IN PARTNERSHIP WITH:
CNRS

Université Rennes 1

Activity Report 2011

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

1. Members

Research Scientists

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

Technical Staff

Loïc Besnard [Research Engineer, CNRS]

PhD Students

Adnan Bouakaz [University of Rennes 1]
Sun Ke [INRIA, since Oct. 1st.]
Chan Ngo [INRIA, since Jan. 1st.]

Post-Doctoral Fellows

Yue Ma [Expert Engineer, INRIA, since Feb. 1st.]
An Phung-Khac [Expert Engineer, INRIA, since Nov. 1st.]
Huafeng Yu [Expert Engineer, INRIA]

Administrative Assistant

Stéphanie Lemaile [Secretary, INRIA]

Other

François Fabre [Junior Engineer, INRIA, until Sep. 1st.]

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 [11]. 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) 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. Starting August 1st., 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 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 system. 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 polychromous 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).

The most effective demonstrations of these features are found in our recent collaborative projects Spacify, Opees and Cesar to equip industrial toolset 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 [5]. This affinity explains the ease of generating systems and architectures and verify their functionalities using compilers and related tools that implement this approach.

In the relational mathematical model behind the design language Signal, the supportive data-flow notation of Polychrony, this affinity goes beyond the domain of purely sequential systems and synchronous circuits and embraces the context of complex architectures consisting of synchronous circuits and desynchronization protocols: globally asynchronous and locally synchronous architectures (GALS).

This unique feature is obtained thanks to the fundamental notion of *polychrony*: the capability to describe systems in which components obey to multiple clock rates. It provides a mathematical foundation to a notion of *refinement*: the ability to model a system from the early stages of its requirement specifications (relations, properties) to the late stages of its synthesis and deployment (functions, automata).

The notion of polychrony goes beyond the usual scope of a programming language, allowing for specifications and properties to be described. As a result, the Signal design methodology draws a continuum from synchrony to asynchrony, from specification to implementation, from abstraction to refinement, from interface to implementation. Signal gives the opportunity to seamlessly model embedded systems at multiple levels of abstraction while reasoning within a simple and formally defined mathematical model.

The inherent flexibility of the abstract notion of signal handled in Signal invites and favors the design of correct-by-construction systems by means of well-defined model transformations that preserve the intended semantics and stated properties of the architecture under design.

2.4. Highlights

The main headline of 2011 is the release of the Polychrony toolset in open-source under GPL v2.0 and EPL licenses in July. It is the result of a process initiated several years ago and conducted in close collaboration with INRIA's DTI in order to precisely define the perimeter of the license and identify the best-suited licensing terms compatible with its users and potential contributors.

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 [11] consisting of a *domain* of traces or behaviors and of semi-lattice structure that renders the synchronous hypothesis using a timing equivalence relation: clock equivalence. Asynchrony can be superimposed on this model by considering a flow equivalence relation as well as heterogeneous systems [33] by parameterizing composition with arbitrary timing relations.

3.1.1. A synchronous model of computation

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

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

In the remainder, we write $\mathrm{tags}(s)$ for the tags of a signal s, $\mathrm{vars}(b)$ for the 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.

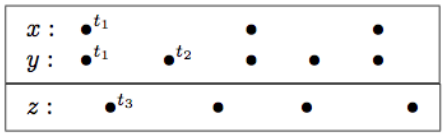

Figure 1. Behavior b over three signals x, y and z in two clock domains

3.1.1.1. Composition

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

3.1.1.2. Scheduling

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

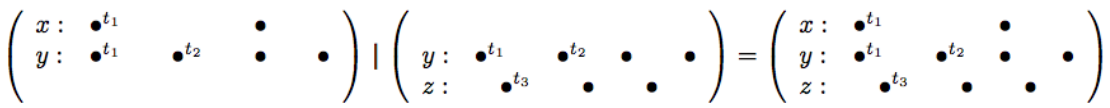

Figure 2. Synchronous composition of $b \in p$ and $c \in q$

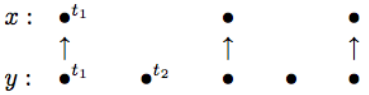

Figure 3. Scheduling relations between simultaneous events

3.1.1.3. Structure

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

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

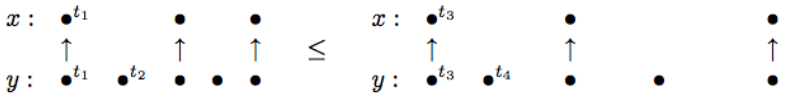

Figure 4. Relating synchronous behaviors by stretching.

3.1.2. A declarative design language

Signal [6] is a declarative design language expressed within the polychronous model of computation. In Signal, a process P is an infinite loop that consists of the synchronous composition $P \,|\, Q$ of simultaneous equations $x = y\ f\ z$ over signals named x, y, z. The restriction of a signal name x to a process P is noted P/x.

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

Equations $x = y\ f\ z$ in Signal more generally denote processes that define timing relations between input and output signals. There are four primitive combinators in Signal:

- delay $x = y \,\$\, \texttt{init}\, v$, initially defines the signal x by the value v and then by the previous value of the signal y. The signal y and its delayed copy $x = y \,\$\, \texttt{init}\, v$ are synchronous: they share the same set of tags $t_1, t_2, \cdots$. Initially, at t_1, the signal x takes the declared value v and then, at tag t_n, the value of y at tag t_{n-1}.

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

- sampling $x = y\,\texttt{when}\,z$, defines x by y when z is true (and both y and z are present); x is present with the value v_2 at t_2 only if y is present with v_2 at t_2 and if z is present at t_2 with the value true. When this is the case, one needs to schedule the calculation of y and z before x, as depicted by $y_{t_2} \rightarrow x_{t_2} \longleftarrow z_{t_2}$.
- 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 ^= count`.

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

```
| reset ^= tick ^= value
|);
```

3.1.3. Compilation of Signal

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

3.1.3.1. Synchronization and scheduling specifications

In Signal, the clock $\hat{}x$ of a signal x denotes the set of instants at which the signal x is present. It is represented by a signal that is true when x is present and that is absent otherwise. Clock expressions represent control. The clock `when` x (resp. `when not` x) represents the time tags at which a boolean signal x is present and true (resp. false).

The empty clock is written 0 and clock expressions e combined using conjunction, disjunction and symmetric difference. Clock equations E are Signal processes: the equation $e\hat{} = e'$ synchronizes the clocks e and e' while $e\hat{} < e'$ specifies the containment of e in e'. Explicit scheduling relations $x \rightarrow y$ `when` e allow to schedule the calculation of signals (e.g. x after y at the clock e).

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

3.1.3.2. Synchronization and scheduling analysis

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

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

Figure 5. Clock inference system

3.1.3.3. Hierarchization

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

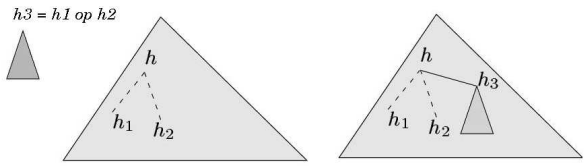

Figure 6. Hierarchization of clocks

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

3.2. Application domains

The application domains covered by the Polychrony toolbox are engineering areas where a system design-flow requires high-level model transformations and verifications to be applied during the development-cycle. The project-team has focused on developing such integrated design methods in the context of avionics applications, through the European IST projects Sacres, Syrf, Safeair, Speeds, and through the national ANR projects Topcased, OpenEmbeDD, Spacify. In this context, Polychrony is seen as a platform on which the architecture of an embedded system can be specified from the earliest design stages until the late deployment stages through a number of formally verifiable design refinements.

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 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.

4. Software

4.1. The Polychrony toolset

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 the 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 platform, a front-end to the Signal toolbox in the Eclipse environment. The SME 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.

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 INRIAGForge: 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 SME 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 www.geensoft.com).

4.2. The Eclipse interface

Participants: Loïc Besnard, Yann Glouche, Huafeng Yu, François Fabre, Yue Ma.

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 tool-chain. 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 tool-chain.

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 7) 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 [27], in the current OpenEmbeDD distribution [26], or in the TopCased distribution [28].

4.3. Integrated Modular Avionics design using Polychrony

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

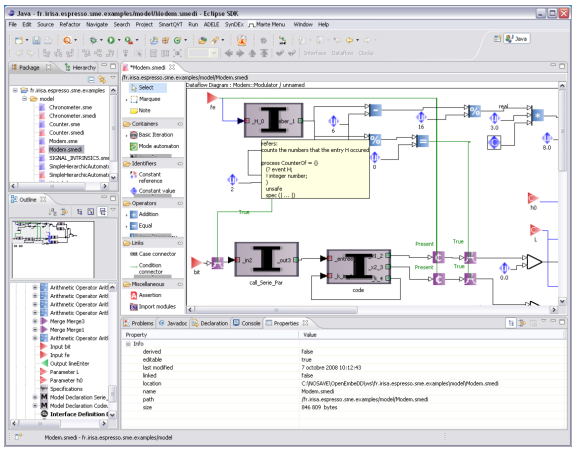

Figure 7. Eclipse SME Environment.

The Apex interface, defined in the ARINC standard [29], 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 [30]). 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 [7] is now part of the Polychrony distribution and offers a complete implementation of the Apex communication, synchronization, process management and partitioning services. Its Signal implementation consists of a library of generic, parameterizable Signal modules.

4.4. Multi-clocked mode automata

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

Gathering advantages of declarative and imperative approaches, mode automata were originally proposed by Maraninchi et al. to extend the functionality-oriented data-flow paradigm with the capability to model transition systems easily and provide an additional imperative flavor. Similar variants and extensions of the same approach to mix multiple programming paradigms or heterogeneous models of computation [36] have been proposed until recently, the latest advance being the combination of stream functions with automata in [38]. Nowadays, commercial toolsets such as the Esterel Studio's Scade or Matlab/Simulink's Stateflow are largely inspired from similar concepts.

While the introduction of preemption mechanism in the multi-clocked data-flow formalism Signal was previously studied by Rutten et al. in [51], no attempt has been made to extend mode automata with the

capability to model multi-clocked systems and multi-rate systems. In [53], we extend Signal-Meta with an inherited meta-model of multi-clocked mode automata. A salient feature is the simplicity incurred by the separation of concerns between data-flow (that expresses structure) and control-flow (that expresses a timing model) that is characteristic to the design methodology of Signal.

While the specification of mode automata in related works requires a primary address on the semantics and on compilation of control, the use of Signal as a foundation allows to waive this specific issue to its analysis and code generation engine Polychrony and clearly exposes the semantics and transformation of mode automata in a much simpler way by making use of clearly separated concerns expressed by guarded commands (data-flow relations) and by clock equations (control-flow relations).

4.5. Hyper-text source documentation of Polychrony

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

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 8.

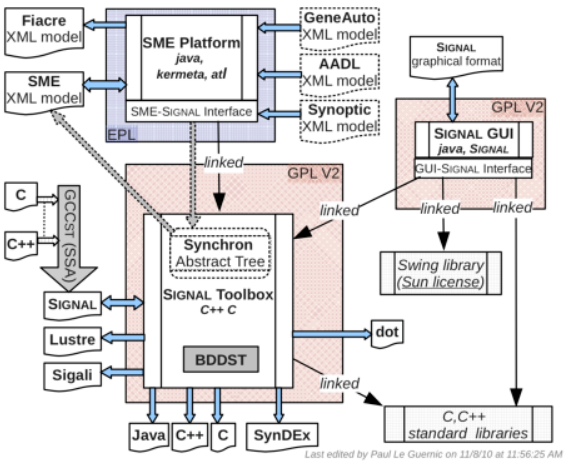

Figure 8. The Polychrony toolset high-level architecture

5. New Results

5.1. Polychrony as open-source toolset

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

A major event for us is that the open-source distribution of the Polychrony toolset has been effective since Summer 2011. The Polychrony toolset is described in Section 4.1. Following the considered part of the software, the distribution is made with the GPL V2 or EPL license. One of the objectives of this opening is to make possible a distribution of the software "by apartment" corresponding to a given functionality or to a group of functionalities, for users or developers that would be interested by only a given part of the whole software. To make this possible, a deep restructuration of the whole software has been undertaken. This takes several forms:

- One is related to the polychronous semantics and the transformations that are applied by a compilation process. A typical example is that of the representation type of the Data Control Graph (DCG), for which different levels of representation are distinguished. Some of them are based on the level of representation of the clock hierarchy.
  - The *DCGBasic* level is the general type. The DCG represents a program with all dependences set. Clocks are represented as signals of event type.
  - The *DCGPoly* level is the subtype of *DCGBasic* such that the clock hierarchy in the DCG is the result of the clock calculus. Specific clocks such as *tick* are created, but the clock hierarchy, in the general case, has several roots.
  - The *DCGEndo* level is the subtype of *DCGPoly* such that the clock hierarchy in the DCG is a tree (it is provided with a single root which is *tick*). The program is endochronous.
  - The *DCGBool* level is the subtype of *DCGEndo* such that all clock expressions are boolean extractions. Clocks are represented as Boolean signals (no event type is used). Boolean signals representing clocks have themselves clocks represented as Boolean signals (the clock hierarchy still exists).
  - The *DCGSeq* level is the subtype of *DCGBool* such that all nodes of the graph are statically sorted.
  - The *DCGFlat* level is the subtype of *DCGBool* such that the clock hierarchy in the DCG is flat: every boolean clock signal is a direct child of the *tick*. Moreover, each state variable (corresponding to delayed signals) is defined at *tick*.
- Another aspect of the reorganization is the automatic reconstruction of the toolset from basic components. For that purpose, a new tool, called *pKmake*, has been developed, that allows the architect of the software to describe its structure and construction independently of external tools (such as *emacs* that was used previously). It is especially useful for portability reasons, considering the different systems on which the toolset is provided.
- A third aspect that has required special attention is the automatic generation of the documentation of the source, which is realized using *cmake*, with an automatic management of cross-references.

In the context of the ITEA2 OPEES project, the Polychrony toolset is being provided as base component of the open-source toolchain of the Polarsys platform and Industry Working Group of the Eclipse consortium. A qualification plan will be defined in this context.

5.2. New features of Polychrony

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

Some new features have been implemented in the Signal toolbox of the Polychrony toolset:

- It is now possible to declare *virtual* objects (types, constants and process models), which are distinguished from external objects, though objects declared as external may also be redefined in the context of declaration. The actual value of an object declared as virtual is provided in the syntactic context of declaration or in a module. A module provides a context of definition for some of the objects described as virtual in the model or the module containing the module importation command. These virtual objects are *overridden* in this way if they are imported (as corresponding objects with the same name) from an imported module, or transitively, from a module imported in an imported module.

- Process models as (static) parameters have been implemented: the formal parameters of the interface of a process model can contain process model parameters, that appear as a formal name of process model typed with a process model type. The call of a process model sets up an expansion context in which an effective process model, designated by its name, is associated with each formal model.
- The connection to the SynDEx tool (http://www.syndex.org/) has been completed as follows. So far, only the functional part of a given application described in Signal was translated as a corresponding "algorithm" in SynDEx. The multicomponent architecture (typically, processors interconnected through communication medias) and the mapping of the algorithm onto the architecture had to be provided directly within SynDEx. As the polychronous model may be used as intermediate common formalism for applications described in languages where these aspects may be specified (this may be the case in AADL, for instance), they have to be taken into account in the translation. Thus, required elements of the architecture and distribution constraints are described using specific "pragmas" in Signal. These features are then translated into the SynDEx formalism. Using all these information, SynDEx can explore the possible implementations of the algorithm onto the multicomponent.

Moreover, we have redefined the meta-model of Signal in Ecplise, now called SSME (for Signal Syntax meta-model under Eclipse). The SME meta-model, that was used previously, suffered from several drawbacks. It was not fully complete in some parts of the language and, due to design choice, required a strict separation between clock and data flow relations. Thus specific program transformations had to be applied, which did not facilitate traceability. SSME is a full syntax oriented meta-model of Signal, very close to the abstract syntax that is used in the Signal toolbox. Compared to SME, this facilitates model transformations and traceability requirements, which are the primary objectives for its use. The transformation of AADL models into Signal, for instance, now uses the SSME meta-model.

5.3. Extensions of the language and the model

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

The different works on using the polychronous model as semantic median model (which has also a syntactic instance) for different effective models (AADL, Simulink via GeneAuto, UML via CCSL...) lead us to study various possible extensions of the semantic model as well as the syntactic one. Some of them have already been defined while for others, the study is still ongoing. In particular, we plan to add to the Signal language a new syntax for automata, partly inspired from AADL mode automata and hierarchic automata existing in other formalisms. An automaton is considered as an instance of a new process model and the "and" composition is the Signal composition.

A fundamental issue that we wish to address in a new way is that of globally asynchronous, locally synchronous (GALS), or globally asynchronous, locally polychronous systems. The idea we have is to extend Signal with a syntactic structure that encapsulates a polychronous (or synchronous) process P in a *system*, S, that creates a continuous temporal domain providing a real-time clock presented in different time units (..., *fs*, ..., *ms*, ..., *sec*, *mn*, ...). Such a real-time clock can be used as a usual "synchronous" 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 standard Signal constraints in the system S on these different signals, to express for instance some variation limits of different clocks.

We have also started a new work on causality aspects in order to express and operate more elaborate dependencies than instantaneous dependencies currently computed on the graph of a program. This theoretical work allows one to express dependencies that cross several instants, in a formal framework of word automata and graph algebra.

5.4. Source to source traceability in Polychrony

Participants: Loïc Besnard, François Fabre, Thierry Gautier.

To fulfill a mandatory requirement for adoption and qualification of Polychrony environment on the open-source industrial platform of the Polarsys IWG, we have integrated source to source traceability features into the Polychrony toolset. The implementation of traceability is based on the definition of structures of data and algorithms allowing to follow the transformation of objects since the Eclipse modeler of the SME Platform until the generated code. These elements have a direct application with our industrial partners, as, for example, Geensoft with whom, within the framework of the ANR project Spacify, we implemented a simulator of embedded software for satellite applications. We have also integrated such a simulator mode in the Polychrony toolset. Moreover, the error messages from the Signal compiler (Signal Toolbox) are now directly visible on the SME Graphical User Interface and on the Synoptic model (Synoptic is a satellite domain-specific modeling language).

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

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

Clock Constraint Specification Language (CCSL) [32] is defined in an annex of the UML MARTE profile [48]. We are interested in the analysis, synthesis and code generation of multi-clocked/polychronous systems specified in CCSL. Timed systems subject to clock expressions or relations can be modeled, specified, analyzed, and simulated within the software environments, such as SCADE [41], TimeSquare [44] and Polychrony. However, code generation from a multi-clocked system is far from obvious. For instance, SCADE always uses a reference or master clock (the fastest); all clocks and all conditions are defined as a functional sampling of this master clock, from the highest specification down to the lowest generated code. In TimeSquare, clock constraints are solved using a heuristic algorithm, which is generally non-deterministic. On the contrary, in Polychrony, a formally defined refinement process yields to the generation of (sequential or concurrent) code by the addition of control variables to get a deterministic behavior satisfying the constraints and allowing the desired amount of concurrency.

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[22]. Yet, our work considers a novel approach with regards to previous approaches: to generate executable specifications by considering discrete controller synthesis (DCS) [50], [45], [46]. Clock constraint resolution is addressed by DCS, which does not necessarily require a master clock to address polychronous clocks. In our approach, polychronous (CCSL) specifications are first partitioned: clock relations are considered as control objectives, other constraints are considered as the system to be controlled. The all the constraints are translated into, via SIGNAL, polynomial dynamical systems (PDSs). A PDS represents the transition system of a specification as well as the constraints (invariants) it must satisfy. The Sigali tool is then used to generate the controller. Finally, the generated controller, together with the original system, is composed to complete the code generation for simulation. In our approach, the temporal semantics of CCSL is mapped onto a polychronous model of computation, on which effective synthesis is carried out to meet constraint requirements. This approach provides both a useful mapping in theory and a flow, which is practical in the generation of reactive controllers.

5.6. The CESAR demonstrator and reference technology platform

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

The design of embedded systems from multiple views and heterogeneous models is ubiquitous in avionics as, in particular, different high-level modeling standards are adopted for specifying the structure, hardware and software components of a system. The system-level simulation of such composite models is necessary but difficult task, allowing to validate global design choices as early as possible in the system design flow. Inspired by the Ptolemy [40], MoBIES [31], SML-Sys [47], etc., we propose an approach to the issue of composing, integrating and simulating heterogeneous models in a system co-design flow [21]. First, the functional behavior of an application is modeled with synchronous data-flow and Statechart diagrams using Simulink/Gene-Auto [54], [55]. The system architecture is modeled in the AADL standard [52]. These high-level, synchronous

and asynchronous, models are then translated into a common model, based on a polychronous model of computation, allowing for a Globally Asynchronous Locally Synchronous (GALS) interpretation of the composed models. This translation is implemented as an automatic model transformation within Polychrony. Simulation, including profiling, value change dump demonstration [24], Syndex adequation [43], etc., is carried out based on the common model within Polychrony.

Polychrony has been integrated to the Reference Technology Platform (RTP) V2 and V3 of CESAR to serve as a framework for co-modeling and architecture exploration. ModelBus [49] is used for the integration of Polychrony into the RTP. ModelBus [25], an integration platform based on Service-Oriented Architecture (SOA), connects different services offered by tools connected to ModelBus. In the demonstration, we participated in the pilot application of Sub-Project 3 (SP3), whose aim is to use the RTP to define a complete software design flow for the doors management system (DMS) of an Airbus A350 in the framework of ModelBus. In the pilot application of the DMS, functional components are modeled in the synchronous model of computation of Simulink, whereas the architecture is modeled in the asynchronous model of computation of AADL[14], [18]. These high-level models are transformed into Signal programs via SME models. Additional models, which are used in the simulation of a closed system, are coded manually in Signal and synchronously composed with the Signal programs transformed from Simulink and AADL models. Finally, C or Java code is generated from Signal programs. Simulation can then be carried out for the purpose of performance evaluation and VCD (Value Change Dump) based demonstration in RTP V2. In RTP V3, Syndex adequation is also integrated to demonstrate real-time scheduling and distribution. Our whole model transformation and simulation chain has been implemented with Galileo Eclipse and attached to ModelBus as a provider of registered remote service. This demonstration also shows the integration of Polychrony with other tools, such as OSATE (AADL), Simulink, Gene-Auto, TimeSquare, ATL, Kermeta, etc.

5.7. Modeling AADL in a polychronous model of computation

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

Architecture Analysis and Design Language (AADL) is an SAE standard aimed at high level design and evaluation of architecture of embedded systems. We are interested in the analysis, simulation and verification of timed systems specified in AADL. Polychrony is well suited for the GALS architecture, and it enables deterministic specifications and formal analysis for the design of safety-critical systems. In order to benefit from the advantages provided by Polychrony, a proposition of a methodology for system-level modeling and validation of embedded systems specified in AADL via the polychronous model of computation is proposed.

By studying the different timing semantics of AADL and Polychrony, we have proposed an approach that automatically translates AADL models to a polychronous model of computation (SSME model). In the Polychrony framework, the Signal program can be generated, and an executable model can be obtained. The systems can be analyzed by tools and technologies associated with Polychrony allowing early simulation, testing and verification.

We implemented a plug-in for Eclipse framework to perform model transformation from AADL to SSME (new meta model of Signal). This transformation is implemented in Java. The following new features have been developed this year:

- Temporal interpretation of AADL model. Due to the different timing semantics between AADL and Signal, we keep the ideal view of instantaneous computations of polychronous model, moving computing latencies and communication delays to specific memory process, that introduce delays and well suited synchronizations. Each component modeled in Polychrony is composed of a behavior process (which models the functional behaviors) and a property process (which models the temporal properties).
- Architecture restructures. The architecture of the transformation is optimized. Functional architecture and meta architecture are described to give a global view of the transformation. The translation is recursive. Each AADL component is separated into a java class. The hierarchy of classes are reserved.

- Library developments. We define a Signal library containing the Signal process models representing some basic AADL concepts.
- Documentation. A new technical documentation of the transformation from AADL to SSME has been developed to accompany its implementation. This document aims to provide a global view of our implementation, from a high-level structural view to low-level implementation technical details of components.
- Programming language updates. This version of model transformation uses Java as the programming language. It avoids the disadvantages of dependent on other model transformation languages, and it provides more conveniences and flexibility. The new version is integrated as a plug-in in the Eclipse platform.
- Papers published. Three papers [14], [18], [21] are published this year.

5.8. Composing Simulink and AADL

Participants: An Phung-Khac, Jean-Pierre Talpin, Benoit Combemale, Jean-Marc Jezequel.

The goal of this work is to improve an import function of the Polychrony environment proposed by the team. Particularly, Polychrony comprises a co-modeling tool supporting the import a high-level Simulink (functional) and AADL (architectural) specifications [21]. This import function is currently implemented by two different transformations, namely Simulink-to-Signal, and AADL-to-Signal. To integrate the Signal programs resulting from these transformations, some Signal interfaces are manually implemented. The composition of Simulink and AADL models thus depends on system designers who implement the interfaces, making difficult its maintenance and validation. To deal with this issue, the model composition approach proposed by the Triskell team, namely ModMap [37], could be used to build a new Simulink and AADL model composition framework.

In ModMap, model composition is considered as a pair of a mapping and an interpretation. A mapping aligns concepts of two meta-models, while the interpretation describes the composition goal. As a model mapping framework, ModMap provides an extensible modeling language supporting the definition of generic mappings and the definition of interpretations. Together with this language, the ModMap kernel is also implemented as an extensible set of mapping processing functions. Model composition frameworks are then built by extending the language and the kernel according to specific composition purposes.

As mentioned above, we intend to apply the ModMap approach to the development of the Simulink and AADL model composition framework. To this end, we need to extend the ModMap mapping language to obtain an other one that allows system designers to align elements between Simulink and AADL models regarding the purpose of co-simulation in Signal. Then, a transformation, namely ModMap-to-Signal, needs to be implemented by extending the ModMap kernel. This transformation uses mappings provided by system designers as inputs to generate Signal interfaces. The three transformations (i.e., Simulink-to-Signal, AADL-to-Signal, and ModMap-to-Signal) form the new model composition framework. Comprared to the previous one, this framework will more automated. On the other hand, existing transformations will also be reused.

5.9. From affine-related dataflow models to Safety-critical Java

Participants: Adnan Bouakaz, Jean-Pierre Talpin, Jan Vitek.

The objective of this work is to investigate a dataflow concurrency model in order to help specifying, analyzing, and synthesizing functionally deterministic and schedulable SCJ applications. Indeed, the SCJ shared-memory concurrency model makes proving functional determinism and schedulability of applications quite hard if not impossible.

The new model is called the firing related dataflow (FRDF) model in which actors are connected to each other by means of bounded channels. The operational semantics of this model is based on the notion of firing relations. Each actor is associated with a firing clock (an infinite set of activation ticks). The proportionality of the rates of two clocks is expressed by a firing relation. A special and enough expressive case of firing relations is the class of affine relations. Some results about the canonical form of affine relations are already developed by the ESPRESSO team.

Our first study was about synthesizing affine relations between firing clocks in such a way that overflow and underflow exceptions cannot occur during execution. This synthesis is conducted by minimizing the overall of buffer sizes. It is proven that the operational semantics of the dataflow graph based on the computed affine relations is equivalent to the Kahn semantics. This implies that functional determinism is guaranteed.

The previous analysis step (called affine relations synthesis) aims to produce an abstract schedule of the dataflow graph. The computed schedule is abstract in the sense that it is independent from the implementation code of actors and from the target machine. Executing the graph on a mono-processor system using EDF scheduling algorithm is investigated in our study. We synthesize the timing characteristics of each actor (i.e. its period and phase) in such a way the set of tasks is schedulable. In this timing synthesis, we use the worst-case execution times computed from the Java implementation code of actors.

Our objective is to automatically generate a SCJ application from a dataflow specification. Currently, we work on increasing the expressivity of the underlying dataflow model together with providing the necessary analysis tool for generating deterministic and schedulable SCJ code.

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

Participants: Van-Chan Ngo, Jean-Pierre Talpin, Loïc Besnard.

Synchronous languages such as SIGNAL have been introduced and used successfully for the design and implementation of embedded and critical real-time systems. They rely on the fact that programs are modeled as data-flow equations or finite state machines that allow formal reasoning on designs. In consequence of that, a full toolset of synchronous languages provides formal transformation, automatic code generation, formal verification...

In general, the synchronous language's compiler takes several translations from the source program before generating the target code (e.g. C/C++ or Java code), thus we present an approach to verify these translations of synchronous language compiler. Our approach adopts the translation validation notion [49]. The idea of translation validation is the following: rather than proving in advance that the compiler always produces correct translations, each individual translation (e.g. every run of the compiler) is followed by a validation phase which verifies that the final output of this run correctly implements the input source program. This method avoids the drawback of freezing the potential improvements and/or developments of the compiler of the traditional compiler verification. For every small change in the compiler, the verification must be redoing the proof, that is an extremely complex task.

The validation phase is made automated which consists of: *(i)* Represent both the input source and output target SIGNAL programs as *Polynomial Dynamical Systems - PDSs*. *(ii)* Propose a *refinement* relation for the PDS models of the source and target programs. *(iii)* Use a syntactic simulation-based proof method which automatically verifies the refinement. This automated proof is done by extending the functionality of the model checker SIGALI in the *Polychrony* toolset

5.11. PDSs for translation validation: from SIGNAL to C

Participants: Van-Chan Ngo, Jean-Pierre Talpin, Loïc Besnard.

Synchronous programming languages provide a formal and abstract model of concurrency to facilitate the implementation of concurrent embedded software by automating the most complex tasks of verification, validation and code generation. They also guarantee the reliability of the design/implementation of concurrent embedded software by providing either the proof of compiler's correction or the validation of each run of the compiler. Adopting the translation validation approach [49], we provide an automatic process to formally verify the code C generation task of the SIGNAL's compiler.

The verification framework will take the SIGNAL program and the generated C code program as the input and proves whether the generated C code correctly implements the SIGNAL program. It also allows to automatically generate the refinement and counterexamples of the generated C code.

Polynomial dynamical system - PDS is used as a common semantic framework to model the behavior of both the SIGNAL program and its generated C code. First, the generated C code is translated into the target SIGNAL program [34] thanks to the intermediate SSA forms. An appropriate relation called *refinement* for PDSs is proposed to represent the correct implementation relation between the SIGNAL program and its generated C code. The generated code C correctly implements the SIGNAL program if and only if there is a refinement for their PDSs and we say that the generated C code's PDS refines the SIGNAL program's PDS. A proof method which allows to generate the refinement or counterexamples, and then proposes a refining process for the generated C code.

5.12. Synchronous symbolic translation systems for translation validation

Participants: Van-Chan Ngo, Jean-Pierre Talpin.

We propose a framework for verification of the correct implementation of the SIGNAL compiler's generation code task. In order to present the formal semantics of SIGNAL and generated code programs we introduce *synchronous symbolic transition system (SSTS)* which is the computational model of our formal verification approach. We denote $\mathcal{D}_\mathcal{V} = \prod_{i\in[1,n]} \mathcal{D}_{v_i}$ as the domain of a set of variables $\mathcal{V} = (v_1, ..., v_n)$. A set of states $P \subset \mathcal{D}_\mathcal{V}$ is defined as a *predicate* over the set of variables $\mathcal{V}$ such that the predicate is held in P. An *assignment* A is a function $A : \mathcal{D}_\mathcal{V} \mapsto \mathcal{D}_\mathcal{V}$ that the values of the variable set $\mathcal{V}$. A SSTS is a tuple $L = (\mathcal{V}, \Theta, \Gamma, \mathcal{E})$ where:

- $\mathcal{V} = (v_1, ..., v_n)$ is a set of variables,
- $\Theta \subseteq \mathcal{D}_\mathcal{V}$ is a predicate on $\mathcal{V}$ defining the initial condition on the variable set,
- Γ is a finite set of symbolic transitions $\gamma = (P_\gamma, A_\gamma)$ where:
  - $P_\gamma \subseteq \mathcal{D}_\mathcal{V}$ is a predicate on $\mathcal{V}$, which guards γ
  - $A_\gamma : \mathcal{D}_\mathcal{V} \to \mathcal{D}_\mathcal{V}$ is the assignment function of γ
- $\mathcal{E} \subseteq \mathcal{V}$ is a set of *externally observable variables*.

The generated code correctly implements the SIGNAL program if and only if there is a *refinement* for their SSTSs and we say that the generated code's SSTS refines the SIGNAL program's SSTS. This framework also works with SIGNAL programs which is considered as infinite state systems. To obtain the verification results, we apply abstraction interpretation techniques [39] which provide over-approximations of the *refinement* relation between the input SIGNAL program's model and the output generated code's model.

5.13. An integrated environment for Esterel/Quartz and Polychrony/Signal

Participants: Jens Brandt, Ke Sun, Jean-Pierre Talpin.

The design of modern embedded software architectures relies on models and programs built and reused from engineering teams with specific skills and know-how. Each of these skills and backgrounds correspond to specific tools and processes that help implement the viewpoint under consideration with mathematically grounded foundations.

It is not uncommon, for instance, that the design of the only functional views of a system may require the use of tools as heterogeneous and exotic as Catia, Scade, Matlab or Rhapsody. The same holds for design objectives that may range from that of mapping the functional design on specific hardware architectures to that of virtual prototyping for simulation or performance or energy usage evaluation.

Co-modeling itself encompasses the variety of engineering activities that cross the border between the functional and physical views of system design. It is typically the system architects, who will put together functional components and explore different metrics for an effective and efficient mapping on target systems.

We wish to further and scale the framework and experiments developed within the CESAR and OPEES projects in that respect, by thinking a new, domain-specific language, built from synchronous modules designed with Quartz, an imperative synchronous programming language, and connected by data-flow networks described in Signal, the polychronous data-flow language at the core of Polychrony. The combination of viewpoints or paradigms offered by these two design environments provides powerful abstractions and easy to use concepts in order to address two design challenges of utmost importance:

- To provide a natural and dependable specification of elementary synchronous functionalities, most of them algorithmic and control-intensive, in the imperative framework offered by Quartz.
- To synthesize the scheduling of computations and communications among these functionalities starting from the multi-clocked synchronous abstractions offered by the Signal data-flow language.

The remaining long-term goal will then be homogenize this programming framework by further extending it with the capability to control polychromous networks, seen as modes of execution, with a Quartz module, which would control mode changes.

5.14. Quality assessment and qualification of Polychrony on the open-source Polarsys IWG platform

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

Since the open-source release of Polychrony and in the context of the ITEA2 OPEES project, we are collaborating with CS to the integration of Polychrony on the Polarsys platform. This integration proceeds according to guidelines and requirements under definition within the OPEES project and aims first, at putting them to the test. The qualification process of Polychrony in Polarsys consists of checking the maturity level of its implementation and documentation (a standard software engineering assessment) but is also concerned with its capabilities to be composed, inter-operated and mapped with other components on the platform to form a application-specific design toolchain, by using model-driven engineering technologies for model transformation and orchestration. The last phase of this assessment is with regards to its qualifiability, as a simulation tool, as a verification tool, as a code generation tools, which needs to adhere standards such as these defined the in DO178 documents. In parallel, the quality assessment of Polychrony is complemented with a case study, of the APOTA network protocol, whose aim is to document, as a tutorial, the use and added-value of the toolset for its future users on the Polarsys platform.

6. Contracts and Grants with Industry

6.1. Artemisia project CESAR

Participants: Huafeng Yu, An Phung-Khac, Yue Ma, Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin.

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 will cover 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 aim at 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.

6.2. ITEA2 project OPEES

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

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 is in the process of becoming an Industrial Working Group of the Eclipse consortium, Polarsys, whose goal will be 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 is 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 is currently under a quality assessment performed in collaboration with CS.

6.3. ANR project VERISYNC

Participants: Loïc Besnard, Chan Ngo, Jean-Pierre Talpin.

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.

6.4. FUI project P

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

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/ 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 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. Partnerships and Cooperations

7.1. National Actions

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

7.1.1. ONERA/Thales TORRENTS working group

Team Espresso participates to the TORRENTS working group since its inaugural seminar in 2010. TORRENTS is a federates the activities related to time-oriented embedded systems being primarily carried out in research labs in Toulouse. It is supported by the RTRA STAE foundation. TORRENTS aims at proposing a methodology for time-based design of embedded aerospace real-time systems.

http://www.irit.fr/torrents

7.2. European Actions

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

7.2.1. Network of excellence ARTIST2

The Espresso project-team participates to the Artist2 network of excellence. Detailed presentations on the aim and scope of the network can be found in the book [1] and the website http://www.artist-embedded.org/FP6 of the project. In particular, we have contributed to a survey of real-time programming languages edited by Alan Burns [42].

7.3. International collaborations

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

7.3.1. INRIA associate project POLYCORE

In the frame of three consecutive joint NSF-INRIA and INRIA associated project programs, together with additional funds from INRIA scientific direction, INRIA-Rennes, the University of Rennes, the ARTIST NoE, we have established a long-lasting and scientifically fruitful collaboration with the Fermat Laboratory at Virginia Tech (Pr. Sandeep Shukla) and UC San Diego (Pr. Rajesh Gupta). The collaboration started in 2002 and was prolonged until 2009 with the one-year sabbatical of Sandeep Shukla as invited professor. This collaboration resulted in the joint publication of 10 scientific books and series volumes as well as 22 international journal and conference articles. In the frame of this collaboration, we jointly created the ACM-IEEE MEMOCODE (http://www.memocode-conference.com) international symposium series as well as the FMGALS international workshop series. Finally, we jointly organized four tutorials. This series of collaborations resulted in a technology transfer of the Polychrony toolset with the launch of the project CodeSyn at Virginia Tech, funded by the US Air Force Research Laboratories (AFRL), and now employs one of our former post-doctorates, Julien Ouy.

Our collaboration is now been renewed in the frame of the 2011 INRIA Associate Project POLYCORE and extended to a key additional partner, the Embedded System Group of Pr. Klaus Schneider at TU Kaiserslautern.

Our joint project 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.

In Europe, it has been widely claimed that the embedded software for ’fly-by-wire’ was mostly automatically generated using tools based on the synchronous programming models. Unfortunately, software generated in those contexts usually operate in a time-triggered execution model. Such models are simple but way less efficient than multi-threaded software when run on multi-core processors, just because of the periodic synchronization overhead.

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 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 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 (whom we submitted the white-paper of a project proposal for funding in 2012).

8. Dissemination

8.1. Invited Lectures

Jean-Pierre Talpin gave a one day lecture at the 2011 SIAT International Summer School on Embedded Systems (ISSES'11) on "polychronous systems modeling and automated code generation". Jean-Pierre Talpin gave an invited talk at the Ecole d'été temps-réel 2011 (ETR'11) on "System-level co-simulation of embedded software architectures in a polychronous moel of computation".

8.2. Visits

Pr. Kai Hu (Beihang University, Beijing) visited ESPRESSO in summer 2011 with the support of the University of Rennes 1.

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.

8.3. 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

Thierry Gautier served as technical program committee member for the conference MEMOCODE'11 http://research.microsoft.com/en-us/um/cambridge/events/memocode2011 and ESLSYN'11 http://www.ecsi.org/eslsyn.

8.4. Teaching

- Thierry Gautier taught on formal methods for component and system synthesis at the Master 2 Graduate program of the University of Rennes 1.

9. Bibliography

Major publications by the team in recent years

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

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

[3] T. P. AMAGBEGNON, L. BESNARD, P. LE GUERNIC. *Implementation of the Data-flow Synchronous Language Signal*, in "Proceedings of the ACM Symposium on Programming Languages Design and Implementation (PLDI'95)", ACM, 1995, p. 163–173.

[4] A. BENVENISTE, B. CAILLAUD, P. LE GUERNIC. *From synchrony to asynchrony*, in "CONCUR'99, Concurrency Theory, 10th International Conference", J. C. M. BAETEN, S. MAUW (editors), Lecture Notes in Computer Science, Springer, August 1999, vol. 1664, p. 162–177.

[5] A. BENVENISTE, P. CASPI, S. EDWARDS, N. HALBWACHS, P. LE GUERNIC, R. DE SIMONE. *The Synchronous Languages Twelve Years Later*, in "Proceedings of the IEEE Special issue on Modeling and Design of Embedded Systems", 2003, vol. 91(1).

[6] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT. *Synchronous programming with events and relations: the Signal language and its semantics*, in "Science of Computer Programming", 1991, vol. 16, p. 103-149.

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

[8] T. GAUTIER, P. LE GUERNIC. *Code generation in the SACRES project*, in "Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99", Huntingdon, UK, F. REDMILL, T. ANDERSON (editors), Springer, February 1999, p. 127–149.

[9] P. LE GUERNIC, T. GAUTIER. *Data-Flow to von Neumann: the Signal approach*, in "Advanced Topics in Data-Flow Computing", J. L. GAUDIOT, L. BIC (editors), 1991, p. 413–438.

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

[11] P. LE GUERNIC, J.-P. TALPIN, J.-C. LE LANN. *Polychrony for system design*, in "Journal of Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design", 2003.

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

[13] J.-P. TALPIN, J. OUY, T. GAUTIER, L. BESNARD, P. LE GUERNIC. *Compositional design of isochronous systems*, in "Science of Computer Programming", 2010.

Publications of the year

Articles in International Peer-Reviewed Journal

[14] Y. MA, T. GAUTIER, J.-P. TALPIN, P. LE GUERNIC, H. YU. *Modélisation compositionnelle d'architectures GALS dans un modèle de calcul polychrone*, in "Journal Européen des Systèmes Automatisés", November 2011, http://hal.inria.fr/hal-00639589/en/.

[15] D. POTOP-BUTUCARU, Y. SOREL, R. DE SIMONE, L. BESNARD, J.-P. TALPIN. *From concurrent multi-clock programs to concurrent multi-threaded implementations*, in "Fundamenta Informaticae", 2011.

International Conferences with Proceedings

[16] A. BOUAKAZ, I. PUAUT, E. ROHOU. *Predictable Binary Code Cache: A First Step Towards Reconciling Predictability and Just-In-Time Compilation*, in "The 17th IEEE Real-Time and Embedded Technology and Applications Symposium", Chicago, United States, Marco Caccamo, April 2011, http://hal.inria.fr/inria-00589690/en.

[17] J. BRANDT, M. GEMUNDE, K. SCHNEIDER, S. SHUKLA, J.-P. TALPIN. *Integrating System Descriptions by Clocked Guarded Actions*, in "Forum on Design Languages", September 2011.

[18] Y. MA, H. YU, T. GAUTIER, J.-P. TALPIN, L. BESNARD, P. LE GUERNIC. *System Synthesis from AADL using Polychrony*, in "Electronic System Level Synthesis Conference", June 2011, http://hal.inria.fr/inria-00594943/PDF/eslsyn11-ma.pdf.

[19] V. PAPAILIOPOULOU, D. POTOP-BUTUCARU, Y. SOREL, R. DE SIMONE, L. BESNARD, J.-P. TALPIN. *From concurrent multi-clock programs to concurrent multi-threaded implementations*, in "Electronic System Level Synthesis Conference", San Diego, California, United States, June 2011, http://hal.inria.fr/inria-00578585/en.

[20] Z. YANG, J.-P. BODEVEIX, L. PI, D. MA, J.-P. TALPIN. *Two formal semantics for a subset of the AADL*, in "UML&AADL workshop at the IEEE International Conference on Engineering of Complex Computer Systems", 2011.

[21] 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, Taiwan, Province Of China, March 2011, http://hal.inria.fr/inria-00536907/en.

[22] 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, United Kingdom, July 2011, http://hal.inria.fr/inria-00594942/en.

Research Reports

[23] V. PAPAILIOPOULOU, D. POTOP-BUTUCARU, Y. SOREL, R. DE SIMONE, L. BESNARD, J.-P. TALPIN. *From concurrent multi-clock programs to concurrent multi-threaded implementations*, INRIA, March 2011, n^{o} RR-7577, http://hal.inria.fr/inria-00578585/en.

References in notes

[24] *IEEE Standard for Verilog Hardware Description Language (VHDL)*, 2006, IEEE Std. 1364 - 2005.

[25] *ModelBus*, http://www.modelbus.org.

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

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

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

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

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

[31] R. ALUR, T. DANG, J. ESPOSITO, Y. HUR, F. IVANCIC, V. KUMAR, I. LEE, P. MISHRA, G. PAPPAS, O. SOKOLSKY. *Hierarchical modeling and analysis of embedded systems*, in "Proc. IEEE", 2003, vol. 91, n^{o} 1, p. 11–28.

[32] 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.

[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] 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 "In Proceedings of the 9th Workshop on Automated Verification of Critical Systems", 2009.

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

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

[37] M. CLAVREUL, O. BARAIS, J.-M. JÉZÉQUEL. *Integrating legacy systems with MDE*, in "Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2", New York, NY, USA, ICSE '10, ACM, 2010, p. 69–78, http://doi.acm.org/10.1145/1810295.1810306.

[38] J.-L. COLACO, B. PAGANO, M. POUZET. *A conservative extension of synchronous data-flow with state machines*, in "In Embedded Software Conference.", ACM Press, 2005.

[39] P. COUSOT, R. COUSOT. *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in "In POPL'77", 1977, p. 238-252.

[40] J. EKER, J. JANNECK, E. LEE, J. LIU, J. LUDWIG, S. NEUENDORFFER, S. SACHS, Y. XIONG. *Taming Heterogeneity: the Ptolemy Approach*, in "Proc. IEEE", 2003, vol. 91, n° 1, p. 127–144.

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

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

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

[44] INRIA AOSTE TEAM. *TimeSquare*, http://www-sop.inria.fr/aoste/dev/time_square.

[45] 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.

[46] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment*, in "Discrete Event Dynamic System: Theory and Applications", October 2000, vol. 10, n° 4, p. 325–346.

[47] D. MATHAIKUTTY, H. PATEL, S. SHUKLA, A. JANTSCH. *SML-Sys: a functional framework with multiple models of computation for modeling heterogeneous system*, in "Design Automation for Embedded Systems", 2008, vol. 12, p. 1–30.

[48] OBJECT MANAGEMENT GROUP (OMG). *Modeling and Analysis of Real-time and Embedded systems (MARTE), v1.0*, November 2009, Document number: formal/2009-11-02, http://www.omg.org/spec/MARTE/1.0/PDF/.

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

[50] P. J. RAMADGE, W. M. WONHAM. *The Control of Discrete Event Systems*, in "Proceedings of the IEEE, Special issue on Dynamics of Discrete Event Systems", 1989, vol. 77, n° 1, p. 81–98.

[51] É. RUTTEN, F. MARTINEZ. *Signal GTI: implementing task preemption and time intervals in the synchronous data flow language Signal*, in "Proceedings of the 7th Euromicro Workshop on Real-Time Systems, Odense, Denmark", IEEE Publ., june 1995.

[52] SOCIETY OF AUTOMOTIVE ENGINEERS (SAE). *Architecture Analysis Design Language (AADL, SAE standard ASS5506)*, http://www.sae.org.

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

[54] THE MATHWORKS. *Simulink*, http://www.mathworks.com/products/simulink/.

[55] A. TOOM, T. NAKS, M. PANTEL, M. GANDRIAU, I. WATI. *Gene-Auto: An Automatic Code Generator for a Safe Subset of SimuLink/StateFlow and Scicos*, in "European Conference on Embedded Real-Time Software (ERTS'08)", 2008.