INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Espresso

Environnement de Spécification de Programmes RéactifS SynchrOnes

Rennes

THEME 1C

Table of contents

1. Team

Head of project-team
Jean-Pierre Talpin [Research scientist (CR Inria)]

Administrative assistant
Maryse Auffray [AA Inria]

Staff members (Inria)
Thierry Gautier [Research scientist (CR)]
Paul Le Guernic [Research director (DR)]

Staff member (CNRS)
Loïc Besnard [Technical staff (IR)]

Faculty members
Bernard Houssais [Associate Professor (Université de Rennes 1)]
Mickaël Kerbœuf [Lecturer (INSA Rennes)]

Technical staff
Bruno Le Dez [Project technical staff, until June 1st]
Luc-Michel Sévère [Junior technical staff, until October 1st]

Post-doctoral fellow
Qiuling Pan [Inria, until March 31th]

Ph. D. students
David Berner [Inria]
Abdoulaye Gamatié [Inria]
Laurent Vibert [*normalien,* until September 1st]

2. Overall Objectives

2.1. Introduction

An everyday broader range of software engineering areas requires reasoning on a combination of sequential or synchronous and asynchronous interaction at the different architectural levels of the system under design. Relevant practical examples are co-designed hardware-software architectures, multi-tasked/threaded embedded systems, distributed telecom applications. In summary, every system whose design requires robustness to latency, to distribution, to threading, to separate compilation.

The objectives of ESPRESSO project-team are to propose formal models and implement formal methods and reliable tools for engineering trusted application components and architectures for embedded and mission-critical systems, on a spectrum of architectures ranging from circuits to distributed systems, yet within a simple mathematical framework offering the best reliability guarantees. The project-team demonstrates the effectiveness and efficiency of this approach by means of the POLYCHRONY design environment. POLYCHRONY is an integrated development environment and technology demonstrator consisting of a compiler, a visual editor and simulator and a model checker.

2.2. Strategy and vision

The ESPRESSO project-team builds upon the achievements of the former EP-ATR project-team to propose formal models, methods and tools to aid the development of reliable software components in embedded system design. The main paradigm shift of the ESPRESSO project-team is the move from programming languages to formal methods, where models, methods and tools that made the success of the former are further put into focus in the latter in the context of heterogeneous computing and programming paradigms. The models considered in the project-team homogeneously span from synchrony to asynchrony, from discrete time to real time, and are

captured by the polychronous model of computation [16]. The formal methods considered in the project-team put this model to work for the refinement-based (top-down) and component-based (bottom-up) development of software, yielding support for a true platform-based design approach suitable for modern avionics, automotive and system-level architecture development.

The project-team makes continuous efforts in the development and release of the POLYCHRONY workbench (http://www.irisa.fr/espresso/Polychrony). The POLYCHRONY workbench consists of a toolbox offering services for the implementation of the formal design methodologies studied in the project-team. Its commercial implementations (SILDEX and RTBUILDER from TNI-VALIOSYS) have achieved industrial-scale usage at Airbus Industries and Snecma in the context of the past IST SAFEAIR project. Past projects (projects SAFEAIR, EXPRESSO, ACOTRIS, NSF-INRIA collaboration) and forthcomming ones (IST proposals ESPACE, PANDORA, SEA, ARTIST network) demonstrate extensive collaborations with academic and industrial partners in the applications areas of avionics, automotive and architecture-aware embedded system design. The project-team puts a sustained effort in promoting the experimental POLYCHRONY workbench as free software with the long-term goal of releasing an open-source distribution.

2.3. Context and motivations

The high-level design of embedded systems and architectures have gained prominence in the face of rising technological complexity, increasing performance requirements and shortening time to market demands for electronic equipments. Today, the installed base of intellectual property (IP) further stresses the requirements for adapting existing components with new services within complex integrated architectures, calling for appropriate mathematical models and methodological approaches to that purpose.

Over the past decade, numerous programming models, languages, tools and frameworks have been proposed to design, simulate and validate heterogeneous systems within abstract and rigorously defined mathematical models. Formal design frameworks provide well-defined mathematical models that yield a rigorous methodological support for the trusted design, automatic validation, and systematic test-case generation of systems.

However, they are usually not amenable to direct engineering use nor seem to satisfy the present industrial demand. As a matter of fact, the attention of the industry tends to shift to modeling frameworks based on general-purpose programming language variants, in response to a growing industry demand for higher abstraction-levels in the system design process and an attempt to fill the so-called *productivity gap*.

At present, a possibility of widening existing divergences between formal methods and industrial practices is perceivable. It seems that any useful methodology cannot avoid the industrial trend of using emerging programming languages. This contrasted picture calls for an effort toward the convergence between the theory of formal methods and the industrial practice and trends in system design.

Project-team ESPRESSO aims at this convergence by considering the formal modeling framework POLYCHRONY (as one of many formal design frameworks proposed over the past decade). Project-team ESPRESSO aims at the implementation of present industrial practices and trends in such a framework by considering the required definition of formal engineering models and appropriate methodological approaches by means of effective program analysis and transformation techniques implemented in the experimental platform POLYCHRONY.

2.4. 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 and integrated circuits [15]. 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 notation of the POLYCHRONY workbench, 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 desynchronization, from specification to implementation, from abstraction to concretization, from interfaces to implementations. SIGNAL gives the opportunity to seamlessly model circuits and devices 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 the synchronous-desynchronized design model of SIGNAL invites and favors the design of correct-by-construction systems by means of well-defined transformations of system specifications (morphisms) that preserve the intended semantics and stated properties of the architecture under design.

3. Scientific Foundations

In practice, a multi-clocked system description is often the representation or the abstraction of an asynchronous system or of a GALS architecture. In system design, the asynchronous implementation of a system is obtained through several refinements of its initial description. However, clocks are often left unspecified at the functional level, and no choice on a master clock is made at the architectural level. As communication and implementation layers are reached, however, multiple clocks might be a way of life. In the polychronous design paradigm, one can actually design a system with partially ordered clocks and refine it to obtain master-clocked components integrated within a multiply-clocked architecture, while preserving the functional properties of the original high-level design, thanks to the formal verification methodology provided by the formal theory (model and theorems) of polychronous signals. Our goal is to derive conditions on specifications under which design refinement principles work. We seek toward tools and methodologies to allow to take high-level specifications and to refine them in a semantic-preserving manner into GALS implementations.

We start with an outline of the formalism supported by the POLYCHRONY workbench: the data-flow synchronous programming language SIGNAL, and give an informal presentation of its polychronous model of computation.

3.1.1. The synchronous data-flow notation Signal

In SIGNAL, a process P consists of the composition of simultaneous equations over signals. A signal $x \in \mathcal{X}$ describes a possibly infinite flow of discretely-timed values $v \in \mathcal{V}$. An equation $x = fy$ denotes a relation between a sequence of operands y and a sequence of results x by a process $f \in F$. Synchronous composition $P \mid Q$ consists of considering a simultaneous solution of the equations P and Q at any time. SIGNAL requires

three primitive processes: **pre** , to reference the previous value of a signal in time; **when** , to sample a signal; and **default** , to deterministically merge two signals (and provides, e.g. negation **not** , equality **eq** , etc.).

$$P ::= \vec{x} = f\,\vec{y} \mid P \parallel Q \mid P / x$$
$$f \in F \supseteq \{\, \mathsf{pre}\, v \mid v \in \mathcal{V} \} \cup \{\, \mathsf{when}\,, \mathsf{default}\,, \ldots \}$$

The equation $x = \mathsf{pre}\, v\, y$ (whose concrete syntax is x:=y$1 init v) initially defines x by v and then by the previous value of y in time (tags t_1, t_2, t_3 denote instants).

$$\begin{aligned} y &: (t_1, v_1)\,(t_2, v_2)\,(t_3, v_3)\ldots \\ \mathsf{pre}\ v\, y &: (t_1, v)\ \ (t_2, v_1)\,(t_3, v_2)\ldots \end{aligned}$$

Figure 1.

3.1.2. Example

We exemplify the equational/relational design model of SIGNAL by considering the definition of a counting process: **Count**. It accepts an input event **reset** and delivers the integer output **val**. A local **counter**, initialized to 0, stores the previous value of **val** (equation **counter := pre 0 val**). When the event **reset** occurs, **val** is reset to 0 (i.e. `(0 when reset)`). Otherwise, **counter** is incremented (i.e. `(counter + 1)`). The activity of **Count** is governed by the clock of its output **val**, which differs from that of its input **reset**: **Count** is multi-clocked.

```
process Count     =  (? event reset ! integer val)
  (| counter      :=  pre 0 val
   | val          :=  (0 when reset) default (counter + 1)
   |) where            integer counter; end;
```

reset				♯		
counter	0	1	2	3	0	1
val	1	2	3	0	1	2

3.1.3. Model transformations

In SIGNAL, distributed protocol synthesis and sequential code generation are design stages performed by correctness-preserving transformations of the system model. Transformations rely on the analysis of synchronization relations (e.g. $c \hat{=} v$, i.e. signals c and v are synchronous) and scheduling relations (e.g. $r \rightarrow^{c} v$, i.e. v cannot happen before r when c is present). This relational information is used to construct a canonical control flow graph.

model	clocks	scheduling	code
$c := \mathsf{pre}\, 0\, v$	$c \hat{=} v$	$r \rightarrow^{r} v$	if r then v = 0 else v = c + 1;
$\mid v := (0\ \mathsf{when}\ r)\ \mathsf{default}\ c+1$	$v \hat{=} r \vee c$	$c \rightarrow^{c \setminus r} v$	c = v;

Figure 2.

Hierarchization [28] is the key transformation that allows to obtain this canonical control flow graph. Given a set of synchronization relations (e.g. $h_3 \hat{=} h_1 op \quad h_2$), it allows to place clocks (e.g. h_3) in the control-flow tree by determining their least upper bounds (e.g. h). Each clock (e.g. r) is the trigger of a set of actions that are performed in sequence according to scheduling relations (e.g. $r \rightarrow^{r} v$).

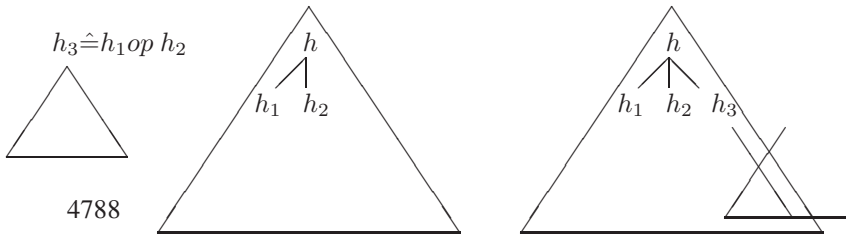

Figure 3. Hierarchization of a tree h_3 under a main tree of root h

3.1.4. A polychronous model of computation

The polychronous model of computation is proposed in [16] for the formal study of globally asynchronous and locally synchronous systems. We consider:

- A set of *values* $v \in \mathcal{V}$ to represent the operands and results of computations.
- A partially ordered set of tags $t \in \mathbb{T}$ to denote symbolic time, of minimum 0.

Events and signals defined starting from a partially ordered set of tags yield a polychronous structure of behaviors (finite maps from names to signals) and processes (sets of behaviors).

- An *event* $e \in \mathcal{E} = \mathcal{T} \times \mathcal{V}$ is the pair of a value and a tag.
- A signal $s \in \mathcal{S}$ is a function from a *chain* of tags to a set of values.
- A *behavior* $b \in \mathcal{B}$ is a function from names $x \in \mathcal{X}$ to signals $s \in \mathcal{S}$.
- A *process* $p \in \mathcal{P}$ is a set of behaviors of same domains.

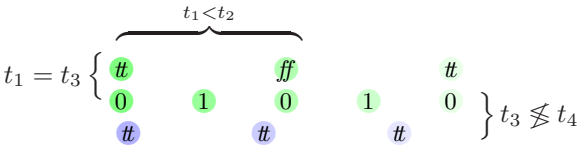

Figure 4. Partially-ordered events in the polychronous model of computation.

Notice that the chain of tags corresponding to the clock domain of a given signal may only be partially related to that of another signal, allowing to capture signals belonging to multiple clock domains within a behavior of a given process: partially ordered tags effectively describe a timing abstraction of multi-clocked systems. By contrast, considering a totally ordered timing domain requires the addition of a special mark τ or $\perp$ to render the absence of the value carried by a signal at a given time.

3.1.5. Scheduling

To additionally render the possible scheduling of events at a given time, we equip processes with a pre-order relation $t_x \rightarrow t'_y$ to denote scheduling: an event along the signal named y at t' may not happen before x at t. The pair t_x of a time tag t and of a signal name x renders the date d of an event along the signal x at the symbolic time or instant t. Hence, a tag t corresponds to the equivalence class of a synchronization relation between dates, in the spirit of synchronous structures [41]. The domain of dates $\mathcal{D} = \mathcal{T} \times \mathcal{X}$ is subject to a pre-order relation $\rightarrow$ that denotes scheduling.

The synchronous composition $p \,|\, q$ of two processes p and q is defined by the union of all behaviors b (from p) and c (from q) such that all signals along the interface I of p and q carry the same values at the same tags

3.1.6. Algebraic structure of polychrony

Scalability is a key concept for engineering embedded systems in a smooth design process. A formal support for timing scalability in design is given in our model by the so-called *stretch-closure property*. The intuition behind this relation is to consider a signal as an elastic with ordered marks on it (tags). If it is stretched, marks remain in the same relative and partial order but have more space (time) between each other. The same

$$P ::= \vec{x} = f\,\vec{y} \mid P \parallel Q \mid P \,/\, x$$
$$f \in F \supseteq \{\, \mathsf{pre}\, v \mid v \in \mathcal{V} \} \cup \{\, \mathsf{when}\,, \mathsf{default}\,, \ldots \}$$

The equation $x = \mathsf{pre}\, v\, y$ (whose concrete syntax is x:=y$1 init v) initially defines x by v and then by the previous value of y in time (tags t_1, t_2, t_3 denote instants).

$$\begin{array}{r} y : (t_1, v_1)\,(t_2, v_2)\,(t_3, v_3)\ldots \\ \mathsf{pre}\ v\, y : (t_1, v)\ \ (t_2, v_1)\ (t_3, v_2)\ldots \end{array}$$

Figure 5. Scheduling relations.

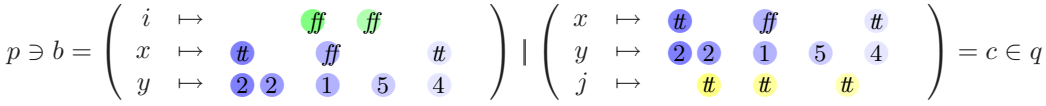

Figure 6. Synchronous composition $p \,|\, q$: matching behaviors along common signals (in blue)

holds for a set of elastics: a behavior. If elastics are equally stretched, the order between marks is unchanged. Stretching is a partial-order relation which gives rise to an equivalence relation between behaviors: b and c are *clock-equivalent*, written $b \sim c$, iff there exists a behavior d s.t. $d \leq b$ and $d \leq c$.

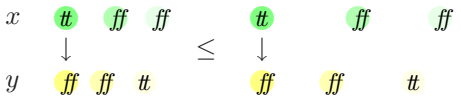

Figure 7. Stretching the clock of a behavior.

To model asynchrony, we consider a weaker relation which discards synchronization relations and allows for comparing behaviors w.r.t. the sequences of values signals hold. The *relaxation* relation allows to individually stretch the signals of a behavior. Relaxation is a partial-order relation that defines the flow-equivalence relation. Two behaviors are flow-equivalent iff their signals hold the same values in the same order. The behaviors b and c are *flow-equivalent*, written $b \approx c$, iff there exists a behavior d s.t. $d \sqsubseteq b$ and $d \sqsubseteq c$.

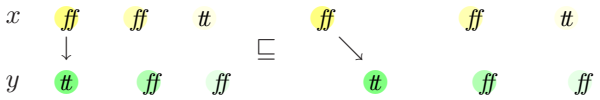

Figure 8. Relaxation of a behavior.

The model of polychrony provides a purely relational denotation of SIGNAL, defined in [16], consisting of the function $[\![P]\!] = p$ that associates a SIGNAL process P with the set p of its possible behaviors. Notice that the semantics of SIGNAL is stretch-closed: whenever a process P has a behavior b, written $b \in [\![P]\!]$, then it admits any stretching $c \in [\![P]\!]$ of b.

3.1.7. Formal design properties

The model of polychronous signals allows to define formal properties that are essential for the component-based design of GALS architectures [16]. *Input-endochrony* is a key design property. A process p is input-endochronous iff, given an external (asynchronous) stimulation of its inputs I[1], it reconstructs a unique synchronous behavior (up to clock-equivalence) i.e. $\forall b, c \in p,\ (b|_I) \approx (c|_I) \Rightarrow b \sim c$. An interpretation of

[1] We write $b|_X$ for the projection of a behavior b on a set $X \subset \mathcal{X}$ of names.

endochrony (figure 9) consists of considering any flow equivalent inputs $(b|_I) \approx (c|_I)$ of p and by observing that, among these flow equivalent inputs, an endochronous process only admits clock-equivalent behaviors $(b|_I) \sim (c|_I)$ by producing clock equivalent outputs $(b|_O) \sim (c|_O)$. In other words, controllability denotes the class of processes that are stallable or insensitive to (internal and) external propagation delays.

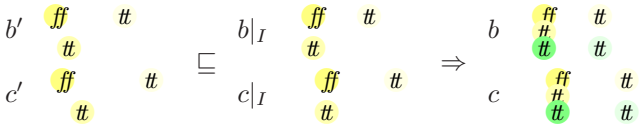

Figure 9. Endochrony.

Flow-invariance is the property that ensures that the refinement of a functional specification $p \,|\, q$ by an asynchronous implementation $p \parallel q$ preserves flow-equivalence. Formally, the refinement of $p \,|\, q$ by $p \parallel q$ is flow-invariant iff, for all $b \in p \,|\, q$, for all $c \in p \parallel q$, $(b|_I) \approx (c|_I)$ implies $b \approx c$ for I the input signals of $p \,|\, q$. In SIGNAL, GALS architectures are modeled as *endo-isochronously* communicating endochronous processes: Two endochronous processes p and q are endo-isochronous iff $(p|_I) \,|\, (q|_I)$ is endochronous (with $I = \mathrm{vars}(p) \cap \mathrm{vars}(q)$ the interface between p and q). Endo-isochrony implies flow-invariance and is, like controllability, amenable to static verification. [22] formally defines the synchronization and scheduling analysis of SIGNAL and gives decision procedures for controllability and endo-isochrony based on that (behavioral) type information.

4. Application Domains

The application domains covered by the platform POLYCHRONY, developed by the ESPRESSO project-team are engineering areas where a system design-flow requires high-level 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. This research track is being continued in the submitted ESPACE (*avionics*) and SEA (*automotive*) projects.

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. One of the most prominent recent developments resulting from work in this application area is the complete model of the APEX services library in SIGNAL. A variant of it is in use at SNECMA-Hispano-Suiza and AIRBUS Industries [33], [18].

Another prominent result of the project-team in the same context is the development of a real-time JAVA plug-in for POLYCHRONY [21], developed in the context of the RNTL project EXPRESSO. This tool enables the modeling of avionics applications in JAVA and allows for applying high-level transformation and verification on that model, using POLYCHRONY, and generate an optimized, serialized code, whose runtime subsystem footprint and generated scheduling are specific to the target architecture of the application.

Recent trends in *system-level design* show, in a far from unrelated way, the need for modeling systems on chips as globally asynchronous and locally synchronous systems. It is indeed manifest in the charter of the ACM-IEEE MEMOCODE conference [12]. It is the subject of an ongoing NSF-INRIA collaboration of the project-team with UC San Diego, UC Irvine and Virginia Tech.

5. Software

5.1. The Polychrony workbench

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

The POLYCHRONY toolset, which is registered at the APP, is now freely distributed (at binary level) for non-commercial use on the site http://www.irisa.fr/espresso/Polychrony. Based on the SIGNAL language, it provides a formal framework:

1. to validate a design at different levels,
2. to refine descriptions in a top-down approach,
3. to abstract properties needed for black-box composition,
4. to assemble predefined components (bottom-up with COTS).

It constitutes a development environment for critical systems, from abstract specification until deployment on distributed systems. It relies on the application of formal methods, allowed by the representation of a system, at the different steps of its development, in the SIGNAL polychronous semantic model. Based on the same polychronous principles, there is a commercial tool, SILDEX, provided by the TNI-VALIOSYS company (http://www.tni-valiosys.com).

POLYCHRONY is a set of tools composed of:

1. A SIGNAL batch compiler providing a set of functionalities viewed as a set of services for, e.g., program transformations, optimizations, formal verification, abstraction, separate compilation, mapping, code generation, simulation, temporal profiling...
2. a Graphical User Interface with interactive access to compiling functionalities.
3. The SIGALI tool, an associated formal system for formal verification and controller synthesis. SIGALI is jointly developed with the VERTECS project-team (http://www.irisa.fr/vertecs).

POLYCHRONY supports SIGNAL as a native, high-level, design language. It is also an open design platform that supports several input formalisms and notations and that is connected to other verification and design tools. For example:

1. in the context of the EXPRESSO project, a prototype[2] of a real-time JAVA plug-in [21] has been developed, demonstrating the capabilities of POLYCHRONY to model multi-threaded (i.e. GALS) architectures.
2. in the context of the ACOTRIS project, a prototype of UML to SIGNAL translation has been developed.

The POLYCHRONY platform offers services for modeling system behavior and architectures from high-level, heterogeneous notations and formalisms. Such models are hosted in POLYCHRONY using the polychronous data-flow notation SIGNAL. POLYCHRONY operates these model representation by performing transformations and optimizations on them (hierarchization of control, desynchronization protocol synthesis, separate compilation, clustering, abstraction) in order to deploy such system specifications on mission specific target architectures. The POLYCHRONY platform supports both component-based embedded system design, allowing for the capture and integration of existing components on a given architecture; and refinement-based design, allowing for a seamless upgrade of specifications and components toward deployment on specific target architectures while ensuring compliance to formal design properties. Finally, C, C++, multi-threaded and real-time JAVA and SYNDEX code generators are provided. The connection to the SYNDEX distribution tool (http://www-rocq.inria.fr/syndex) has been developed in the context of the ACOTRIS project.

[2] i.e. implementing the translation of a functional subset of real-time JAVA suitable for avionics application design: no dynamic resource allocations, no non-terminal recursion, etc.

5.2. The Apex services library

Participants: Abdoulaye Gamatié, Thierry Gautier.

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* (Figure 10).

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 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.

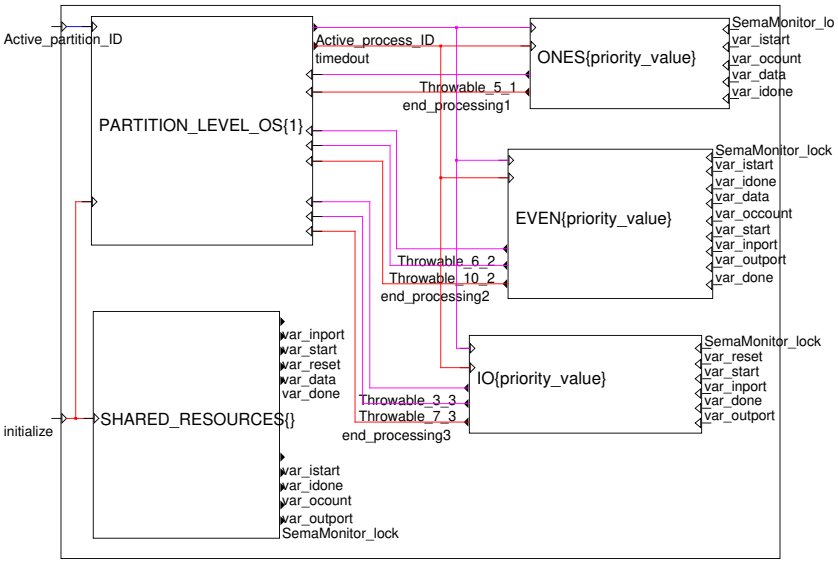

Figure 10. ARINC *partition-level model of an even-parity checker model in* SIGNAL

5.3. Real-time Java plug-in

Participants: Loïc Besnard, Bruno Le Dez, Luc-Michel Sévère, Jean-Pierre Talpin.

The real-time JAVA plug-in of POLYCHRONY is a tool which was developed during the frame of the RNTL project EXPRESSO (section 7.2). It consists of modeling a domain-specific subset of the real-time JAVA specification (for modeling avionics applications or control dominated embedded systems) in SIGNAL: thus it provides a compiler from this subset of RT-JAVA into SIGNAL. This model is obtained from a given JAVA class-file hierarchy (at either byte-code or source level) by, first, automatic modeling of the application

architecture using instances of the POLYCHRONY-APEX services and, second, by translating periodic/sporadic JAVA threads and event handlers in the data-flow polychronous design language SIGNAL [26].

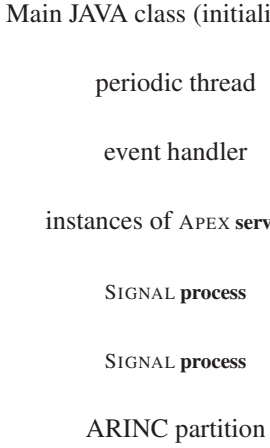

Figure 11. Architecture of the real-time JAVA *plug-in for* POLYCHRONY

The novelty of integrating POLYCHRONY in a high-level design tool-chain lies in the formal support offered by the former to automate critical and complex design verification and validation stages yielding a correct-by-construction system design and refinement in the latter. Polychronous design allows for an early requirements capture and a compositional and formally checked transformational refinement, automating the most difficult design steps toward implementation using efficient clock resolution and synthesis techniques, implemented in the SIGNAL compiler.

5.4. A model of Signal in Coq

Participants: Mickaël Kerbœuf, Jean-Pierre Talpin.

The verification of a reactive system is usually done by elaborating a *discrete* model of the system specified in a dedicated formalism and then by checking a property against the model. The use of formal proof systems enables to prove *hybrid properties* about *infinite state systems*: the *correctness* and the *completeness* of a reactive system.

To this aim, the ESPRESSO project-team has developed a complete model of the SIGNAL design language in COQ [40]. More precisely, we have defined a translation scheme of the trace semantics of SIGNAL to the logical framework of COQ. We have conducted several case studies to demonstrate the applicability of the approach to resolve sophisticated verification problems: a complete model and proof of the well-known steam-boiler problem [36], the correctness of an implementation of a SIGNAL protocol for loosely timed-triggered architectures [20].

Such a proof, of course, cannot always be done automatically: it requires human-interaction to direct the proof strategy. The prover can nonetheless automate its most tedious and mechanical parts. In general, formal proofs of programs are difficult and time-consuming. In the particular case of modeling a reactive system using SIGNAL, experience however shows that this difficulty is significantly reduced thanks to the combined declarative style of programming and a relational style of modeling.

6. New Results

6.1. Main contributions

The activity of the project-team this year is marked with several important milestones, rooting the foundations of the project-team on solid ground.

- **A polychronous model of computation.** The article [16], whose original aim was to synthesize the model and methods used in the POLYCHRONY platform, presents the first theoretical model of multi-clocked synchrony (polychrony) that is characterized by a semi-lattice structure of tagged traces (in the spirit of Lee et al. "*models of computation*" [39] that is closed by clock equivalence and flow equivalence. In this structure, we actually gave the first accurate characterization of key formal properties and decision procedures to refinement-based design: so called input-endochrony (controllability of a synchronous system from its input) and flow-invariance (invariance of a design refinement by flow-equivalence).
- **The release of the POLYCHRONY platform.** The work that yielded the release of the POLYCHRONY platform was an important investment of the ESPRESSO project-team, totaling four man/year since the project-team creation (January 2002). Since its first release on the website http://www.irisa.fr/espresso/Polychrony early February, the package has been registered by our close academic and industrial partners. The POLYCHRONY platform offers services for modeling system behavior and architectures from high-level, heterogeneous notations and formalisms, and implement services for the transformation and optimization of such specifications (hierarchization of control, desynchronization protocol synthesis, separate compilation, clustering) in order to deploy such specifications on mission specific target architectures.
- **A model of APEX services in POLYCHRONY.** The complete specification of the APEX services API in SIGNAL is undoubtedly the most demonstrative application of the polychronous model of computation implemented in the POLYCHRONY platform. It offers a complete implementation of the APEX communication and synchronization services, of the process management and partitioning services. Its SIGNAL implementation consists of a library of generic, parameterizable SIGNAL modules. It is used locally to model the real-time JAVA virtual machine.
- **A real-time JAVA plug-in for POLYCHRONY.** A complementary demonstration of the expressive capabilities of the POLYCHRONY platform and its APEX services library is the development of a real-time JAVA plug-in [21]. This plug-in consists of modeling a domain-specific subset of the real-time JAVA specification (for modeling avionics applications or control dominated embedded systems) by, first, modeling the application architecture using instances of APEX services and, second, translating periodic/sporadic JAVA threads and event handlers into the data-flow polychronous design language SIGNAL. The development of this tool is an effort of two man/year in the context of the RNTL EXPRESSO project together with its transfer to the TURBOJ compiler of SILICOMP (1 man/year).
- **Case-studies on high-level system design using POLYCHRONY.** In the context of an NSF-INRIA collaboration with UC San Diego, UC Irvine and Virginia Tech, we have conducted several case studies aiming at demonstrating the usability of POLYCHRONY as a semantics platform for compositional system design in SystemC. This work has been initiated through several visits/internships and preliminary results in this direction are reported in [23][22]. The platform POLYCHRONY is now in use in the BALBOA project in the aim of capturing behavioral abstractions of existing SystemC IP block descriptions, in order to correctly interface them using appropriate protocol synthesis techniques.
- **Events:** the creation of the ACM-IEEE sponsored MEMOCODE conference [12] and of the ACM supported FMGALS workshop [13] (section 9.2).

In addition, new results, related to on-going projects, have been accomplished on the connection of SIGNAL to UML and SYNDEX (sections 6.2 and 6.3), on modeling real-time applications and on evaluating their performances (section 6.4), on modeling system-level design languages (section 6.5) and on modeling weakly synchronized protocols (section 6.6).

6.2. A Signal plug-in for Uml-Accord

Participant: Thierry Gautier.

Following previous studies on the BDL model and, more specifically, on the translation of UML state diagrams into the BDL notation [42] (resp. STATEMATE into SIGNAL [31]), we have, with our partners of the ACOTRIS project (see 7.1), defined and experimented models of translation of applications specified using the UML/ACCORD methodology [38] to SIGNAL. A main objective is to take benefit from verification and code generation tools available in the synchronous model. Two models of translation have been defined.

The first one considers a strictly synchronous semantics of state-diagrams. More precisely: $O_t = f(I_t, S_t)$ and $S_t = f(I_t, S_{t-1})$, where I, O and S represent respectively inputs, outputs and state of the system. This is not in accordance with the execution semantics defined in UML, but has to be compared with the models considered in Argos or SyncCharts. It may be used as one element for a future proposal of a synchronous semantics of UML state-diagrams. A plug-in called YATUS, from UML/ACCORD to SIGNAL, following this approach, has been realized by CS with INRIA support.

The second model follows the UML asynchronous execution semantics, which considers that events are managed in queues. One queue of events is associated with each state-diagram representing the behaviour of an "active" real-time object. A basic hypothesis for this translation is that queues of events are bounded. Then, the state-diagrams, the handling of the queues, and the language of actions are translated into SIGNAL [27]. A plug-in called GASP, from UML/ACCORD to SIGNAL, following the UML semantics, has been realized by CS and CEA-List with INRIA support.

Both approaches, which have been validated using applications in the ACOTRIS project, should be more deeply compared. Another direction, not yet explored, for a UML/SIGNAL connection, would be the possible use of the OCL notation to specify polychronous properties.

6.3. A Signal plug-in for SynDEx

Participants: Loïc Besnard, Thierry Gautier, Qiuling Pan.

The SYNDEX environment (http://www-rocq.inria.fr/syndex, AOSTE project-team) aims at rapid prototyping and optimization of real-time embedded applications on multicomponent architectures. Its goal is to find an optimized implementation of an application algorithm on an architecture, while satisfying constraints.

In the ACOTRIS project (see 7.1), we have developed a translator from SIGNAL to SYNDEX, for the software part of an application. This translator is realized along two axes:

- The first one (semantic axis) is the clock hierarchy: a SYNDEX *algorithm* is associated with each clock of the hierarchy.
- The second one (syntactic axis) is composed of the set of SIGNAL processes which have to be kept as units of structuration (SYNDEX algorithms) in the generated SYNDEX code.

The main add-ons of the translator are the following:

- It gives SIGNAL designs access to the functionalities of SYNDEX, in particular the possibility to get and to prototype distributed implementations obtained from quantitative criteria;
- It enables SIGNAL as possible input formalism for SYNDEX users;
- It allows the use of SYNDEX on applications developed in formalisms for which it is of interest to have a SIGNAL intermediate representation.

Future extensions of the translator will take into account the hardware architecture part.

6.4. Modeling and evaluation of distributed real-time applications

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

A way for modeling distributed real-time applications using SIGNAL consists in the definition of a set of basic components required for such applications. For that purpose, we described various kinds of components. A large part of these components has been defined based on specifications from the avionics standard ARINC 653 [34] [33]. This standard relies on the IMA approach (Integrated Modular Avionics). It gives the basic services that form the API between the application software and the core software (the operating system and system specific functions), within a system. They are commonly called APEX services (APplication EXecutive). The standard also defines the basic executive entities: *partitions* and *processes* (see 5.2). Among APEX services, one can mention for instance, those which achieve communications between partitions as well as processes, or the management of processes and time.

We studied how these components can be used within the POLYCHRONY platform. In such a way, one can take advantage of available formal tools and techniques. We first illustrated the general approach to the design of basic components [18] [19]. Then, we focused on how they can be used in order to model real-time applications.

A general methodology for the design of distributed real-time applications already exists within POLYCHRONY (earlier studies about this methodology began during the european project Sacres [35]). Basically, the design of such an application consists in the distribution of an initial SIGNAL program representing the application on a target architecture (composed of a set of possibly heterogeneous execution components, e.g. processors, micro-controllers). A general observation is that the level of detail at which the architecture needs to be known depends quite a lot on the refinement of the mapping to the chosen architecture. This means that in the simplest cases, the amount of data required is fairly small, and simple to assess:

- the set of processors or tasks, and the mapping from the SIGNAL program sub-parts of the application to those processors or tasks;
- the topology of the network of processors, the set of connections between processors, and a mapping from inter-process communications to these communication links;
- a definition of the set of system-level primitives used e.g. for communications (readings and writings to the media).

Further degrees of refinement of the description may be required for a better architecture-adaptation: for example, concerning communications, the type and nature of the links (that could be implemented using shared variables, synchronous or asynchronous communications). If the target architecture features an OS, the required model consists basically in the profile of the corresponding functions. For instance, according to the degree of use of the OS, we need models of synchronization gates, communications (possibly including routing between processors) or task management services (e.g. **start, stop, suspend and resume**).

The components we have described in [18] [19] provide a means to model most of the above features in order to get detailed models of applications: communication and synchronization mechanisms, system-level functions, execution entities (like process and partitions). A sample case study of avionic application has been modeled in SIGNAL using APEX components [17] (the considered example takes its inspiration from a real world avionics application). The aim was first to show the feasibility of describing avionics applications using our SIGNAL models. Then, the application model can be formally analyzed for various purposes. In particular, we focus on timing issues which are critical in the design of real-time systems. So, we illustrate how timing issues are addressed, e.g. to compute worst case execution times on the resulting SIGNAL model. For that, we use a technique based on transformations of SIGNAL programs [37], which yields a "temporal interpretation" associated with an initial program. Roughly, a SIGNAL program that models an application is recursively composed of sub-programs, where elementary sub-programs belong to the language kernel and called *atomic nodes*. A profiling of such a process substitutes each signal x with a new signal representing availability dates $date\_x$ and automatically replaces atomic nodes with their timing model counter-part ("timing" morphism).

The resulting time model is composed (by standard synchronous composition) with the original functional description of the application, and for each signal x, a synchronization with the signal $date\_x$ is added. The resulting process is close to (or even represents exactly) the model of the temporal behavior of the application running on its actual architecture. One can obviously design less strict modeling to get faster simulation (or formal verification); it is sufficient to consider more abstract representations either of the architecture or of the program.

Combining our component models and the above performance evaluation technique within POLYCHRONY, the whole design framework (from specification to verifications and analysis) relies on the use of the single formalism of the SIGNAL language.

6.5. Polychronous model of system-level design in SystemC

Participants: David Berner, Paul Le Guernic, Jean-Pierre Talpin.

SYSTEMC is a system modeling and specification language widely supported by industry and a growing user base. It is based on the C++ language, extending it with classes for the modeling of concurrency, time, communication, reactivity, hardware data types, and a growing number of features that ease design entry, testing and verification. Major advantages are the easy system specification, the usage of standard C++ tools such as compiler and debugger, and the ability to describe both the hardware and the software part of the system.

The usage of standard (and highly developed) compilers such as GCC and the ability to treat high levels of abstraction result in simulation speeds, orders of magnitude higher than those of conventional VHDL or VERILOG simulation. For big designs however, simulation speed still is a major hindering to exhaustive system testing and validation. While looking at the SYSTEMC compilation flow, we notice that the scheduling part is far from optimal. Signals in SystemC sensitivity lists are evaluated more often than actually necessary.

With the appropriate dependency and clock hierarchy information available in a SIGNAL description, we aim at obtaining an optimized scheduling and significantly higher simulation speeds. To achieve this goal, we are targeting to transform a valid SYSTEMC description into a SIGNAL description. This is done in three steps. First we determine the threading and synchronization structure of the program. This can then be transformed quite straightforward into SIGNAL skeleton code. SYSTEMC blocks that do not contain any synchronization points can be included into the SIGNAL skeleton as is, with the help of PRAGMA statements.

```
process epc = (? integer in_data; boolean start; ! boolean out_data; boolean done;)
 spec (| start --> out_data
       | start --> done
       | done ^= out_data
       | in_data ^= when (start=true)
       |)
 pragmas CPP_CODE "&o1 = EPC_SPECC(&i1,&i2,&o2)"
 end pragmas
```

Such blocks will be treated for the scheduling as black boxes of instantaneous execution, thus allowing for a much more selective evaluation of signals in sensitivity lists and giving raise to significant scheduling and therefore simulation improvements. In order to improve security and to help the detection of errors, we are currently investigating an interface description of these blocks of instantaneous execution with the help of SIGNAL processes. These interfaces would expose signal dependency and clock synchronization information and form a behavioral type description. The composition of such components can then be checked for correctness with a type checker, and the correctness of the component itself can then be verified with the help of a model checker. This part is ongoing joint research together with the FERMAT group of Virginia Tech. Synchronous modeling of SYSTEMC descriptions will combine the ease of description and the expressiveness of SystemC with the scheduling, dependency calculation, formal typing, and verification capabilities of synchronous languages, notably SIGNAL.

6.6. Formal proofs of polychronous protocols with Coq

Participants: Mickaël Kerbœuf, Jean-Pierre Talpin.

A protocol for loosely time-triggered architectures (or LTTA) has been proposed [32] to provide an abstract synchronous specification on top of real-time architectures. This abstract model is designed so as to satisfy the synchronous hypotheses and meet the implementation architecture constraints. It makes it possible to design, specify and verify reactive systems in the context of the synchronous approach. The robustness of the protocol has been proved manually and automatically with LUSTRE and SIGNAL in [32]. We proposed a new formalization of the protocol in COQ, a proof-assistant for higher-order logics. We aimed at defining a most general model of the protocol, founded on the least set of assumptions about the physical characteristics of the architecture. The manual and the automatic proofs of the robustness of the protocol can be considered as a refinement of the COQ model. These results have been published at the 5th. International conference on Formal Engineering Methods [20].

6.6.1. Loosely Time-Triggered Architectures.

A loosely time-triggered architecture is a physical system in which:

- Bus access is quasi-periodic and non-blocking;
- Read and write operations are independent;
- Values are sustained by the bus and periodically refreshed.

The LTTA is composed of three devices, a *writer,* a *bus,* and a *reader*. Each device $\mathbf{d}$ is activated by its own, approximately periodic, clock (denoted by a function $t^{\mathbf{d}}$). At the nth clock tick (time $t^{\mathbf{w}}(n)$), the *writer* generates the value $x^{\mathbf{w}}(n)$ and an alternating flag $b^{\mathbf{w}}(n)$. Both values are stored in its output buffer, denoted by $y^{\mathbf{w}}$. At any time t, the writer's output buffer $y^{\mathbf{w}}$ contains the last value that was written into it. At $t^{\mathbf{b}}(n)$, the *bus* fetches $y^{\mathbf{w}}$ to store in the input buffer of the reader, denoted by $y^{\mathbf{b}}$. At $t^{\mathbf{r}}(n)$, the *reader* loads the input buffer $y^{\mathbf{b}}$ into the variables $x(n)$ and $b(n)$. Then, in a similar manner as for an alternating bit protocol, the reader extracts $x(n)$ iff $b(n)$ has changed.
writer reader $\cdot$ sustain $y^{\mathbf{b}}$ $\cdot$ bus$x^{\mathbf{w}}$ $t^{\mathbf{w}}$ $y^{\mathbf{w}} = (x^{\mathbf{w}}, b^{\mathbf{w}})$ $x^{\mathbf{r}} = x$ $t^{\mathbf{r}}$ $y^{\mathbf{r}} = (x, b)$ $t^{\mathbf{b}}$

6.6.2. Correctness of the protocol.

In any execution of the protocol, the sequences $x^{\mathbf{w}}$ and $x^{\mathbf{r}}$ must coincide, i.e.,

$$\forall n \cdot x^{\mathbf{r}}(n) = x^{\mathbf{w}}(n) \tag{1}$$

In order to prove the correctness of the protocol, we need to prove that, under some hypotheses on the clocks, the property (1) is true.

6.6.3. Abstraction and formalization in Coq

The translation in COQ of the specifications is quite straightforward. Physical time is seen as an abstract data type i.e., a type $\mathcal{T}$, a binary predicate $\leq$ and the assumption that $\leq$ is reflexive, transitive and total. A clock c is modeled by a function which maps any natural number n in its domain to the instant $t \in \mathcal{T}$ when the nth sampling tick occurs. The only assumption on this function is that it is strictly monotonic.

We introduce a function called $read\_index$ which maps to a given reading tick k the writing tick $read\_index(k)$ corresponding to the instant (on the writer's clock) when the writer emitted the value that can be read at the instant k (on the reader's clock). To prove the correctness of the protocol, we prove that $read\_index$ is increasing, and that it covers $\mathbb{N}$ (so that all written values are actually read). Thus, all written values are actually read (and possibly more than once) in a correct order. The property (1) follows when $\forall k \in \mathbb{N}, b^{\mathbf{w}}(k{+}1) \neq b^{\mathbf{w}}(k)$. It is actually the case with the alternating bit protocol. This result holds under the following specific condition (where $\tau^{\mathbf{b}}(n)$ stands for the first instant where the bus can fetch the nth writing):

$$\forall n \in \mathbb{N},\ \ \exists k \in \mathbb{N},\ \ \text{st.}\ \ \tau^{\mathbf{b}}(n) \ < \ t^{\mathbf{r}}(k) \ \leq \ \tau^{\mathbf{b}}(n{+}1)$$

6.6.4. A formal framework for any implementation

The COQ encoding of the protocol for LTTAs is founded on the smallest set of physical requirements and logical requirements. Thus, any implementation must provide at least these requirements. Its correctness then follows. We refine this approach by adding an intermediate level between COQ and the analyzed implementation. This interface details the expected form of the time domain (variable $\mathcal{T}$ in COQ and its order $\leq$), and the clocks. To prove the correctness of any implementation built upon the model denoted by the intermediate level, we only have to prove that its specification implies the assumptions of its interface.

6.6.5. Future work

An attractive aspect of the use of COQ is the extraction of a reference implementation of the protocol. The only difficulty is that this protocol involves partial function that are difficult to deal with in COQ[3]. We also plan to connect the COQ model of SIGNAL (cf. section 6.6) with this approach, to prove the correctness of protocols stated in SIGNAL.

7. Contracts and Grants with Industry

7.1. Rntl project Acotris, no2 00 C 0527 00 31307 01 1 (02/2001-07/2003)

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

The partners of the RNTL **project** ACOTRIS (http://www.acotris.c-s.fr) are CS-SI, CEA-List, MBDA (ex. Aérospatiale Matra Missiles), SITIA and INRIA (project-teams ESPRESSO and AOSTE). The project proposes a methodology and a systematic approach (with tools support) to rationalize the development phases of real-time embedded systems. The objectives are to help with the complete specification of the need and the design of real-time applications, while integrating:

- an analysis and design methodology based on a standard UML formalism (UML with ACCORD method from CEA),
- a design and realization methodology based on the synchronous model (SIGNAL and AAA/SYNDEX method from INRIA).

The goal is to assist the designers of real-time multi-task applications during the co-design process by a quasi complete automation of this process. For that purpose, the project adapts and connects the existing tools by developing "plugins".

UML is used for modeling, SIGNAL is used for software validation, and finally, SYNDEX is used for hardware/software "adequation" (efficient matching). This is illustrated on figure 12 [25].

The following tools plugins have been developed during the project:

- UML to SIGNAL (for the software architecture),
- SIGNAL to SYNDEX (for the software architecture, too),
- UML to SYNDEX (for the hardware architecture).

[3] http://pauillac.inria.fr/pipermail/coq-club/2002/thread.html#569

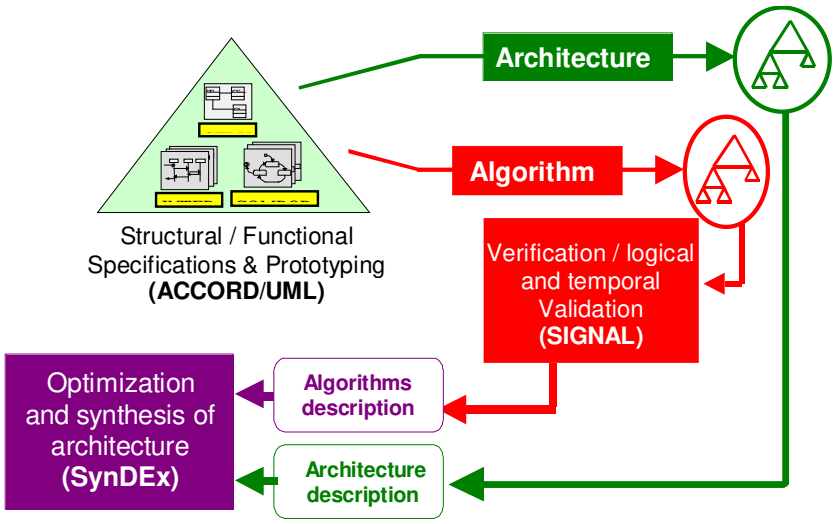

Figure 12. The ACOTRIS *approach*

7.2. Rntl project Expresso, no2 01 C 0299 00 31307 01 1 (06/2001-05/2003)

Participants: Bruno Le Dez, Abdoulaye Gamatié, Paul Le Guernic, Jean-Pierre Talpin.

The partners of the RNTL **project** EXPRESSO (http://www.irisa.fr/rntl-expresso) are AONIX, SILICOMP, THALÈS, EDF, AIRBUS, VERIMAG. The contribution of the ESPRESSO project-team consists of the definition and implementation of the real-time JAVA plug-in for POLYCHRONY, seen as a high-level transformation and verification tool for domain-specific real-time JAVA programs in avionics (described in section 5.3).

7.3. Caroll project Protes (10/2003-10/2005)

Participants: Thierry Gautier, Jean-Pierre Talpin.

The partners of the **CAROLL project** PROTES (which started October 2003) are THALES, CEA-List and the INRIA project-teams ESPRESSO, AOSTE and DART. The aim of the project PROTES is to propose a UML profile for real-time and embedded systems and to defend it before the OMG.

8. Other Grants and Activities

8.1. Nsf-Inria program

Participants: David Berner, Paul Le Guernic, Jean-Pierre Talpin.

In the frame of the NSF-INRIA Cooperation of the ESPRESSO project-team with Virginia Tech and UC San Diego, David Berner is currently spending three months at Virginia Tech (October-December 2003). He is working with Prof. Dr. Sandeep Shukla in the FERMAT (Formal Engineering Research using Methods, Abstraction, and Transformations) group of the ECE (Electrical and Computer Engineering) department. The main goal of this exchange is to advance in the definition of behavioral interface descriptions for SYSTEMC program blocks. The interface of a SYSTEMC block can be described as a SIGNAL process, containing signal dependency and clock synchronization information. This information (commonly not used in, nor available to a SYSTEMC compiler), will form the type of this component. If we have sound type information for the components available, type checking will expose problems in the model that would otherwise have been detected much later in the design process. Also types can be the basis of the automatic generation and insertion

of interfaces, which would significantly facilitate the development process. Finally the implementation of a component can be formally checked against its type with the help of existing model checkers such as SPIN. Our common topic of investigation, the typing of SYSTEMC components, turns out to be a key capability for the speedup, cost reduction, and formalization for the design process of embedded systems.

8.2. Network of excellence Artist

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

The ESPRESSO project-team is involved in the activity of the ARTIST network of excellence (http://www.artist-embedded.org). This activity essentially consists of the definition of a road-map on embedded system design [24].

9. Dissemination

9.1. Advisory

- Paul Le Guernic is executive board member of the Réseau National en Technologies Logicielles and steering committee member of the Réseau National en Micro-Nano Technologies.
- Paul Le Guernic and Jean-Pierre Talpin are steering committee member of the ACM-IEEE conference on methods and models for codesign (MEMOCODE).
- Jean-Pierre Talpin is a member of the external advisory board for the center of embedded systems at Virginia Tech.
- Jean-Pierre Talpin is member of the organization committee of the FMGALS (formal methods for GALS architectures) workshop series.

9.2. Conferences

Creation of the ACM-IEEE MEMOCODE **conference.** The project-team initiated the joint ACM-IEEE conference on formal methods and models for codesign, MEMOCODE'2003, chaired by Pr. Gupta, Pr. Shukla, J.-P. Talpin and P. Le Guernic (http://www.irisa.fr/MEMOCODE). The main topics of high-level system design, of hierarchical and incremental verification, of conformance checking, are central and key issues from an industrial point of view. The call for papers is in complete synergy with the demand of the industry majors in the area and calls for contributions to five industrial cases studies, proposed by researchers from Synopsys, Motorola, Cadence, Verplex Systems, Mentor Graphics, and from the SystemC consortium.

- Jean-Pierre Talpin served as technical program co-chair of MEMOCODE'2003.
- Paul Le Guernic served as technical program committee member of MEMOCODE'2003.
- Loïc Besnard managed the technical support for the MEMOCODE'2003 conference.
- Jean-Pierre Talpin served as technical program co-chair for the FME'2003 workshop on formal methods for GALS architectures (FMGALS'2003).
- Jean-Pierre Talpin served as technical program committee member for the IEEE DATE'2003 conference.
- Thierry Gautier served as technical program committee member for the EUROMICRO workshop SLAP'2003.

9.3. Events

- Jean-Pierre Talpin gave a tutorial on "High-level modeling and validation methodologies for embedded systems—bridging the productivity gap" at the IEEE conference on *VLSI design*, January 2003.
- Loïc Besnard and Abdoulaye Gamatié participated to the 10th. INRIA-Industry meeting "Applications de l'informatique et de l'automatique aux transports", January 2003.
- Thierry Gautier participated to the workshop "UML model checking" (IRIT, Toulouse, January 2003), with a presentation on "ACOTRIS: Real-Time and Model Checking".
- Jean-Pierre Talpin gave a seminar on "Formal refinement-checking in an embedded system design methodology" at the Inaugural of the Embedded Systems Center at Virginia Tech, October 2003.
- Jean-Pierre Talpin defended his HDR thesis [14] at IRISA on November 25th.

9.4. Teaching

- Thierry Gautier, Bernard Houssais and Loïc Besnard teached on real-time programming at the DESS-ISA and DIIC 2 of University of Rennes I.
- Abdoulaye Gamatié gave courses as teaching assistant at the University of Rennes I.
- Mickaël Kerbœuf gave courses as teaching assistant at INSA-Rennes.
- Thierry Gautier and Loïc Besnard managed a team of DIIC 3 LSI/ARC students for a mini-project on the implementation of a VHDL code generator for SIGNAL.

9.5. Visits

- Pr. Sandeep Kumar Shukla (Virginia Tech) visited the project-team as Invited Professor of the University of Rennes in July 2003.
- Mohammad Reza Mousavi (Eindhoven University) visited the project-team from June 1st to July 15th.
- The project-team invited Dr. Grant Martin (Cadence) for a conference at IRISA on "A brief history of the SoC revolution" June 23rd, 2003.
- The project-team invited Pr. Rajesh Gupta UC San Diego) for a conference at IRISA on "Online Strategies for Power and Performance Management in Embedded Systems" June 27rd, 2003.

10. Bibliography

Major publications by the team in recent years

[1] A. BENVENISTE, B. CAILLAUD, P. LE GUERNIC. *From synchrony to asynchrony.* in « CONCUR'99, Concurrency Theory, 10th International Conference », series Lecture Notes in Computer Science, volume 1664, Springer, J. C. M. BAETEN, S. MAUW, editors, pages 162–177, August, 1999.

[2] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT. *Synchronous programming with events and relations: the* SIGNAL *language and its semantics.* in « Science of Computer Programming », volume 16, 1991, pages 103-149.

[3] T. GAUTIER, P. LE GUERNIC. *Code generation in the SACRES project.* in « Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99 », Springer, F. REDMILL, T. ANDERSON, editors, pages 127–149, Huntingdon, UK, February, 1999, ftp://ftp.irisa.fr/local/signal/publis/articles/SSS-99:format_dist.ps.gz.

[4] A. KOUNTOURIS, C. WOLINSKI. *High-level Pre-synthesis Optimization Steps using Hierarchical Conditional Dependency Graphs.* in « Proceedings of the EUROMICRO'99 », IEEE Computer Society Press, Milan, Italie, August, 1999.

[5] A. KOUNTOURIS, P. LE GUERNIC. *Profiling of* SIGNAL *Programs and its Application in the Timing Evaluation of Design Implementations.* in « Proc. of the IEE Colloq. on HW-SW Cosynthesis for Reconfigurable Systems », IEE, pages 6/1–6/9, Février, 1996, ftp://ftp.irisa.fr/local/signal/publis/articles/HWSWCRS-96:profiling.ps.gz.

[6] 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 », number 4, volume 10, October, 2000, pages 347–368.

[7] H. MARCHAND, M. SAMAAN. *On the Incremental Design of a Power Transformer Station Controller using Controller Synthesis Methodology.* in « FM'99 — Formal Methods », series Lecture Notes in Computer Science, volume 1709, Springer, J. M. WING, J. WOODCOCK, J. DAVIES, editors, pages 1605–1624, Toulouse, France, September, 1999, ftp://ftp.irisa.fr/local/signal/publis/articles/FM99:control_synth_appli.ps.gz.

[8] 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, pages 163–173, 1995, ftp://ftp.irisa.fr/local/signal/publis/articles/PLDI-95:compil.ps.gz.

[9] T. P. AMAGBEGNON, P. LE GUERNIC, H. MARCHAND, E. RUTTEN. *Signal- the specification of a generic, verified production cell controller.* in « Formal Development of Reactive Systems - Case Study Production Cell », series Lecture Notes in Computer Science, number 891, Springer Verlag, C. LEWERENTZ, T. LINDNER, editors, pages 115-129, January, 1995.

[10] 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, pages 413–438, 1991, ftp://ftp.irisa.fr/local/signal/publis/articles/ATDFC-91:sem_distr.ps.gz.

[11] P. LE GUERNIC, T. GAUTIER, M. LE BORGNE, C. LE MAIRE. *Programming Real-Time Applications with Signal.* in « Proceedings of the IEEE », number 9, volume 79, Septembre, 1991, pages 1321–1336, ftp://ftp.irisa.fr/local/signal/publis/articles/ProcIEEE-91:gen_lang.ps.gz.

Books and Monographs

[12] *Proceedings of the 1st.* ACM-IEEE *conference on methods and models for codesign (*MEMOCODE*).* S. K. SHUKLA, J.-P. TALPIN, editors, IEEE Press, June 2003, http://www.irisa.fr/MEMOCODE.

[13] *Proceedings of the 1st* ACM-FME *workshop on formal methods for globally asynchronous locally synchronous architectures (*FMGALS*).* S. K. SHUKLA, J.-P. TALPIN, editors, INRIA, September 2003,

http://fermat.ece.vt.edu/fmgals03.

Doctoral dissertations and "Habilitation" theses

[14] J.-P. TALPIN. *Méthodes formelles pour la modélisation et l'analyse de systèmes.* Ph. D. Thesis, Habilitation à diriger des recherches, Université de Rennes 1, December 2003.

Articles in referred journals and book chapters

[15] 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 », volume 91(1), 2003.

[16] 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.

Publications in Conferences and Workshops

[17] A. GAMATIÉ, T. GAUTIER, L. BESNARD. *Modeling of avionics applications and performance evaluation techniques using the synchronous language SIGNAL.* in « Synchronous Languages, Applications, and Programming », series Electronic Notes in Theoretical Computer Science, volume 88, Elsevier Science, 2003.

[18] A. GAMATIÉ, T. GAUTIER. *The SIGNAL approach to the design of system architectures.* in « 10th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems », IEEE Press, April 2003.

[19] A. GAMATIÉ, T. GAUTIER. *Synchronous modeling of avionics applications using the SIGNAL language.* in « 9th IEEE Real-time/Embedded technology and Applications symposium », IEEE Press, May 2003.

[20] M. KERBOEUF, D. NOWAK, J.-P. TALPIN. *Formal proof of a polychronous protocol for loosely time-triggered architectures.* in « Formal Methods and Software Engineering: 5th International Conference on Formal Engineering Methods », series Lecture Notes in Computer Science n. 2885, Springer Verlag, November 2003.

[21] J.-P. TALPIN, A. GAMATIÉ, D. BERNER, B. LE DEZ, P. LE GUERNIC. *Hard real-time implementation of embedded software in JAVA.* in « International Workshop on scientiFic engIneering of Distributed Java applIcations », series Lecture Notes in Computer Science, Springer Verlag.

[22] J.-P. TALPIN, P. LE GUERNIC, S. K. SHUKLA, R. GUPTA, F. DOUCET. *Polychrony for formal refinement-checking in a system-level design methodology.* in « 3rd Conference on Application of Concurrency to System Design », IEEE Press, June 2003.

[23] J.-P. TALPIN, P. LE GUERNIC, S. K. SHUKLA, R. GUPTA, F. DOUCET. *Polychrony for refinement-based design.* in « Design, Automation and Test in Europe », IEEE Press, March 2003.

Internal Reports

[24] A. BENVENISTE, ET AL.. *Hard Real-Time Development Environments.* Technical report, ARTIST, Project IST-2001-34820, May 2003, http://www.artist-embedded.org/Roadmaps/A1-roadmap.pdf.

[25] M. NAKHLÉ, Y. TANGUY, F. TERRIER, T. GAUTIER, C. MODIGUY, A. LAVERRE, C. SCHLOSSERS, P. PICARD. *ACOTRIS – Rapport scientifique final.* Technical report, CS Systèmes d'Information, December 2003.

[26] J.-P. TALPIN, B. LE DEZ, A. GAMATIÉ, P. LE GUERNIC, D. BERNER. *Component-based engineering of real-time JAVA applications on a polychronous design platform.* Technical report, number 4744, INRIA, February 2003, http://www.inria.fr/rrrt/rr-4744.html.

[27] Y. TANGUY, S. GÉRARD, T. GAUTIER. *Modélisation ACCORD/UML – SIGNAL.* Technical report, number DTSI/SLA/03-621, CEA/Saclay, Direction de la recherche technologique, October 2003.

Bibliography in notes

[28] T. P. AMAGBEGNON, L. BESNARD, P. LE GUERNIC. *Implementation of the data-flow synchronous language SIGNAL.* in « Conference on Programming Language Design and Implementation », ACM Press, 1995.

[29] *Design Guidance for Integrated Modular Avionics.* Technical report, number ARINC Specification 651-1, Airlines Electronic Engineering Committee, November 1997.

[30] *Avionics Application Software Standard Interface.* Technical report, number ARINC Specification 653, Airlines Electronic Engineering Committee, January 1997.

[31] J.-R. BEAUVAIS, E. RUTTEN, T. GAUTIER, P. LE GUERNIC, Y.-M. TANG. *Modelling Statecharts and Activitycharts as Signal equations.* in « Transactions on Software Engineering and Methodology », volume 10(4), 2001.

[32] A. BENVENISTE, P. CASPI, P. LE GUERNIC, H. MARCHAND, J.-P. TALPIN, S. TRIPAKIS. *A protocol for loosely time-triggered architectures.* in « Embedded Software Conference », series Lecture Notes in Computer Science, Springer Verlag, 2002.

[33] A. GAMATIÉ, T. GAUTIER. *Modeling of modular avionics architectures using the synchronous language SIGNAL.* in « Proceedings of the 14th. Euromicro Conference on Real-Time Systems, work-in-progress session », IEEE Press, 2002.

[34] A. GAMATIÉ, T. GAUTIER.. *Synchronous modeling of modular avionics architectures using the SIGNAL language.* Technical report, number 4678, INRIA, December 2002.

[35] T. GAUTIER, P. LE GUERNIC. *Code generation in the SACRES project.* in « Proceedings of the Safety-critical Systems Symposium », Springer Verlag, February 1999.

[36] M. KERBŒUF, D. NOWAK, J.-P. TALPIN. *The steam-boiler problem in SIGNAL-COQ.* in « International

Conference on Theorem Proving in Higher-Order Logics », series Lecture Notes in Computer Science, Springer Verlag, October 2000.

[37] A. KOUNTOURIS, P. LE GUERNIC. *Profiling of SIGNAL programs and its application in the timing evaluation of design implementations.* in « IEE Colloqium on HW-SW Cosynthesis for Reconfigurable Systems », IEE, February 1996.

[38] A. LANUSSE, S. GÉRARD, F. TERRIER. *Real-Time Modeling with UML: the ACCORD Approach.* in « UML98: Beyond the Notation », series Lecture Notes in Computer Science, Springer Verlag, October 1998.

[39] E. LEE, A. SANGIOVANNI-VINCENTELLI. *A framework for comparing models of computation.* in « Transactions on computer-aided design », volume 17(12), December 1998.

[40] D. NOWAK, J.-R. BEAUVAIS, J.-P. TALPIN. *Co-inductive axiomatization of a synchronous language.* in « International Conference on Theorem Proving in Higher-Order Logics », series Lecture Notes in Computer Science, Springer Verlag, October 1998.

[41] D. NOWAK, J.-P. TALPIN, P. LE GUERNIC. *Synchronous structures.* in « International Conference on Concurrency Theory », series Lecture Notes in Computer Science, Springer Verlag, August 1999.

[42] Y. WANG. *UML et technologie synchrone pour les systèmes réactifs distribués.* Ph. D. Thesis, Thèse de doctorat de l'Université de Rennes 1, December 2001.