms” between Inria project-team ESPRESSO, the VTRL Fermat Laboratory and the TUKL embedded system research group, under the program of the Polycore associate-project.

7.3. International Research Visitors

7.3.1. Visits to International Teams

- Jean-Pierre Talpin was awarded a visiting researcher grant by the Chinese Academy of Science. In this context, he visited the Shenzhen Institutes of Advanced Technology and the University of Hong Kong in January, July and August, and Beihang University in November and December.
- In the context of the associate project Polycore, Jean-Pierre Talpin visited Virginia Tech Research Laboratories, Arlington, in April and October.

8. Dissemination

8.1. Scientific Animation

- Jean-Pierre Talpin is Associate Editor with the ACM Transactions for Embedded Computing Systems (TECS); he is also Associate Editor with the Springer journal on Frontiers of Computer Science (FCS) and Associate Editor with EURASIP journal of embedded systems.
  Jean-Pierre Talpin co-chaired the program committee of the 12th International Symposium on Methods and Models for System Design (MEMOCODE'13).
  He served as program committee member of the International Conference on Embedded Systems (EMSOFT'13).
- Thierry Gautier served as program commitee member for the 2013 Electronic System Level Synthesis Conference, ESLsyn 2013 (http://www.ecsi.org/eslsyn2013).
- Loïc Besnard participated to the Inria stand at the EclipseCon France (Toulouse, June 2013).

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

- Jean-Pierre Talpin taught two conference-lectures to Graduate students at the Arlington campus of Virginia Tech in April and October.
- Jean-Pierre Talpin taught a series of fifteen MooC lectures to Master students at Beihang University, in November and December.
- Adnan Bouakaz taught three courses (Operating systems organization, Java programming, functional programming) to undergraduate students at University of Rennes 1 from September to November.

8.2.2. Juries

- Thierry Gautier served on an Associate Professor selection committee of Université de Rennes 1 in May 2013.

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, pp. 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, pp. 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^{o} 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^{o} 5, pp. 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^{o} 3-4, pp. 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^{o} 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^{o} 1-2, pp. 91–118, http://dl.acm.org/citation.cfm?id=2362088.2362094

[10] J.-P. TALPIN, P. LE GUERNIC, S. K. 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^{o} 6, pp. 613-643 [*DOI :* 10.1007/S10766-005-8907-Y], http://hal.archives-ouvertes.fr/hal-00541986

[11] 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^{o} 2, pp. 113-128 [*DOI :* 10.1016/J.SCICO.2010.06.006], http://hal.archives-ouvertes.fr/hal-00768341

Publications of the year

Doctoral Dissertations and Habilitation Theses

[12] A. BOUAKAZ. , *Ordonnancement temps-réel des graphes flots de données*, Université Rennes 1, November 2013, http://hal.inria.fr/tel-00916515

[13] A. BOUAKAZ. , *Real-time scheduling of dataflow graphs*, Université Rennes 1, November 2013, http://hal.inria.fr/tel-00945453

Articles in International Peer-Reviewed Journals

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

[15] V. C. NGO, L. BESNARD, P. LE GUERNIC, J.-P. TALPIN, T. GAUTIER. *Formal Verification of Synchronous Data-flow Program Transformations Toward Certified Compilers*, in "Frontiers in Computer Science", July 2013, http://hal.inria.fr/hal-00846279

[16] H. YU, Y. MA, T. GAUTIER, L. BESNARD, P. LE GUERNIC, J.-P. TALPIN. *Polychronous modeling, analysis, verification and simulation for timed software architectures*, in "Journal of Systems Architecture",

November 2013, vol. 59, n^o 10, pp. 1157-1170 [*DOI :* 10.1016/J.SYSARC.2013.08.004], http://hal.inria.fr/hal-00916418

[17] H. YU, Y. MA, T. GAUTIER, L. BESNARD, J.-P. TALPIN, P. LE GUERNIC, Y. SOREL. *Exploring system architectures in AADL via Polychrony and SynDEx*, in "Frontiers of Computer Science", October 2013, vol. 7, n^o 5, pp. 627-649 [*DOI :* 10.1007/S11704-013-2307-Z], http://hal.inria.fr/hal-00916445

International Conferences with Proceedings

[18] A. BOUAKAZ, J.-P. TALPIN. *Buffer Minimization in Earliest-Deadline First Scheduling of Dataflow Graphs*, in "ACM SIG- PLAN/SIGBED conference on languages, compilers and tools for embedded systems", Seattle, WA, United States, June 2013, vol. 48, pp. 133-142 [*DOI :* 10.1145/2499369.2465558], http://hal.inria.fr/hal-00916485

[19] A. BOUAKAZ, J.-P. TALPIN. *Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks*, in "International Workshop on Software and Compilers for Embedded Systems", St. Goar, Germany, June 2013, pp. 58-67 [*DOI :* 10.1145/2463596.2463600], http://hal.inria.fr/hal-00916487

[20] 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 p. , http://hal.inria.fr/hal-00763379

[21] 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

References in notes

[22] , *Cost-efficient methods and processes for safety relevant embedded systems (CESAR project)*, 2012, http://www.cesarproject.eu/

[23] , *Open Platform for the Engineering of Embedded Systems (OPEES Project)*, 2012, http://www.opees.org/

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

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

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

[27] 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, pp. 163-173 [*DOI :* 10.1145/223428.207134], http://hal.archives-ouvertes.fr/hal-00544128

[28] S. S. BATTACHARYYA, E. A. LEE, P. K. MURTHY. , *Software synthesis from dataflow graphs*, Kluwer Academic PublishersNorwell, MA, USA, 1996

[29] 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, pp. 162–177, http://hal.inria.fr/inria-00073032

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

[31] A. BENVENISTE, P. LE GUERNIC, P. AUBRY. *Compositionality in dataflow synchronous languages: specification & code generation*, in "Compositionality: The Significant Difference", Springer, 1998, pp. 61–80, http://hal.inria.fr/inria-00073379

[32] 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^o 2, pp. 103-149, http://dx.doi.org/10.1016/0167-6423(91)90001-E

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

[34] G. BLISEN, M. ENGELS, R. LAUWEREINS, J. PEPERSTRAETE. *Cycle-static dataflow*, in "Trans. Sig. Proc.", February 1996, vol. 44, n^o 2, pp. 397–408

[35] 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, pp. 183-192 [*DOI :* 10.1109/ACSD.2012.16], http://hal.inria.fr/hal-00763387

[36] J. BRANDT, K. SCHNEIDER. , *Separate Translation of Synchronous Programs to Guarded Actions*, Department of Computer Science, University of KaiserslauternKaiserslautern, Germany, March 2011, n^o 382/11

[37] R. I. DAVIS, A. BURNS. *A survey of hard real-time scheduling for multiprocessor systems*, in "ACM Comput. Surv.", 2011, vol. 43, n^o 4, pp. 35:1–35:44

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

[39] 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

[40] 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, pp. 127-149, http://hal.archives-ouvertes.fr/hal-00543824

[41] A. GIRAULT. *A survey of automatic distribution method for synchronous programs*, in "International workshop on synchronous languages, applications and programs, SLAP", 2005, vol. 5

[42] INRIA AOSTE TEAM. , *SynDEx*, 2013, http://www-rocq.inria.fr/syndex/

[43] JSR-302. , *Safety critical Java technology specification*, 2010

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

[45] 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, pp. 1321-1336 [*DOI :* 10.1109/5.97301], http://hal.inria.fr/inria-00540460

[46] E. A. LEE, D. G. MESSERSCHMITT. *Synchronous data flow*, in "Proceedings of the IEEE", 1987, pp. 1235–1245

[47] V. C. NGO. , *SigCert*, 2013, https://scm.gforge.inria.fr/svn/sigcert/trunk

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

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

[50] K. SCHNEIDER. , *The Synchronous Programming Language Quartz*, Department of Computer Science, University of KaiserslauternKaiserslautern, Germany, December 2009, n^o 375

[51] L. SHA, T. ABDELZAHER, K.-E. ARZÉN, A. CERVIN, T. BAKER, A. BURNS, G. BUTTAZZO, M. CACCAMO, J. LEHOCZKY, A. K. MOK. *Real time scheduling theory: a historical perspective*, in "Real-Time Syst.", 2004, vol. 28, n^o 2-3, pp. 101–155

[52] D. TANG, A. PLSEK, J. VITEK. *Static checking of safety critical Java annotations*, in "Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems", 2010, pp. 148–154

[53] J.-B. TRISTAN, P. GOVEREAU, G. MORRISETT. *Evaluating value-graph translation validation for LLVM*, in "ACM SIGPLAN Conference on Programming and Language Design Implementation", 2011

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