INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team espresso

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

Rennes - Bretagne-Atlantique

Theme : Embedded and Real Time Systems

Activity Report

2010

Table of contents

1. Team

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, since october 1st.]
Yue Ma [INRIA]

Post-Doctoral Fellows

Yann Glouche [Post-Doctorate, INRIA]
Kenneth Johnson [Post-Doctorate, INRIA, until september 30th.]
Julien Ouy [Expert Engineer, INRIA, until september 30th.]
Julio Peralta [Expert Engineer, INRIA, until june 30th.]
Huafeng Yu [Expert Engineer, INRIA]

Administrative Assistant

Stéphanie Lemaile [Secretary, INRIA]

Others

François Fabre [Junior Engineer, INRIA]
Vincent Mahé [Expert Engineer, INRIA, until march 30th.]

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 december 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 2010 is the delivery of the Polychrony toolset in open-source under GPL and EPL licenses. It is the result of a process initiated early this year and conducted in close collaboration with INRIA's DTI in order to precisely identify the perimeter of the license and identify the best-suited licensing terms compatible with its users and potential contributors.

Our second headline is the publication of a book on the "Synthesis of embedded software" with Springer, co-edited by Sandeep Shukla and Jean-Pierre Talpin. The publication of this book is consecutive to organization of a one-day tutorial at the Design Automation and Test in Europe (DATE) 2009 Conference on "Correct-by-Construction Embedded Software Synthesis: Formal Frameworks, Methodologies, and Tools", in 2009.

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 [42] 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.

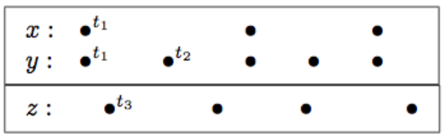


Figure 1. Behavior b over three signals x, y and z in two clock domains

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 $t' \neg < t$.

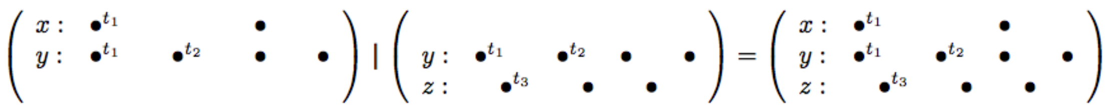


Figure 2. Synchronous composition of $b \in p$ and $c \in q$

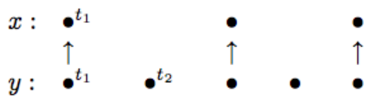


Figure 3. Scheduling relations between simultaneous events

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 $d \leq c$.

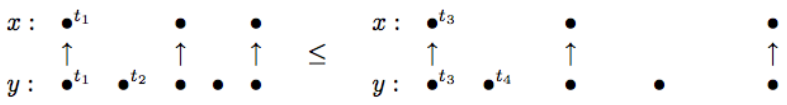


Figure 4. Relating synchronous behaviors by stretching.

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}.

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

- 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).

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

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 ^= 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 $\texttt{when}\,x$ (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 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).

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

3.1.3.2. 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.

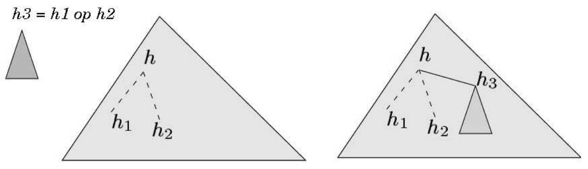


Figure 6. Hierarchization of clocks

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 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). Signal GUI requires the Signal toolbox or another component that redefines the Signal toolbox Syn and Sem APIs. 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 requires the Signal toolbox or another component that redefines the Signal toolbox Syn and Sem APIs. 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 programs examples,
- user oriented and implementation documentations,
- facilities to generate new versions.

The Polychrony toolset is downloadable on the following web sites:

- The Polychrony toolset public web site: `http://www.irisa.fr/espresso/Polychrony`. This site, intended for users, contains downloadable executable versions of the software for differents platforms, user documentations, examples, etc.
- The INRIAGForge: `https://gforge.inria.fr`. This site, intended for 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.

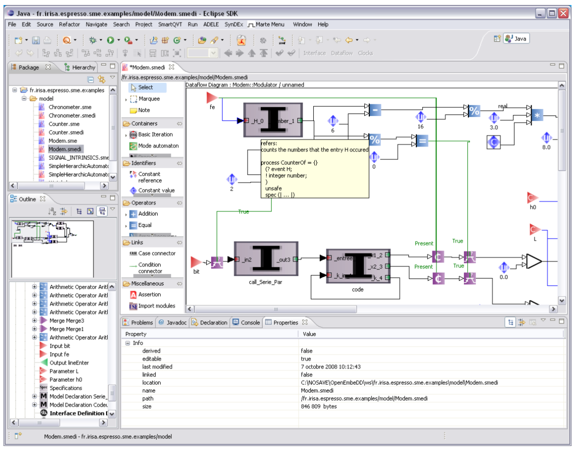
The Polychrony toolset currently runs on Linux, MacOS and Windows systems.

The Geensoft company, now part of Dassault Systèmes, supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects: by Airbus for the A380 embedded software developments, by Hispano-Suiza (SAFRAN Group) for aircraft engines applications (see `www.geensoft.com`).

4.2. The SME platform

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

We have developed a metamodel and interactive editor of Polychrony in Eclipse. Signal-Meta is the metamodel of the Signal language. It describes all syntactic elements specified in [43]: all Signal operators (e.g. arithmetic, clock synchronization), model (e.g. process frame, module), and construction (e.g. iteration, type declaration). Signal-Meta has been extended to allow the definition of mode automata to extend the functionality-oriented data-flow paradigm with the capability to model transition systems easily and provide an additional imperative flavor (Sec. 4.4).



Property	Value
Info	
derived	false
editable	true
last modified	7 octobre 2008 10:12:43
linked	false
location	C:\NOSAVE\OpenEmbeDD\ws\fr.irisa.espresso.sme.examples\model\Modem.smedi
name	Modem.smedi
path	/fr.irisa.espresso.sme.examples/model/Modem.smedi
size	846 809 bytes

Figure 7. Eclipse SME Environment.

These metamodels aim at providing a user with a graphical framework allowing to model applications using a component-based approach. Application architectures can be easily described by just selecting these components via drag and drop, creating some connections between them and specifying their parameters as component attributes. Using the modeling facilities provided with the Topcased framework, we have created a graphical environment for Polychrony (see figure 7) called SME (Signal-Meta under Eclipse). To 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 [34], in the current OpenEmbeDD distribution [33], or in the TopCased distribution [37].

4.3. MIMAD - Integrated Modular Avionics design using Polychrony

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

The Apex interface, defined in the ARINC standard [38], 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 [39]). 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 [44] have been proposed until recently, the latest advance being the combination of stream functions with automata in [45]. 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 metamodel of multi-clocked mode automata. A salient feature is the simplicity incurred by the separation of concerns between data-flow (that expresses structure) and control-flow (that expresses a timing model) that is characteristic to the design methodology of Signal.

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

5. New Results

5.1. Polychrony and Kahn Process Networks

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

We have continued our reconsideration work on fundamental bases of the polychronous model of computation and its relation to the well-known model of computation of Kahn Process Networks. This is leading us to a new model, currently referred to as WAGaLS, new acronym for ‘Widely Asynchronous, Globally limited asynchrony, Locally Synchronous”. It is based on an algebra of Kahn networks (characterized by unbounded fifos) of polychronous systems (with bounded fifos), the local components of which are atomic endochronous nodes.

5.2. Extensions of Signal

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

Starting from a thorough study of the functional aspects of the AADL (Architecture Analysis and Design Language) and their modeling in the polychronous model of computation, we are working on extensions of the Signal language with new general concepts. This work has different objectives such as the use of AADL concepts as description language for a functional version of Signal (AADL specifying the globally asynchronous architecture), the use of Signal as description language for thread behaviors in AADL, etc. There are different ways for these extensions such as, at a first level, the development of a Signal library for AADL that contains parameterized Signal processes to represent ports, queues, synchronization schemes and so on, but also the addition in the definition of Signal of AADL useful general concepts and features. The concept of functor could be introduced in Signal for this purpose. One meaningful extension we are studying consists in considering different types of systems and composition operators with respect to time models. For instance, a time domain, providing real-time events, may be associated with a synchronous system; an asynchronous system is built as an asynchronous composition of synchronous systems, with an assertion over time domains.

5.3. Toward a new clock calculus

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

The writing of an article that provides a detailed presentation of the program analysis and code generation techniques currently present in Polychrony [19] led us to consider an extension of the hierarchical clock calculus. The current implementation of the clock calculus is based on a structure of forest of clocks. However, this structure is not really convenient when two equivalent clock formulas may be placed in two distinct trees. Then choices have to be made when some other formula built on one of these two formulas must be placed. A new structure, called *banyan system*, is defined to represent such configurations in some non arbitrary way.

5.4. Documentation of Polychrony

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

A new technical documentation of the Polychrony toolset has been designed to accompany its open-source distribution. This multiscale, multipurpose documentation aims to provide different views of the software, from a high-level structural view to low-level descriptions of basic modules. One of the objectives 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. This provides also a model for contributors that would wish to add new functionalities or to replace some components of the software by newly developed ones.

A high-level architectural view of the Polychrony toolset is given in Figure 8.

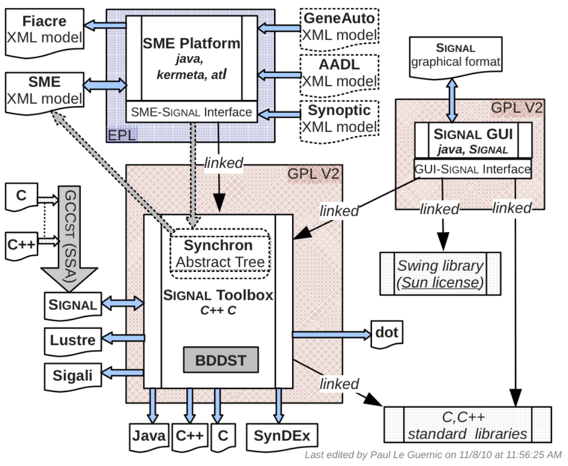


Figure 8. The Polychrony toolset high-level architecture

5.5. New features of Polychrony

Participants: Loïc Besnard, François Fabre, Thierry Gautier.

To answer a strong requirement of industrials in the domain of embedded systems, we have introduced traceability features into the Polychrony toolset. Indeed, in this field, any transformation of program must be justified and it must be possible to trace it. On the other hand, such traceability information can also be exploited in purposes of debugging applications. 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 have realized 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).

Two new features have been also developed for the SME Platform: interactive compilation and general compilation. The interactive compilation feature calls directly the needed functions when the user clicks on the corresponding button. In the general compilation feature, the user first clicks on the available options. Then, the Signal Toolbox batch compiler is called.

To facilitate building the Sigali tool environment, jointly developed with the Vertecs project-team (http://www.irisa.fr/vertecs), on various platforms, we have integrated the use of the *cmake* http://www.cmake.org tool. Cmake is a cross-platform build generator. Projects specify their build process with platform-independent files included in each directory of a source tree. Users build a project by generating a build system for a native tool on their platform.

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

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

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

The satellite management software is usually divided in parts that are quite autonomous one from the other, even if they share the same platform and resources. Inside each of these parts, the subparts are most of the time strongly linked and must cooperate in a synchronous manner. These parts usually exchange information but with less time critical constraints, thus relying on asynchronous exchanges. Such a system is usually called GALS. The flight software consists of a set of subsystems, called synchronous islets. These islets can communicate with each other, with other equipment or with the ground station via the middleware services. The middleware is built on a real-time kernel that provides services such as task management and synchronization primitives. We have defined this year a code generation toolchain in collaboration with Thales Alenia Space (TAS). This chain has been applied on a case study specified by TAS. The goal of this study was to show the feasibility of automatic code generation from the Synoptic model using the Polychrony toolset and its implementation using the TAS framework (Ostrales). Ostrales supplies essentially a service for task management including mechanisms of synchronization. An overview of this toolchain is given in Figure 9.

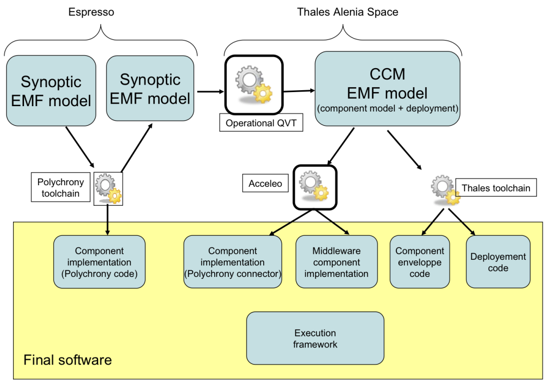


Figure 9. The Espresso/Thales Alenia Space tool chain

The Espresso toolchain is composed of three parts (described in the Spacify report Spa5bis):

- The transformation of the annotated Synoptic program in a Signal program. It is realized by a model to model transformation using the Kermeta tool. The details of this transformation are given in the Spacify report Spa4b.
- The automated code generation with threads, dynamically scheduled, using the Signal compiler.
- The production of a new Synoptic program induced by the modification of the interface of the islets (control signals are added).

The Thales Alenia Space toolchain is described in the Spacify report Spa11c.

5.7. Labelled transition systems for validation of Signal specifications

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

Labelled Transition Systems (LTSs) are a well-known model for describing and reasoning about concurrent communication systems (à la CSP). We have proposed a compositional approach for translating Signal into LTSs by systematically translating each source Signal kernel operator into an LTS and then composing the resulting LTSs using the standard parallel composition operator. In general, this approach requires a global analysis so that the locking scheme does not render the resulting LTS deadlocking when an operational interpretation of the Signal program does not deadlock. For a subclass of Signal programs, a sufficient global analysis is that provided by the clock calculus. Our translation may be exploitable for automatic verification (e.g. model checking), for distribution verification (e.g. for globally asynchronous, locally synchronous architectures), for translation validation, etc. We have proposed in particular a translation validation approach in which we translate both the Signal specification and the associated C simulator into LTSs. Then, an appropriate preorder test between both LTSs can be interpreted as a refinement between the source Signal specification and its C implementation [24].

5.8. An algebraic theory of data-flow processing

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

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

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

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

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

5.9. Virtual prototyping of imperative programs

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

Many software systems involve multi-threaded programs featuring thread communication and shared resources. The correct and efficient use of shared resources between threads relies on synchronization methods, such as specialized commands or events communicated between threads. We have proposed a new approach that demonstrates a semi-automated method of translating cooperatively threaded software to the language Signal in order to verify the correctness of thread synchronizations in the source code. We consider FairThreads threads attached to a scheduler that uses deterministic and cooperative scheduling policy. Our method is based on automatic translation of C programs into Signal via an intermediate Static Single Assignment (SSA) representation, translation which is extended to take into account cooperative threads [30], [22].

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

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

Clock Constraint Specification Language (CCSL) [41] is defined in an annex of the UML MARTE profile [50]. We are interested in the analysis, verification, and synthesis of timed systems specified in CCSL. These systems subject to clock constraints can be modeled, specified, analyzed, and simulated within two software environments: TimeSquare [48] and Polychrony. Clock constraints are solved using a heuristic algorithm in TimeSquare, which is generally non-deterministic. Simulation is carried out and demonstrated in the form of waves. In comparison, Polychrony enables deterministic specifications and formal analysis for the design of safety-critical systems. It is a promising approach to integrate the complementary technologies present in the two software environments for the purpose of system-level design.

In order to benefit from the advantages provided by Polychrony, the clock constraints specified in CCSL are translated into Signal, therefore, they are analyzed by tools and technologies associated with Polychrony. This translation is carried out in the following steps. The first step is involved in the CCSL clock hierarchy analysis with the help of the Polychrony hierarchization technique [28]. This analysis helps to identify the determinism with regard to system temporal behavior, and thus aids code generation for simulation. In comparison, TimeSquare only provides non-deterministic simulations. The second step is the translation of CCSL into Signal. Most primitive CCSL clock relations can be translated into Signal either directly or via the integration of additional clocks in the translation. Currently, only certain CCSL clock relations involved in unbounded FIFO are not addressed. This translation enables to bridge between TimeSquare and Polychrony. Controller synthesis is carried out in the last step. Properties are taken from clock constraints, such as precedence or synchronization, and are ensured by the synthesized controllers. We use the Sigali tool for the controller synthesis.

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

5.11. Transformation of Simulink models in Polychrony

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

Dataflow models and state machines are common models of computation adopted in the system design of avionics, automotive applications, etc. One of the most popular tools that accept these models is Simulink/Stateflow in the Matlab family. One of the models of computation adopted by Simulink is extended dataflow, and Stateflow relies on the model of state machines. A typical Simulink model is defined by a set of interconnected blocks, which model entities in a system, such as sensors, actuators, and logical operations. The library of Simulink includes function blocks that can be linked and edited in order to model the dynamics of the system. Gene-Auto is a framework for code generation from a safe subset of Simulink and Stateflow models for safety critical embedded systems. This safe subset is also adopted in our work. From now on, Simulink is used for short to indicate the subset of Simulink and/or Stateflow languages that is adopted by Gene-Auto. The scope of the Gene-Auto project is initiated for the generation of C-code from safe subsets of Simulink, and Stateflow modeling languages. Considering a Simulink model, we used Gene-Auto transformation for having (by transformations) a model conform to the Gene-Auto Ecore meta-model.

The Gene-Auto model considers only the discrete time approach of Simulink. A Gene-Auto model is defined by a set of blocks representing each entity (inputs, outputs, logical operators,...) of a system. Each block is associated to a specific activation clock. At each tick of this clock, a block carries out a computation on its available inputs and produces new outputs. A global activation clock is used to synchronize the activation clocks of Simulink blocks. From this point of view, our Simulink model is synchronous, and each of its blocks is thus modeled as a synchronous Signal process.

The model transformation chain from functional models in Simulink to Signal is divided into several steps. The first step involves the transformation of Simulink models to Ecore based Gene-Auto models. These models are then translated into Signal via the SME metamodel through a transformation implemented in Kermeta. The whole transformation chain from high level Simulink/Gene-Auto models to executable code is illustrated by Figure 10.

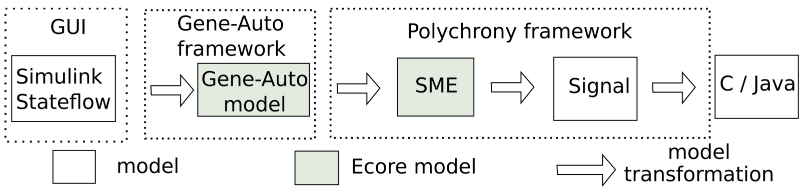


Figure 10. A global view of Gene-Auto SME transformation chain.

5.12. Co-modeling and simulation of GALS architectures

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

In the framework of TopCased project, we are using Polychrony as a formal model for the co-modeling and validation of GALS architecture. It aims at interpreting the GALS architecture specified in AADL to a polychronous model of computation, and then automatically generate its GALS implementation, in order to obtain an executable model, allowing early simulation, testing and verification. Such simulation of GALS systems within the polychronous model has obvious advantages: it allows an easy and precise description of the whole range of applications, from the pure synchronous or asynchronous model to GALS architecture, and an early-phase validation and step-wise refinement of the system.

The main motivation of the work is to bridge the gaps between the semantics of AADL and Polychrony: a proposition of a methodology for system-level modeling and validation of embedded systems specified in AADL via the polychronous model of computation. We implemented a plug-in for TopCased framework to perform model transformation from AADL to SME. This transformation is done using the model transformation language ATL. The main developments this year have focused on:

- Temporal interpretation of AADL model. Due to the different timing semantics between AADL and Signal, we keep the ideal view of instantaneous computations and communications of polychronous model, moving computing latencies and communication delays to specific memory process, that introduce delays and well suited synchronizations. The temporal properties of thread execution and communications are modeled in Signal.
- Software updates. The model transformation updates from ATL version 2.0.2 based on Eclipse 3.4.1, to the latest ATL version 3.1.1 based on TopCased 3.4.1 (Eclipse 3.5.2). The new version is integrated as a plug-in in the TopCased platform.

5.13. The Cesar demonstrator and reference technology platform

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

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 [46], MoBIES [40], SML-Sys [49], etc., we propose an approach to the issue of composing, integrating and simulating heterogeneous models in a system co-design flow ([27], Figure 11). First, the functional behavior of an application is modeled with synchronous data-flow and Statechart diagrams using Simulink/Gene-Auto [35] [54]. 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 [31], Syndex adequation [36], etc., is carried out based on the common model within Polychrony.

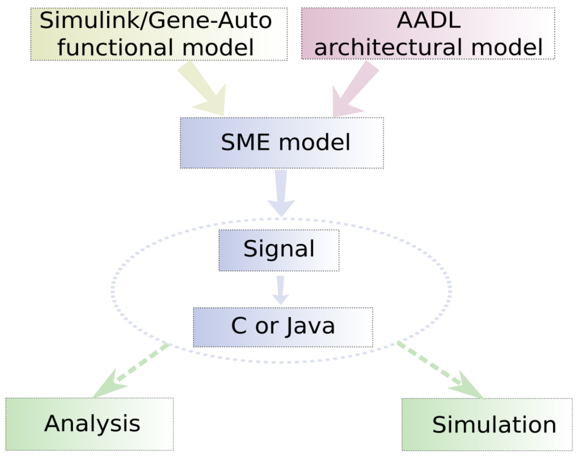


Figure 11. The design process for heterogeneous modeling and simulation within Polychrony

Polychrony has been integrated to the Reference Technology Platform (RTP) of Cesar to serve as a framework for co-modeling and architecture exploration. To demonstrate these capabilities, we participated to 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. We participated to the project's demonstration of the RTP at the annual Artemisia Consortium meeting.

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. 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 (Figure 12). 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.

ModelBus [32] is used for the integration of Polychrony into the Cesar Reference Technology Platform (RTP). ModelBus, an integration platform based on Service-Oriented Architecture (SOA), connects different services offered by tools connected to ModelBus. In addition, ModelBus repositories are used to store models, which are made available for all tools attached to ModelBus. Our whole model transformation and simulation chain has been implemented with Galileo Eclipse and attached to ModelBus as a provider of registered remote

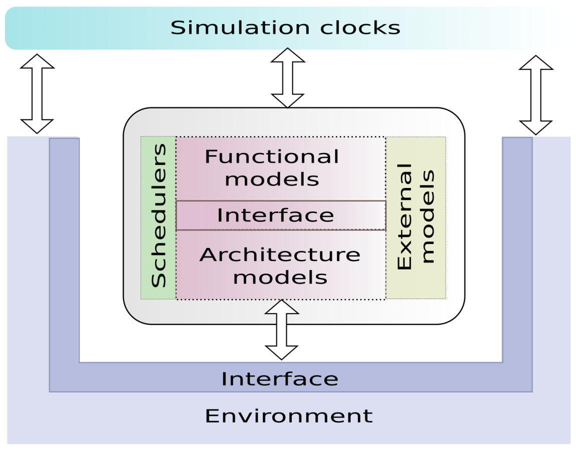


Figure 12. The system integration from the point of view of Signal processes

service. The data consumed by this service is Simulink and AADL models, and the produced data is a VCD file. All these data are exchanged via ModelBus repositories. This demonstration also shows the integration of Polychrony with other tools, such as OSATE (AADL), Simulink, Gene-Auto, TimeSquare, ATL, Kermeta, etc.

5.14. From polychronous to Safety-critical Java

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

Existing real-time and mission-critical programming languages suffer from a low level of abstraction and non-determinism in both the timing and functional domains. Virtualisation is an attractive solution to get around re-certification of systems when changes occur in the hardware environment, which are very probable since such systems are designed to operate for a long period.

One of the most famous high-level language virtual machine architecture is the JVM (Java Virtual Machine). Java is used in many disciplines such the conception and prototyping of embedded systems. Unfortunately, commercial JVMs are designed to maximize the performance of applications at the expense of predictability. Therfore, they cannot be used to implement safety-critical systems. However, if we place some restrictions and modifications, one can surpass the problem. This is the objective of RTSJ (real-time specification for Java), SCJ (safety-critical Java) and many programming models such as Flexotask. Flexotask is a graphic environment under Eclipse, which allows generating automatically safety-critical programs (for an appropriate JVM) from a data-flow specification. This framework is developed at the Purdue University, Indiana, USA.

Started recently with the co-direction of Prof. Jan Vitek (one of the designers of Flexotask), the thesis “A polychronous model of computation for SCJ” aims to provide SCJ or Flexotask with the mathematical tools of polychrony and so the capability of verifying key properties such as determinism.

6. Contracts and Grants with Industry

6.1. RNTL project Spacify

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

In the context of the ANR project Spacify, whose aim is to design a domain-specific programming environment for software onboard spacecraft called Synoptic, the Espresso team provides Polychrony as a code generation infrastructure for Synoptic and has achieved two major results assessing the impact of this technology transfer.

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

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

First, a long-lasting point-to-point case study conducted with one of the industrial end-user of the project, Thalès Alenia Space, has been concluded by validating the usability of modular and concurrent code generation services of Polychrony to implement, together with TAS in-house real-time operating system, the main control-software functionalities (AOCS) of a generic satellite model.

Second, and through a close collaboration with another industrial partner of the project, Geensoft, who distributes a commercial environment based on Signal and its polychronous model of computation: RT-Builder; we successfully demonstrated the traceability functionalities present in the Polychrony environment by allowing the joint utilization of the concurrent code generated by Polychrony from Synoptic diagrams with the RT-Builder as a simulation toolset.

6.2. ANR project FotoVP

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

The Espresso project-team participates to the ANR project FotoVP, leaded by Verimag. The aim of the FotoVP is to develop virtual prototyping tools and techniques for the simulation and verification of system-level C/SystemC specifications in synchronous models of computation. Our results related to this project are presented in section 5.9

6.3. Artemisia project Cesar

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

Our collaborations with the partners of projects Topcased and Spacify is continued in the Artemisia project Cesar who, from the results obtained so far, have contributed to integrating the Polychrony toolset in the Reference Technology Platform (RTP, v1.0) of the project. The Polychrony toolset has also been selected to serve as academic demonstrator of Cesar sub-project SP3. A first demonstration has been presented at the plenary meeting of the Artemisia consortium early June.

Cesar means Cost-Efficient methods and processes for SAfety Relevant embedded systems. The project aims at a stronger integration of safety engineering methods and techniques, all along the phases of the development process. The purpose is to optimize globally the safety critical embedded system architecture by taking in account simultaneously all viewpoints and associated criteria (cost, mass, safety).

Part of the demonstration performed in the frame of project Cesar displays results on a case study performed in the context of project ITEA2 project Opees, in collaboration with Airbus. This case study 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.

6.4. ITEA2 project Opees

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

The ITEA2 project Opees is defined as a continuation of the ANR project Openembedd to provide an open-source platform for embedded software design. The mission of Opees is to build a community able to ensure long-term availability of innovative engineering technologies in the domain of dependable / critical software-intensive embedded systems. Its main objectives are to secure the industrial strategy, improve their competitiveness and develop the European software industry. The contribution of team Espresso in Opees consists of the following:

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

7. Other Grants and Activities

7.1. National Actions

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

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

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

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 [47].

7.3. International collaborations

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

7.3.1. Virginia Tech

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 volumes as well as 14 journal and conference articles. In the frame of this collaboration, we jointly created the ACM-IEEE MEMOCODE (http://memocode.irisa.fr/2003/) international symposium series as well as the FMGALS international workshop series. Finally, we jointly organized four tutorials.

The collaboration resulted in a technology transfer of the Polychrony toolset with the launch of the project CodeSyn at Virginia Tech, funded (and renewed) by the USAF Laboratories. CodeSyn is an actor-based system level design formalism whose aim is to implement the polychronous model of computation to design, verify and synthesize concurrent real-time embedded avionics on multi-core architectures. We aim at renewing our past associated project with Fermat in order to support and help this transfer of our research and development results. In the meantime, a former PhD of team Espresso, Julien Ouy, has joined the Fermat laboratory to developp a code generation infrastructure for CodeSyn using the Polychrony toolset.

7.3.2. Purdue University

Since 2009, we started a collaboration with Pr. Jan Vitek, Purdue University. Jan Vitek is a programming language expert participating in the standardization of real-time and safety critical extensions and API for the programming language Java (RTJ and SCJ). The aim of our collaboration is to promote the polychronous model of computation as an abstract, concurrent, semantics framework in which we could capture a high-level (system-level) abstraction of communication and concurrency in SCJ that would be suitable to ease its use by the programmer, to ease formal analysis and verification (to guarantee asynchronous determinism of concurrent Java programs) and lead to automated compilation techniques. In the frame of this collaboration, Jean-Pierre Talpin was invited speaker at JTRES'09 and invited lecturer at the TIC'10 summer school. To support this collaboration, Jan Vitek is invited professor at INRIA-Rennes/IRISA in summer 2010 and the NSF-PIRE proposal NEXUS as been jointly submitted (together with several US and European partners). Our collaboration is strengthened with the arrival of a jointly supervised PhD student, Adnan Bouakaz.

7.3.3. Beihang University

Since 2009, we started an informal collaboration which consisted in two consecutive invitations at Beihang University (Beijing University for Aviation and Aerospace, BUAA). Pr. Ma and Pr. Hu submitted the nomination Jean-Pierre Talpin as invited professor at Beihang. We also submitted a joint collaboration to CNRS jointly with BUAA, project-team ESPRESSO and IRIT (one of our closest partners in the Topcased, Spacify, Cesar and Opees projects). This collaboration aims at exploring the best-suited formal modeling framework, from models of computation to tool-set, to assist the engineering of China's next generation airplanes onboard avionics with formal verification, simulation and test tools, as well as compilers to automatically generate DO178-qualifiable code from system-level (architectural and behavioral) specifications.

8. Dissemination

8.1. Workshops

Loïc Besnard and Jean-Pierre Talpin have participated to the meeting INRIA Industries in Toulouse (May 2010) – `http://www.inria.fr/innovation/aeronautique-defense-spatial-securite/showroom-des-demos/l-exploration-d-architecture`.

8.2. Invited Lectures

Jean-Pierre Talpin was invited speaker at the 3rd International School on Trends in Concurrency (TIC'10, http://web.me.com/vitekj/TIC10) on "Models, methods and tools for virtual prototyping embedded architectures"; and at the inaugural seminar of the ONERA/THALES working group TORRENTS on Time ORiented Reliable Embedded NeTworked Systems (http://www.irit.fr/torrents).

8.3. Research visitors

Jan Vitek (Purdue University) visited team in summer 2010 with the support of the University of Rennes 1.

8.4. Conferences

Jean-Pierre Talpin is a member of the steering committee of the ACM-IEEE conference on methods and models for codesign (MEMOCODE) and of the editorial board of the EURASIP Journal on Embedded Systems. This year, he served as technical program committee member for the following conferences

JTRES'10 - http://d3s.mff.cuni.cz/conferences/jtres2010

ACSD'10 - http://www.informatik.uni-augsburg.de/acsd

SIES'10 - http://events.unitn.it/en/sies2010

SAC'10 - http://oldwww.acm.org/conferences/sac/sac2010

Thierry Gautier served as technical program commitee member for the conference MEMOCODE 2010 – `http://www-memocode2010.imag.fr/`.

8.5. Teaching

- Thierry Gautier and Loïc Besnard taught on real-time programming at the DIIC2 Graduate program of the University of Rennes 1 (ESIR school).
- 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] 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, p. 641-657, http://hal.inria.fr/inria-00522794.

[15] I. R. QUADRI, A. GAMATIÉ, S. MEFTALI, J.-L. DEKEYSER, H. YU, É. RUTTEN. *Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation*, in "International Journal of Embedded Systems", Sep 2010, 18 p, http://hal.inria.fr/inria-00525015.

[16] IMRAN RAFIQ. QUADRI, H. YU, A. GAMATIÉ, S. MEFTALI, J.-L. DEKEYSER, É. RUTTEN. *Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation*, in "International Journal of Embedded Systems", 2010, http://hal.inria.fr/inria-00525015.

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

[18] H. YU, A. GAMATIÉ, É. RUTTEN, J.-L. DEKEYSER. *Adaptivity in High-Performance Embedded Systems: a Reactive Control Model for Reliable and Flexible Design*, in "Knowledge Engineering Review", 2010, Accepted for publication, http://hal.inria.fr/inria-00536883.

International Peer-Reviewed Conference/Proceedings

[19] L. BESNARD, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Compilation of Polychronous Data Flow Equations*, in "Synthesis of Embedded Software", SANDEEP K. SHUKLA, J.-P. TALPIN (editors), Springer, 2010, p. 1–40.

[20] A. CORTIER, L. BESNARD, J.-P. BODEVEIX, J. BUISSON, F. DAGNAT, M. FILALI, G. GARCIA, J. OUY, M. PANTEL, A. RUGINA, M. STRECKER, J.-P. TALPIN. *Synoptic : a domain-specific modeling language for space on-board application software*, in "Synthesis of Embedded Software", SANDEEP K. SHUKLA, J.-P. TALPIN (editors), Springer, 2010, p. 147–171.

[21] Y. GLOUCHE, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *A Module Language for Typing Signal Programs by Contracts*, in "Synthesis of Embedded Software", SANDEEP K. SHUKLA, J.-P. TALPIN (editors), Springer, 2010, p. 147–171.

[22] K. JOHNSON, L. BESNARD, T. GAUTIER, J.-P. TALPIN. *A Synchronous Approach to Threaded Program Verification*, in "Proceedings of the 2010 10th International Workshop on Automated Verification of Critical Systems", AVOCS '10, 2010, p. 168–183.

[23] Y. MA, J.-P. TALPIN, T. GAUTIER. *Interpretation of AADL Behavior Annex into Synchronous Formalism using SSA*, in "Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology", IEEE Computer Society, 2010, p. 2361–2366.

[24] JULIO C. PERALTA, T. GAUTIER, L. BESNARD, P. LE GUERNIC. *LTSs for translation validation of (multi-clocked) Signal specifications*, in "Proceedings of the 2010 8th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)", IEEE Computer Society, 2010, p. 199–208.

[25] J.-P. TALPIN, J. OUY, T. GAUTIER, L. BESNARD, A. CORTIER. *Modular Interpretation of Heterogeneous Modeling Diagrams into Synchronous Equations Using Static Single Assignment*, in "Proceedings of the 2010 10th International Conference on Application of Concurrency to System Design", ACSD '10, IEEE Computer Society, 2010, p. 137–146.

[26] E. VECCHIE, J.-P. TALPIN, S. BOISGÉRAULT. *A higher-order extension for imperative synchronous languages*, in "Software and Compilers for Embedded Systems", SCOPES'10, ACM Press, 2010, p. 168–183.

[27] 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 "Proceedings of the 26th ACM Symposium On Applied Computing (SAC'11)", 2011, http://hal.inria.fr/inria-00536907.

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

Books or Proceedings Editing

[29] SANDEEP K. SHUKLA, J.-P. TALPIN (editors). *Synthesis of Embedded Software*, Springer, 2010.

Research Reports

[30] K. JOHNSON, L. BESNARD, T. GAUTIER, J.-P. TALPIN. *A Synchronous Approach to Threaded Program Verification*, INRIA, Jun 2010, RR-7320, http://hal.inria.fr/inria-00492694.

References in notes

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

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

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

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

[35] *The MathWorks: Simulink*, http://www.mathworks.com/products/simulink/.

[36] *Syndex*, http://www-rocq.inria.fr/syndex/.

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

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

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

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

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] J. EKER, J. W. JANNECK, EDWARD A. LEE, J. LIU, J. LUDWIG, S. NEUENDORFFER, S. SACHS, Y. XIONG. *Taming heterogeneity: The Ptolemy approach*, in "Proc. IEEE", 2003, vol. 91, n^{o} 1, p. 127-144.

[47] 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.

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

[49] D. A. MATHAIKUTTY, H. D. PATEL, S. K. 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.

[50] 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.omgmarte.org/node/4.

[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] 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.