IN PARTNERSHIP WITH:
CNRS

Institut polytechnique de Grenoble

Université Pierre Mendes-France (Grenoble 2)

Université Joseph Fourier (Grenoble 1)

Activity Report 2011

Project-Team POP ART

Programming languages, Operating Systems, Parallelism, and Aspects for Real-Time

IN COLLABORATION WITH: Laboratoire d'Informatique de Grenoble (LIG)

RESEARCH CENTER
Grenoble - Rhône-Alpes

THEME
Embedded and Real Time Systems

Table of contents

Project-Team POP ART

Keywords: Aspect Oriented Programming, Embedded Systems, Fault Tolerance, Scheduling, Verification

1. Members

Research Scientists
Alain Girault [Team Leader, DR INRIA, HdR]
Pascal Fradet [CR INRIA, HdR]
Gregor Goessler [CR INRIA]
Bertrand Jeannet [CR INRIA]

Faculty Member
Gwenaël Delaval [Associate professor, Université Joseph Fourier]

External Collaborators
Emil Dumitrescu [Associate Professor, INSA Lyon]
Xavier Nicollin [Associate Professor, Grenoble INP]

PhD Students
Vagelis Bebelis [CIFRE STMicroelectronics, since 12/2011]
Peter Schrammel [INRIA, SYNCHRONICS project]
Gideon Smeding [DIGITEO grant]
Marnes Hoff [INRIA grant, AUTOCHEM project, until 05/2011]
Henri-Charles Blondeel [INRIA, until 06/2011]
Lies Lakhdar-Chaouch [INRIA, OPENTLM project, until 10/2011]

Post-Doctoral Fellows
Roopak Sinha [CESAR project]
Lacramioara Astefanoaei [ERCIM grant]
Petro Poplavko [INRIA and PILSI/CRI, until 11/2011]
Sebti Mouelhi [VEDECY project, since 10/2011]
Pascal Sotin [ASOPT project, until 09/2011]

Administrative Assistant
Diane Courtiol [Secretary INRIA]

2. Overall Objectives

2.1. Overall Objectives

We work on the problem of the safe design of real-time control systems. This area is related to (discrete) control theory as well as computer science. Application domains are typically safety-critical systems, as in transportation (avionics, railways), production, medical, or energy production systems. These application domains require both formal methods and models for the construction of correct systems, as well as their implementation in computer assisted design tools, targeted to specialists of the applications. We contribute to this research domain by offering solutions all along the design flow, from the specification to the implementation: we develop techniques for the specification, the programmation and the automated generation of safe real-time executives for control systems, as well as static analysis techniques to check additional properties on the generated systems. Our research themes concern:

- implementations of synchronous reactive programs, generated automatically by compilation, particularly from the point of view of automatic distribution and fault tolerance;

- high-level design and programming methods, with support for automated code generation, including: the automated generation of correct controllers using discrete control synthesis, compositionality for the verification and construction of correct systems; reactive programming, and aspect-oriented programming;
- static analysis and abstract interpretation techniques, which are applied both to low-level synchronous models/programs and to more general imperative or concurrent programs; this includes the verification of general safety properties and the absence of runtime errors.

Our applications are in embedded systems, typically in the robotics, automotive, and telecommunications domains with a special emphasis on dependability issues (*e.g.*, fault tolerance, availability). International and industrial relations feature:

- an IST European FP7 network of excellence: ARTISTDESIGN [1], on embedded real-time systems;
- an FP7 European STREP project: COMBEST [2] on component-based design;
- an ARTEMISIA European project: CESAR [3] on cost-efficient methods and processes for safety relevant embedded systems;
- three ANR French projects: ASOPT (on static analysis), AUTOCHEM (on chemical programming), and VEDECY (on cyber-physical systems);
- a MINALOGIC Pôle de Compétitivité project: OPENTLM, dedicated to the design flow for next generation SoC and SystemC;
- an INRIA large scale action: SYNCHRONICS on a language platform for embedded system design;
- an INRIA associated team with the University of Auckland (New Zealand), called AFMES [4] on advanced formal methods for embedded systems.

3. Scientific Foundations

3.1. Embedded systems and their safe design

3.1.1. Safe Design of Embedded Real-time Control Systems

The context of our work is the area of embedded real-time control systems, at the intersection between control theory and computer science. Our contribution consists of methods and tools for their safe design. The systems we consider are intrinsically safety-critical because of the interaction between the embedded, computerized controller, and a physical process having its own dynamics. Such systems are known under various names, notably *cyberphysical systems* and *embedded control systems*. What is important is to design and to analyze the safe behavior of the whole system, which introduces an inherent complexity. This is even more crucial in the case of systems whose malfunction can have catastrophic consequences, for example in transport systems (avionics, railways), production, medical, or energy production systems.

Therefore, there is a need for methods and tools for the design of safe systems. The definition of adequate mathematical models of the behavior of the systems allows the definition of formal calculi. They in turn form a basis for the construction of algorithms for the analysis, but also for the transformation of specifications towards an implementation. They can then be implemented in software environments made available to the users. A necessary complement is the setting-up of software engineering, programming, modeling, and validation methodologies. The motivation of these problems is at the origin of significant research activity, internationally and, in particular, in the European IST network of excellence ARTISTDESIGN (Advanced Real-Time Systems).

[1] http://www.artist-embedded.org
[2] http://www.combest.eu/home
[3] http://www.cesarproject.eu
[4] http://pop-art.inrialpes.fr/~girault/Projets/Afmes

3.1.2. Models, Methods and Techniques

The state of the art upon which we base our contributions is twofold.

From the point of view of discrete control, there is a set of theoretical results and tools, in particular in the synchronous approach, often founded on finite or infinite labeled transition systems [41], [46]. During the past years, methodologies for the formal verification [87], [48], control synthesis [89] and compilation, as well as extensions to timed and hybrid systems [85], [42] have been developed. Asynchronous models consider the interleaving of events or messages, and are often applied in the field of telecommunications, in particular for the study of protocols.

From the point of view of verification, we use the methods and tools of symbolic model-checking and of abstract interpretation. From symbolic model-checking, we use BDD techniques [44] for manipulating Boolean functions and sets, and their MTBDD extension for more general functions. Abstract interpretation [51] is used to formalize complex static analysis, in particular when one wants to analyze the possible values of variables and pointers of a program. Abstract interpretation is a theory of approximate solving of fix-point equations applied to program analysis. Most program analysis problems, among which reachability analysis, come down to solving a fix-point equation on the state space of the program. The exact computation of such an equation is generally not possible for undecidability (or complexity) reasons. The fundamental principles of abstract interpretation are: (i) to substitute to the state-space of the program a simpler domain and to transpose the equation accordingly (static approximation); and (ii) to use extrapolation (widening) to force the convergence of the iterative computation of the fix-point in a finite number of steps (dynamic approximation). Examples of static analyses based on abstract interpretation are linear relation analysis [52] and shape analysis [47].

The synchronous approach [5] [73], [74] to reactive systems design gave birth to complete programming environments, with languages like ARGOS, LUSTRE [6], ESTEREL [7], SIGNAL/ POLYCHRONY [8], LUCID SYNCHRONE, SYNDEX [9], or Mode Automata. This approach is characterized by the fact that it considers periodically sampled systems whose global steps can, by synchronous composition, encompass a set of events (known as simultaneous) on the resulting transition. Generally speaking, formal methods are often used for analysis and verification; they are much less often integrated into the compilation or generation of executives (in the sense of executables of tasks combined with the host real-time operating system). They are notoriously difficult to use by end-users, who are usually experts in the application domain, not in formal techniques. This is why encapsulating formal techniques into an automated framework can dramatically improve their diffusion, acceptance, and hence impact. Our work is precisely oriented towards this direction.

3.2. Issues in Design Automation for Complex Systems

3.2.1. Hard Problems

The design of safe real-time control systems is difficult due to various issues, among them their complexity in terms of the number of interacting components, their parallelism, the difference of the considered time scales (continuous or discrete), and the distance between the various theoretical concepts and results that allow the study of different aspects of their behaviors, and the design of controllers.

A currently very active research direction focuses on the models and techniques that allow the automatic use of formal methods. In the field of verification, this concerns in particular the technique of model checking. The verification takes place after the design phase, and requires, in case of problematic diagnostics, expensive backtracks on the specification. We want to provide a more constructive use of formal models, employing them to derive correct executives by formal computation and synthesis, integrated in a compilation process. We therefore use models throughout the design flow from specification to implementation, in particular by automatic generation of embeddable executives.

[5] http://www.synalp.org
[6] http://www-verimag.imag.fr/SYNCHRONE
[7] http://www.inria.fr/equipes/aoste
[8] http://www.irisa.fr/espresso/Polychrony
[9] http://www-rocq.inria.fr/syndex

3.2.2. *Applicative Needs*

Applicative needs initially come from the fields of safety-critical systems (avionics, energy) and complex systems (telecommunications), embedded in an environment with which they strongly interact (comprising aspects of computer science and control theory). Fields with less criticality, or which support variable degrees of quality of service, such as in the multi-media domain, can also take advantage of methodologies that improve the quality and reliability of software, and reduce the costs of test and correction in the design.

Industrial acceptance, the dissemination, and the deployment of the formal techniques inevitably depend on the usability of such techniques by specialists in the application domain — and not in formal techniques themselves — and also on the integration in the whole design process, which concerns very different problems and techniques. Application domains where the actors are ready to employ specialists in formal methods or advanced control theory are still uncommon. Even then, design methods based on the systematic application of these theoretical results are not ripe. In fields like industrial control, where the use of PLC (Programmable Logic Controller [37]) is dominant, this question can be decisive.

Essential elements in this direction are the proposal of realistic formal models, validated by experiments, of the usual entities in control theory, and functionalities (*i.e.*, algorithms) that correspond indeed to services useful for the designer. Take, for example, the compilation and optimization taking into account the platforms of execution, the possible failures, or the interactions between the defined automatic control and its implementation. A notable example for the existence of an industrial need is the activity of the ATHYS company (now belonging to DASSAULT SYSTEMES) concerning the development of a specialized programming environment, CELLCONTROL, which integrates synchronous tools for compilation and verification, tailored to the application domain. In these areas, there are functionalities that commercial tools do not have yet, and to which our results contribute.

3.2.3. *Our Approach*

We are proposing effective trade-offs between, on the one hand, expressiveness and formal power, and on the other hand, usability and automation. We focus on the area of specification and construction of correct real-time executives for discrete and continuous control, while keeping an interest in tackling major open problems, relating to the deployment of formal techniques in computer science, especially at the border with control theory. Regarding the applications, we propose new automated functionalities, to be provided to the users in integrated design and programming environments.

3.3. Main Research Directions

The overall consistency of our approach comes from the fact that the main research directions address, under different aspects, the specification and generation of safe real-time control executives based on *formal models*.

We explore this field by linking, on the one hand, the techniques we use, with on the other hand, the functionalities we want to offer. We are interested in questions related to:

Component-Based Design. We investigate two main directions: (i) compositional analysis and design techniques; (ii) adapter synthesis and converter verification.

Programming for embedded systems. Programming for embedded real-time systems is considered within POP ART along three axes: (i) synchronous programming languages, (ii) aspect-oriented programming, (iii) static analysis (type systems, abstract interpretation, ...).

Dependable embedded systems. Here we address the following research axes: (i) static multiprocessor scheduling for fault-tolerance, (ii) multi-criteria scheduling for reliability, (iii) automatic program transformations, (iv) formal methods for fault-tolerant real-time systems.

The creation of easily usable models aims at giving the user the role rather of a pilot than of a mechanics *i.e.*, to offer her/him pre-defined functionalities which respond to concrete demands, for example in the generation of fault tolerant or distributed executives, by the intermediary use of dedicated environments and languages.

The proposal of validated models with respect to their faithful representation of the application domain is done through case studies in collaboration with our partners, where the typical multidisciplinarity of questions across control theory and computer science is exploited.

3.3.1. Component-Based Design

Component-based construction techniques are crucial to overcome the complexity of embedded systems design. However, two major obstacles need to be addressed: the heterogeneous nature of the models, and the lack of results to guarantee correction of the composed system.

The heterogeneity of embedded systems comes from the need to integrate components using different models of computation, communication, and execution, at different levels of abstraction and different time scales. The BIP component framework [5] has been designed, in cooperation with VERIMAG, to support this heterogeneous nature of embedded systems.

Our work focuses on the underlying analysis and construction algorithms, in particular compositional techniques and approaches ensuring correctness by construction (adapter synthesis, strategy mapping). This work is motivated by the strong need for formal, heterogeneous component frameworks in embedded systems design.

3.3.2. Programming for Embedded Systems

Programming for embedded real-time systems is considered along three directions: (i) synchronous programming languages to implement real-time systems; (ii) aspect-oriented programming to specify non-functional properties separately from the base program; (iii) abstract interpretation to ensure safety properties of programs at compile time. We advocate the need for well defined programming languages to design embedded real-time systems with correct-by-construction guarantees, such as bounded time and bounded memory execution. Our original contribution resides in programming languages inheriting features from both synchronous languages and functional languages. We contribute to the compiler of the HEPTAGON language (whose main inventor is Marc Pouzet, ENS Uml, PARKAS team), the key features of which are: data-flow formal synchronous semantics, strong typing, modular compilation. In particular, we are working on type systems for the clock calculus and the spatial modular distribution.

The goal of Aspect-Oriented Programming (AOP) is to isolate aspects (such as security, synchronization, or error handling) that cross-cut the program basic functionality and whose implementation usually yields tangled code. In AOP, such aspects are specified *separately* and integrated into the program by an automatic transformation process called *weaving*. Although this paradigm has great practical potential, it still lacks formalization, and undisciplined uses make reasoning on programs very difficult. Our work on AOP addresses these issues by studying foundational issues of AOP (semantics, analysis, verification) and by considering domain-specific aspects (availability or fault tolerance aspects) as formal properties.

Finally, the aim of the verification activity in POP ART is to check safety properties on programs, with emphasis on the analysis of the values of data variables (numerical variables, memory heap), mainly in the context of embedded and control-command systems that exibit concurrency features. The applications are not only the proof of functional properties on programs, but also test selection and generation, program transformation, controller synthesis, and fault-tolerance. Our approach is based on abstract interpretation, which consists in inferring properties of the program by solving semantic equations on abstract domains. Much effort is spent on implementing developed techniques in tools for experimentation and diffusion.

3.3.3. Dependable Embedded Systems

Embedded systems must often satisfy safety critical constraints. We address this issue by providing methods and algorithms to design embedded real-time systems with guarantees on their fault-tolerance and/or reliability level.

A first research direction concerns static multiprocessor scheduling of an application specification on a distributed target architecture. We increase the fault-tolerance level of the system by replicating the computations and the communications, and we schedule the redundant computations according to the faults to be tolerated. We also optimize the schedule *w.r.t.* several criteria, including the schedule length, the reliability, and the power consumption.

A second research direction concerns the fault-tolerance management, by reconfigurating the system (for instance by migrating the tasks that were running on a processor upon the failure of this processor) following objectives of fault-tolerance, consistent execution, functionality fulfillment, boundedness and optimality of response time. We base such formal methods on discrete controller synthesis.

A third research direction concerns AOP to weave fault-tolerance aspects in programs and electronic circuits (seen as synthesizable HDL programs) as mentioned in the previous section.

4. Application Domains

4.1. Industrial Applications

Our applications are in the embedded system area, typically: robotics, automotive, telecommunications, systems on chip (SoC). In some areas, safety is critical, and motivates the investment in formal methods and techniques for design. But even in less critical contexts, like telecommunications and multimedia, these techniques can be beneficial in improving the efficiency and the quality of designs, as well as the cost of the programmation and the validation processes.

Industrial acceptance of formal techniques, as well as their deployment, goes necessarily through their usability by specialists of the application domain, rather than of the formal techniques themselves. Hence our orientation towards the proposal of domain-specific (but generic) realistic models, validated through experience (*e.g.*, control tasks systems), based on formal techniques with a high degree of automation (*e.g.*, synchronous models), and tailored for concrete functionalities (*e.g.*, code generation).

4.2. Industrial Design Tools

The commercially available design tools (such as UML with real-time extensions, MATLAB/ SIMULINK/ dSPACE [10]) and execution platforms (OS such as VXWORKS, QNX, real-time versions of LINUX ...) starts now to provide besides their core functionalities design or verification methods. Some of them, founded on models of reactive systems, come close to tools with a formal basis, such as for example STATEMATE by iLOGIX.

Regarding the synchronous approach, commercial tools are available: SCADE [11] (based on LUSTRE), CONTROLBUILD and RT-BUILDER (based on SIGNAL) from GEENSYS [12] (part of DASSAULT SYSTEMES), specialized environments like CELLCONTROL for industrial automatism (by the INRIA spin-off ATHYS– now part of DASSAULT SYSTEMES). One can observe that behind the variety of actors, there is a real consistency of the synchronous technology, which makes sure that the results of our work related to the synchronous approach are not restricted to some language due to compatibility issues.

4.3. Current Industrial Cooperations

Regarding applications and case studies with industrial end-users of our techniques, we cooperate with STMicroelectronics on two topics: (i) compositional analysis and abstract interpretation for the TLM-based System-on-Chip design flow, and (ii) dynamic data-flow models of computation for streaming applications.

[10]http://www.dspaceinc.com
[11]http://www.esterel-technologies.com
[12]http://www.geensoft.com

5. Software

5.1. NBac

Participant: Bertrand Jeannet.

NBAC (Numerical and Boolean Automaton Checker) [13] is a verification/slicing tool for reactive systems containing combination of Boolean and numerical variables, and continuously interacting with an external environment. NBAC can also handle the same class of hybrid systems as the HyTech tool [76]. It aims at handling efficiently systems combining a non-trivial numerical behaviour with a complex logical (Boolean) behaviour.

NBAC is connected to two input languages: the synchronous dataflow language LUSTRE, and a symbolic automaton-based language, AUTOC/AUTO, where a system is defined by a set of symbolic hybrid automata communicating via valued channels. It can perform reachability analysis, co-reachability analysis, and combination of the above analyses. The result of an analysis is either a verdict to a verification problem, or a set of states together with a necessary condition to stay in this set during an execution. NBAC is founded on the theory of abstract interpretation.

It has been used for verification and debugging of LUSTRE programs [79] [61]. It is connected to the LUSTRE toolset [14]. It has also been used for controller synthesis of infinite-state systems. The fact that the analyses are approximated results simply in the obtention of a possibly non-optimal controller. In the context of conformance testing of reactive systems, it is used by the test generator STG [15] [49] [80] for selecting test cases.

5.2. Prometheus

Participant: Gregor Goessler.

The BIP component model (Behavior, Interaction model, Priority) [72][5] has been designed to support the construction of heterogeneous embedded systems involving different models of computation, communication, and execution, at different levels of abstraction. By separating the notions of behavior, interaction model, and execution model, it enables both heterogeneous modeling, and separation of concerns.

The verification and design tool Prometheus [71] implements the BIP component framework. Prometheus is regularly updated to implement new developments in the framework and the analysis algorithms. It has allowed us to carry out several complex case studies from the system-on-chip and bioinformatics domains [11].

5.3. Implementations of Synchronous Programs

Participant: Alain Girault.

5.3.1. Fault Tolerance

We have been cooperating for several years with the INRIA team AOSTE (INRIA Sophia-Antipolis and Rocquencourt) on the topic of fault tolerance and reliability of safety critical embedded systems. In particular, we have implemented several new heuristics for fault tolerance and reliability within their software SYNDEX [16]. Our first scheduling heuristic produces static multiprocessor schedules tolerant to a specified number of processor and communication link failures [64]. The basic principles upon which we rely to make the schedules fault tolerant is, on the one hand, the active replication of the operations [65], and on the other hand, the active replication of communications for point-to-point communication links, or their passive replication coupled with data fragmentation for multi-point communication media (*i.e.*, buses) [66]. Our second scheduling heuristic is multi-criteria: it produces a static schedule multiprocessor schedule such that the reliability is maximized, the power consumption is minimized, and the execution time is minimized [3][17]. Our results on fault tolerance are summarized in a web page [17].

[13] http://pop-art.inrialpes.fr/people/bjeannet/nbac/
[14] http://www-verimag.imag.fr/Lustre-V6.html
[15] http://www.irisa.fr/prive/ployette/stg-doc/stg-web.html
[16] http://www-rocq.inria.fr/syndex

5.4. Apron and BddApron Libraries

Participant: Bertrand Jeannet.

5.4.1. Principles

The APRON library [18] is dedicated to the static analysis of the numerical variables of a program by abstract interpretation [51]. Many abstract domains have been designed and implemented for analysing the possible values of numerical variables during the execution of a program (see Figure 1). However, their API diverge largely (datatypes, signatures, ...), and this does not ease their diffusion and experimental comparison w.r.t. efficiency and precision aspects.

The APRON library aims to provide:

- a uniform API for existing numerical abstract domains;
- a higher-level interface to the client tools, by factorizing functionalities that are largely independent of abstract domains.

From an abstract domain designer point of view, the benefits of the APRON library are:

- the ability to focus on core, low-level functionalities;
- the help of generic services adding higher-level services for free.

For the client static analysis community, the benefits are a unified, higher-level interface, which allows experimenting, comparing, and combining abstract domains.

In 2011, the Taylor1plus domain [62], which is the underlying abstract domain of the tool FLUCTUAT [58] has been improved. Glue code has also been added to enable the connection of an abstract domain implemented in OCaml to the APRON infrastructure written in C (this requires callbacks from C to OCaml that are safe w.r.t. garbage collection). This will enable the integration in APRON of the MaxPlus polyhedra library written by X. Allamigeon [38] in the context of the ANR ASOPT project.

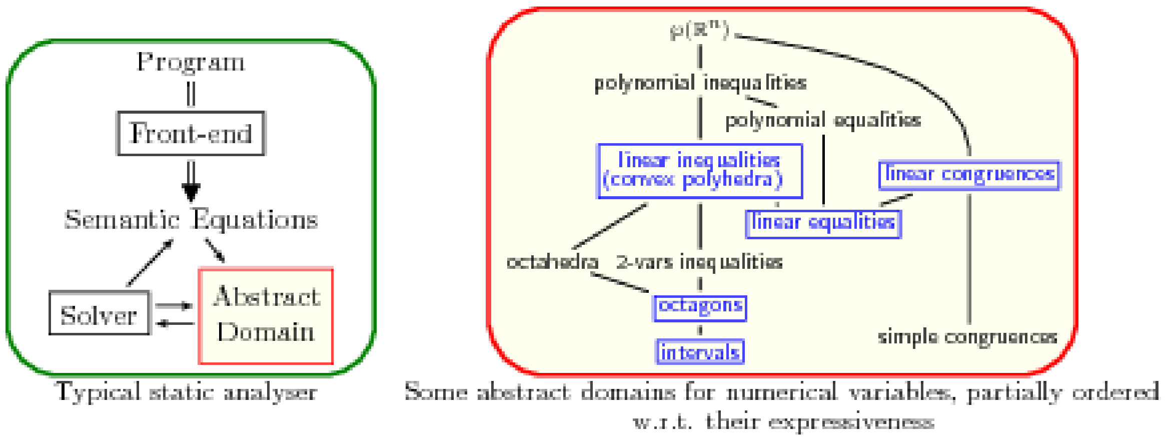

Figure 1. Typical static analyser and examples of abstract domains

[17] http://pop-art.inrialpes.fr/~girault/Projets/FT
[18] http://apron.cri.ensmp.fr/library/

The BDDAPRON library [19] aims at a similar goal, by adding finite-types variables and expressions to the concrete semantics of APRON domains. It is built upon the APRON library and provides abstract domains for the combination of finite-type variables (Booleans, enumerated types, bitvectors) and numerical variables (integers, rationals, floating-point numbers). It first allows to manipulate expressions that freely mix, using BDDs and MTBDDs, finite-type and numerical APRON expressions and conditions. It then provides abstract domains that combines BDDs and APRON abstract values for representing invariants holding on both finite-type variables and numerical variables.

5.4.2. Implementation and Distribution

The APRON library (Fig. 2) is written in ANSI C, with an object-oriented and thread-safe design. Both multi-precision and floating-point numbers are supported. A wrapper for the OCAML language is available, and a C++ wrapper is on the way. It has been distributed since June 2006 under the LGPL license and available at http://apron.cri.ensmp.fr. Its development has still progressed much since. There are already many external users (ProVal/Démons, LRI Orsay, France — CEA-LIST, Saclay, France — Analysis of Computer Systems Group, New-York University, USA — Sierum software analysis platform, Kansas State University, USA — NEC Labs, Princeton, USA — EADS CCR, Paris, France — IRIT, Toulouse, France) and is currently packaged as a REDHAT and DEBIAN package.

The BDDAPRON library is written in OCAML, using polymorphism features of OCAML to make it generic. It is also thread-safe. It provides two different implementations of the same domain, each one presenting pros and cons depending on the application. It is currently used by the CONCURINTERPROC interprocedural and concurrent program analyzer.

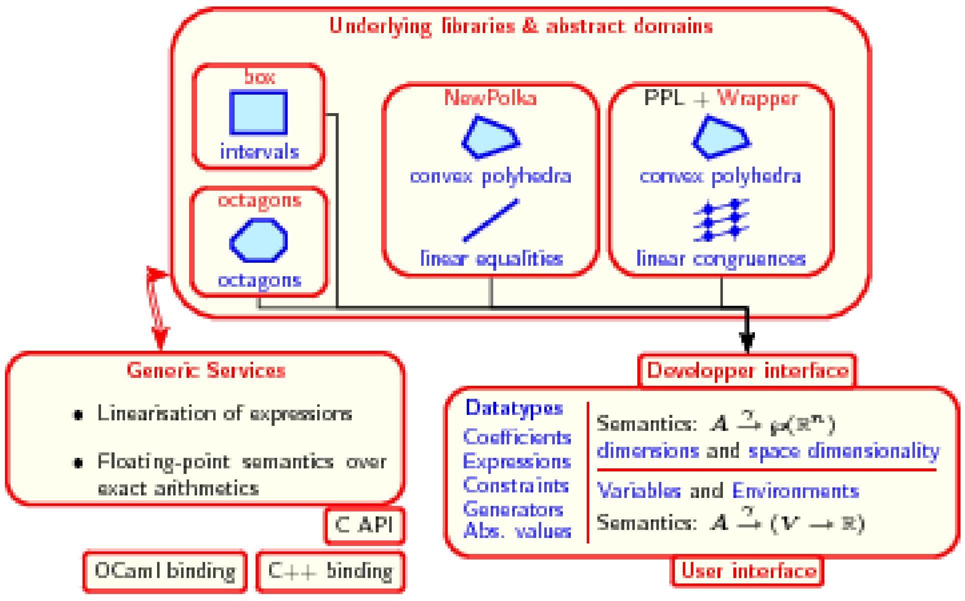
Underlying libraries & abstract domains
box
intervals
octagons
octagons
NewPolka
convex polyhedra
linear equalities
PPL + Wrapper
convex polyhedra
linear congruences
Generic Services
• Linearisation of expressions
• Floating-point semantics over exact arithmetics
C API
OCaml binding
C++ binding
Developper interface
Datatypes
Coefficients
Expressions
Constraints
Generators
Abs. values
Semantics: $A \xrightarrow{\gamma} \wp(\mathbb{R}^n)$
dimensions and space dimensionality
Variables and Environments
Semantics: $A \xrightarrow{\gamma} (\mathcal{V} \to \mathbb{R})$
User interface

Figure 2. Organisation of the APRON *library*

5.5. Prototypes

5.5.1. Logical Causality

Participants: Lacramioara Astefanoaei, Gregor Goessler [contact person].

[19] http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/index.html

We have developed LOCA, a new prototype tool written in Scala that implements the analysis of logical causality described in 6.6.2. LOCA currently supports causality analysis in BIP. The core analysis engine is implemented as an abstract class, such that support for other models of computation (MOC) can be added by instantiating the class with the basic operations of the MOC.

5.5.2. Automatic Controller Generation

Participants: Emil Dumitrescu, Alain Girault [contact person].

We have developed a software tool chain to allow the specification of models, the controller synthesis, and the execution or simulation of the results. It is based on existing synchronous tools, and thus consists primarily in the use and integration of SIGALI [20] and Mode Automata [21]. It is the result of a collaboration with Eric Rutten from the SARDES team.

Useful component templates and relevant properties can be materialized, on one hand by libraries of task models, and, on the other hand, by properties and synthesis objectives.

5.5.3. Rapture

Participant: Bertrand Jeannet.

RAPTURE [22] [78] [53] is a verification tool that was developed jointly by BRICS (Denmark) and INRIA in years 2000–2002. The tool is designed to verify reachability properties on Markov Decision Processes (MDP), also known as Probabilistic Transition Systems. This model can be viewed both as an extension to classical (finite-state) transition systems extended with probability distributions on successor states, or as an extension of Markov Chains with non-determinism. We have developed a simple automata language that allows the designer to describe a set of processes communicating over a set of channels *à la* CSP. Processes can also manipulate local and global variables of finite type. Probabilistic reachability properties are specified by defining two sets of initial and final states together with a probability bound. The originality of the tool is to provide two reduction techniques that limit the state space explosion problem: automatic abstraction and refinement algorithms, and the so-called essential states reduction.

5.5.4. The Interproc family of static analyzers

Participants: Bertrand Jeannet [contact person], Pascal Sotin.

These analyzers and libraries are of general use for people working in the static analysis and abstract interpretation community, and serve as an experimental platform for the ANR project ASOPT (see §8.1.2).

a generic fix-point engine written in OCAML. It allows the user to solve systems of fix-point equations on a lattice, using a parameterized strategy for the iteration order and the application of widening. It also implements recent techniques for improving the precision of analysis by alternating post-fixpoint computation with widening and descending iterations in a sound way [70].

a simple interprocedural static analyzer that infers properties on the numerical variables of programs in a toy language. It is aimed at demonstrating the use of the previous library and the above-described APRON library, and more generally at disseminating the knowledge in abstract interpretation. It is also deployed through a web-interface [25]. It is used as the experimental platform of the ASOPT ANR project.

FIXPOINT[23] INTERPROC[24] CONCURINTERPROC extends Interproc with concurrency, for the analysis of multi-threaded programs interacting via shared global variables. It is also deployed through a web-interface [26].

[20] http://www.irisa.fr/vertecs/Logiciels/sigali.html
[21] http://www-verimag.imag.fr
[22] http://pop-art.inrialpes.fr/people/bjeannet/rapture/rapture.html
[25] http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
[26] http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi

PINTERPROC extends Interproc with pointers to local variables. It is also deployed through a web-interface [27].

5.5.5. *Heptagon/BZR*

Participant: Gwenaël Delaval.

HEPTAGON is a dataflow synchronous language, inspired from LUCID SYNCHRONE [28]. Its compiler is meant to be simple and modular, allowing this language to be a good support for the prototyping of compilation methods of synchronous languages. It is developped within the SYNCHRONICS INRIA large-scale action.

HEPTAGON has been used to built BZR [29], which is an extension of the former with contracts constructs. These contracts allow to express dynamic temporal properties on the inputs and outputs of HEPTAGON node. These properties are then enforced, within the compilation of a BZR program, by discrete controller synthesis, using the SIGALI tool [30]. The synthesized controller is itself generated in HEPTAGON, allowing its analysis and compilation towards different target languages (C, JAVA, VHDL).

6. New Results

6.1. Dependable Distributed Real-time Embedded Systems

Participants: Pascal Fradet, Alain Girault [contact person], Emil Dumitrescu.

6.1.1. *The TSH multi-criteria scheduling heuristic*

For autonomous critical real-time embedded systems (e.g., satellite), guaranteeing a very high level of reliability is as important as keeping the power consumption as low as possible. We have designed an off-line scheduling heuristics which, from a given software application graph and a given multiprocessor architecture (homogeneous and fully connected), produces a static multiprocessor schedule that optimizes three criteria: its *length* (crucial for real-time systems), its *reliability* (crucial for dependable systems), and its *power consumption* (crucial for autonomous systems). Our tricriteria scheduling heuristics, **TSH**, uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling* to lower the power consumption [17]. By running TSH on a single problem instance, we are able to provide the Pareto front for this instance in 3D, therefore exposing the user to several tradeoffs between the power consumption, the reliability and the execution time. Thanks to extensive simulation results, we have shown how TSH behaves in practice. Firstly, we have compared TSH versus an optimal Mixed Linear Integer Program on small instances; the experimental results show that TSH behaves very well compared to the the ILP. Secondly, we have compared TSH versus the ECS heuristic (Energy-Conscious Scheduling [84]); the experimental results show that TSH performs systematically better than ECS.

This is a joint work with Ismail Assayad (U. Casablanca, Morocco) and Hamoudi Kalla (U. Batna, Algeria), who both visit the team regularly.

6.1.2. *Automating the Addition of Fault Tolerance with Discrete Controller Synthesis*

In collaboration with Emil Dumitrescu (INSA Lyon), Hervé Marchand (VERTECS team from Rennes), and Eric Rutten (SARDES team from Grenoble), we have defined a complete framework for the *automatic* design of fault tolerant embedded systems, based on discrete controller synthesis (DCS) [88]. Its interest lies in the ability to obtain automatically systems satisfying by construction formal properties specified *a priori*. Our aim is to demonstrate the feasibility of this approach for fault tolerance. We start with a fault intolerant program, modeled as the synchronous parallel composition of finite labeled transition systems. We specify formally a fault hypothesis, state fault tolerance requirements and use DCS to obtain automatically a program having

[27] http://pop-art.inrialpes.fr/interproc/pinterprocweb.cgi
[28] http://www.di.ens.fr/~pouzet/lucid-synchrone
[29] http://bzr.inria.fr
[30] http://www.irisa.fr/vertecs/Logiciels/sigali.html

the same behavior as the initial fault intolerant one in the absence of faults, and satisfying the fault tolerance requirements under the fault hypothesis. Our original contribution resides in the demonstration that DCS can be elegantly used to design fault tolerant systems, with guarantees on key properties of the obtained system, such as the fault tolerance level, the satisfaction of quantitative constraints, and so on. We have shown with numerous examples taken from case studies that our method can address different kinds of failures (crash, value, or Byzantine) affecting different kinds of hardware components (processors, communication links, actuators, or sensors). Besides, we have shown that our method also offers an optimality criterion very useful to synthesize fault tolerant systems compliant to the constraints of embedded systems, like power consumption or execution times. In summary, our framework for fault tolerance has the following advantages [67]:

- The **automation**, because DCS produces automatically a fault tolerant system from an initial fault intolerant one.
- The **separation of concerns**, because the fault intolerant system can be designed independently from the fault tolerance requirements.
- The **flexibility**, because, once the system is entirely modeled, it is easy to try several fault hypotheses, several environment models, several fault tolerance goals, several degraded modes, and so on.
- The **safety**, because, in case of positive result obtained by DCS, the specified fault tolerance properties are guaranteed by construction on the controlled system.
- The **optimality** when optimal synthesis is used, modulo the potential numerical equalities (hence a non strict optimality). We consider weights cumulated along bounded-length paths. We have adapted our models in order to take into account the additive costs of, *e.g.*, execution time or power consumption, and adapting synthesis algorithms in order to support the association of costs with transitions, and the handling of these new cost functions in the optimal synthesis [59].

We therefore combine, on the one hand, guarantees on the safety of the execution by tolerating faults, and on the other hand, guarantees on the worst cumulated consumption of the resulting dynamically reconfiguring fault tolerant system. Recently, we have incorporated multi-criteria optimization results in this work, to take into account *several* weight functions: for instance the execution costs of several tasks, the execution of which must be controlled thanks to DCS. We therefore propose several synthesis algorithms, to aggregate the costs into a single cost function, to hierarchize the costs (*e.g.*, to reflect the priorities of the tasks), or to compute the Pareto front of non-dominated solutions.

6.2. Controller Synthesis for the Safe Design of Embedded Systems

Participants: Gwenaël Delaval [contact person], Gregor Goessler, Sebti Mouelhi.

6.2.1. Synthesis of Switching Controllers using Approximately Bisimilar Multiscale Abstractions

The use of discrete abstractions for continuous dynamics has become standard in hybrid systems design (see e.g. [92] and the references therein). The main advantage of this approach is that it offers the possibility to leverage controller synthesis techniques developed in the areas of supervisory control of discrete-event systems [88]. The first attempts to compute discrete abstractions for hybrid systems were based on traditional systems behavioral relationships such as simulation or bisimulation, initially proposed for discrete systems most notably in the area of formal methods. These notions require inclusion or equivalence of observed behaviors which is often too restrictive when dealing with systems observed over metric spaces. For such systems, a more natural abstraction requirement is to ask for closeness of observed behaviors. This leads to the notions of approximate simulation and bisimulation introduced in [63].

These notions enabled the computation of approximately equivalent discrete abstractions for several classes of dynamical systems, including nonlinear control systems with or without disturbances, and switched systems. These approaches are based on sampling of time and space where the sampling parameters must satisfy some relation in order to obtain abstractions of a prescribed precision. In particular, the smaller the time sampling parameter, the finer the lattice used for approximating the state-space; this may result in abstractions with a very large number of states when the sampling period is small. However, there are a number of applications where sampling has to be fast; though this is generally necessary only on a small part of the state-space.

In [22] we have presented a novel class of multiscale discrete abstractions for incrementally stable switched systems that allows us to deal with fast switching while keeping the number of states in the abstraction at a reasonable level. We assume that the controller of the switched system has to decide the control input and the time period during which it will be applied before the controller executes again. In this context, it is natural to consider abstractions where transitions have various durations. For transitions of longer duration, it is sufficient to consider abstract states on a coarse lattice. For transitions of shorter duration, it becomes necessary to use finer lattices. These finer lattices are effectively used only on a restricted area of the state-space where the fast switching occurs.

These abstractions allow us to use multiscale iterative approaches for controller synthesis as follows. An initial controller is synthesized based on the dynamics of the abstraction at the coarsest scale where only transitions of longer duration are enabled. An analysis of this initial controller allows us to identify regions of the state-space where transitions of shorter duration may be useful (e.g. to improve the performance of the controller). Then, the controller is refined by enabling transitions of shorter duration in the identified regions. The last two steps can be repeated until we are satisfied with the obtained controller.

In [21] we propose a technique for the synthesis of *safety* controllers for switched systems using multi-scale abstractions. We present a synthesis algorithm that exploits the specificities of multi-scale abstractions. The finest scales of the abstraction are effectively explored only when fast switching is needed, that is when the system approaches the unsafe set. We provide experimental results that show drastic improvements of the complexity of controller synthesis using multi-scale abstractions instead of uniform abstractions.

6.2.2. Modular Discrete Controller Synthesis

Discrete controller synthesis (DCS) [88] allows to design programs in a mixed imperative/declarative way. From a program with some freedom degrees left by the programmer (e.g., free controllable variables), and a temporal property to enforce which is not *a priori* verified by the initial program, DCS tools compute off-line automatically a *controller* which will constrain the program (by e.g., giving values to controllable variables) such that, whatever the values of inputs from the environment, the *controlled program* satisfies the temporal property.

Our motivation w.r.t. DCS concerns its modular application, improving the scalability of the technique by using contract enforcement and abstraction of components. Moreover, our aim is to integrate DCS into a compilation chain, and thereby improve its usability by programmers, not experts in discrete control. This work has been implemented into the HEPTAGON/BZR language and compiler [57]. This work is done in collaboration with Hervé Marchand (VERTECS team from Rennes) and Éric Rutten (SARDES team from Grenoble).

The implemented tool allows the generation of the synthesized controller under the form of an HEPTAGON node, which can in turn be analyzed and compiled, together with the HEPTAGON source from which it has been generated. This full integration allows this method to aim different target languages (currently C, JAVA or VHDL), and its integrated use in different contexts.

A formal semantics of BZR has been defined, taking into account its underlying nondeterminism related to the presence of controllable variables.

This language has been used in different contexts. In [15], BZR is used for the generation of discrete handlers of real-time continuous control tasks, in the framework of the ORCCAD [31] tool. BZR has also been used in a case-study of a Fractal designed HTTP server [19]. The purpose of the synthesized controller is to control the

[31] Open Robot Controller Computer-Aided Design

automatic reconfigurations of the system (e.g., start of new components, migrations of some components from one computing element to another), in order to preserve some properties (either functional, e.g., exclusivity of activities of two components, or non-functional, e.g., bounded overall load of the system).

6.3. Automatic Distribution of Synchronous Programs

Participants: Gwenaël Delaval [contact person], Alain Girault, Gregor Goessler, Xavier Nicollin, Gideon Smeding.

6.3.1. Modular Distribution

Synchronous programming languages describe functionally centralized systems, where every value, input, output, or function is always directly available for every operation. However, most embedded systems are nowadays composed of several computing resources. The aim of this work is to provide a language-oriented solution to describe *functionally distributed reactive systems*. This research is conducted within the INRIA large scale action SYNCHRONICS and is a joint work with Marc Pouzet (ENS, PARKAS team from Rocquencourt) and Xavier Nicollin (Grenoble INP, VERIMAG lab).

We are working on type systems to formalize, in a uniform way, both the clock calculus and the location calculus of a synchronous data-flow programming language (the HEPTAGON language, inspired from LUCID SYNCHRONE [45]). On one hand, the clock calculus infers the clock of each variable in the program and checks the clock consistency: e.g., a time-homogeneous function, like `+`, should be applied to variables with identical clocks. On the other hand, the location calculus infers the spatial distribution of computations and checks the spatial consistency: e.g., a centralized operator, like `+`, should be applied to variables located at the same location. Compared to the PhD of Gwenaël Delaval [55], [56], the goal is to achieve *modular* distribution. By modular, we mean that we want to compile each function of the program into a single function capable of running on any computing location. We make use of our uniform type system to express the computing locations as first-class abstract types, exactly like clocks, which allows us to compile a typed variable (typed by both the clock and the location calculi) into `if ... then ... else ...` structures, whose conditions will be valuations of the clock and location variables.

We currently work on an example of software-defined radio. We have shown on this example how to use a modified clock calculus to describe the localisation of values as clocks, and the architecture as clocks (for the computing resources) and their relations (for communication links).

6.3.2. Distribution of Synchronous Programs under Real-Time Constraints

With the objective to distribute synchronous data-flow programs (e.g. LUSTRE) over GALS architectures, preserving only explicitly specified properties, we have developed a quantitative clock calculus to (1) describe timing properties of the architecture's clock domain, and (2) describe the properties of the synchronous program to be preserved. The clock calculus is inspired by the network calculus [83], with the difference that clocks are described only with respect to one-another, not with respect to real-time.

As a first result, we have applied our clock calculus to analyze the properties of periodic synchronous data-flow programs executed on a network of processors. Because our clock calculus is relational, it can model and preserve correlated variations of streams. In particular, the common case of a data-flow system that splits a stream for separate treatment, and joins them afterwards, this analysis yields more precise result than comparable methods.

We aim to extend the analysis to account for shared resources and synchronization protocols, so as to distribute synchronous programs preserving specified properties.

6.4. New Programming Languages for Embedded Systems

Participants: Alain Girault [contact person], Pascal Fradet, Petro Poplavko, Vagelis Bebelis, Bertrand Jeannet, Peter Schrammel.

6.4.1. The DSystemJ programming language

In collaboration with Avinash Malik (IBM Watson) and Zoran Salcic (University of Auckland), we have designed the SYSTEMJ programming language [9], which implements the Globally Asynchronous Locally Synchronous (GALS) Model of Computation (MoC) over JAVA. In a nutshell, SYSTEMJ uses the notion of *clock domains* (CD) to design portions of the system that must operate at unrelated clocks. CDs communicate with each other via asynchronous rendez-vous. Then, a CD consists of one or several *reactions*, which react synchronously in lock-step and communicate with each other via synchronous broadcast of signals. Finally, all the data computations are implemented in JAVA.

We have further extended SYSTEMJ to allow programmers to design *dynamic* GALS systems: this is the new language DSYSTEMJ [27], [12], aimed at dynamic distributed systems that use socket based communication protocols for communicating between components. DSYSTEMJ allows the creation and control at runtime of CDs, their mobility on a distributed execution platform, as well as the runtime reconfiguration of the system's functionality and topology. We have defined the formal semantics of DSYSTEMJ, based on the Dynamic GALS MoC: it offers very safe mechanisms for implementation of distributed systems, as well as potential for their formal verification. The runtime support is implemented in the SYSTEMJ language, which can as such be considered as a static subset of DSYSTEMJ.

This work has been done within the AFMES associated team with the Electric and Computer Engineering Department of the University of Auckland.

6.4.2. The PRET-C programming language for time-predictable systems

Typical safety critical embedded applications, ranging from complex aircraft flight controllers to embedded health devices require worst case guarantees on their timing behavior. The problem is that general-purpose processors, being highly speculative, are intrinsically non-deterministic, and thus are not ideally suited for implementing such systems: either the computed worst-case execution time is highly pessimistic, or heroic efforts are required to accurately model the caches, pipeline, and speculative execution [93]. For similar reasons, using an RTOS to guarantee the determinism of a program's behavior, along with temporal guarantees, is not feasible. The ability to analyze temporal bounds is dependent on the selected programming language, compiler tool chain, operating system, and the target hardware.

To alleviate these problems, we have defined a synchronous variant of C called PRET-C, together with Sidharta Andalam and Partha Roop (University of Auckland). PRET-C offers constructs for reactive inputs/outputs; it supports a notion of logical time, synchronous concurrency, and preemption [40]. We have also designed the ARPRET architecture for efficient and predictable execution of PRET-C. ARPRET inherits from the long lasting research effort on reactive processors conducted at the University of Auckland. Finally, all timing constraints are precisely verified using a Worst Case Reaction Time (WCRT) analyzer. While there has been a considerable body of work on the timing analysis of procedural programs [93], such analysis for synchronous programs has received less attention. Current state-of-the-art analyses for synchronous programs use integer linear programming (ILP) combined with path pruning techniques to achieve tight results. These approaches first convert a concurrent synchronous program into a sequential program. ILP constraints are then derived from this sequential program to compute the longest tick length. For PRET-C, we have proposed an alternative approach based on model checking [16]. Unlike conventional programs, synchronous programs are concurrent and state-space oriented, making them ideal for model checking based analysis. Our analysis of the abstracted state-space of the program is combined with expressive data-flow information, to facilitate effective path pruning. We have demonstrated through extensive experimentation that the proposed approach is both scalable and about 67% tighter compared to the existing approaches (namely Protothreads [60] and SC [94]).

This overall framework provides an ideal platform for designing and verifying precision timed real-time systems. It has been conducted within the AFMES associated team with the Electric and Computer Engineering Department of the University of Auckland, and is the topic of the PhD of Sidharta Andalam.

6.4.3. Analysis and Scheduling of Parametric Data-Flow Models

Recent data-flow programming environments support applications whose behavior is characterized by dynamic variations in resource requirements. The high expressive power of the underlying models (*e.g.*, Kahn Process Networks, the CAL actor language) makes it challenging to ensure predictable behavior. In particular, checking *liveness* (*i.e.*, no part of the system will deadlock) and *boundedness* (*i.e.*, the system can be executed in finite memory) is known to be hard or even undecidable for such models. This situation is troublesome for the design of high-quality embedded systems.

We have introduced the *schedulable parametric data-flow (SPDF)* model of computation (MoC) for dynamic streaming applications [23], [32], [36], [34], [35]. SPDF extends the standard data flow model by allowing rates to be parametric (*e.g.*, of the form $2xy$). SPDF was designed to be statically analyzable while retaining sufficient expressive power. We formulated sufficient and general static criteria for boundedness and liveness. In SPDF, parameters can be changed dynamically even within iterations. The safety of dynamic parameter changes can be checked and their implementation made explicit in the graph. These different analyses are made possible using well-defined static operations on symbolic expressions. The same holds for quasi-static scheduling which is the first step towards code generation for multi-core systems.

We are now considering other kinds of analyses for this new data-flow MoC. The objective of these analyses is to generate distributed schedules optimizing both the power consumption and the execution time of applications. The targeted hardware is P2012, a new embedded many-core platform designed by STMicroelectronics consisting of several clusters (9 in the current implementation) interconnected through a 2D mesh asynchronous NoC. Each cluster comprises 16 identical computing cores and is equipped with a hardware mechanism for DVFS (dynamic voltage and frequency scaling). As a first step, we have studied energy efficient scheduling of simple data-flow graphs for that platform [81]. The next step is to extend the approach to SPDF.

This line of research will be followed in the PhD thesis of Vagelis Bebelis which has just started. It will be conducted in collaboration with STMicroelectronics.

6.4.4. Translating Hybrid Data-Flow Languages to Hybrid Automata

Hybrid systems are used to model embedded computing systems interacting with their physical environment. There is a conceptual mismatch between high-level hybrid system languages like SIMULINK, which are used for simulation, and hybrid automata, the most suitable representation for safety verification. Indeed, in simulation languages the interaction between discrete and continuous execution steps is specified using the concept of zero-crossings, whereas hybrid automata exploit the notion of staying conditions.

In the context of the INRIA large scale action SYNCHRONICS (see §8.1.4), we studied how to translate the ZELUS hydrid data-flow language [43] developped in this project into logico-numerical hybrid automata by carefully pointing out this issue. We investigated various zero-crossing semantics, proposed a sound translation, and discussed to which extent the original semantics is preserved. This work has been accepted to the conference HSCC'2012 (Hybrid Systems: Computation and Control).

This work is part of the PhD thesis of Peter Schrammel.

6.5. Static Analysis and Abstract Interpretation

Participants: Alain Girault, Bertrand Jeannet [contact person], Lies Lakhdar-Chaouch, Peter Schrammel, Pascal Sotin.

6.5.1. Numerical and logico-numerical abstract acceleration

Acceleration methods are used for computing precisely the effects of loops in the reachability analysis of counter machine models. Applying these methods to synchronous data-flow programs with Boolean and numerical variables, e.g., LUSTRE programs, firstly requires the enumeration of the Boolean states in order to obtain a control graph with numerical variables only. Secondly, acceleration methods have to deal with the non-determinism introduced by numerical input variables.

Concerning the latter problem, we pushed further the work presented in [90] that extended the concept of abstract acceleration of Gonnord et al. [69], [68] to numerical input variables, and we wrote a journal version [13]. The original contributions of [13] compared to [91] is abstract *backward* acceleration (for backward analysis) and a detailed comparison of the abstract acceleration approach with the derivative closure approach of [39], which is related to methods based on transitive closures of relations.

We then worked more on the first point, which is to apply acceleration techniques to data-flow programs without resorting to an exhaustive enumeration of Boolean states. To this end, we introduced (1) *logico-numerical abstract acceleration methods* for CFGs with Boolean and numerical variables and (2) partitioning techniques that make logical-numerical abstract acceleration effective. Experimental results showed that incorporating these methods in a verification tool based on abstract interpretation provides not only significant advantage in terms of accuracy, but also a gain in performance in comparison to standard techniques. This work was published in [28].

This line of work is part of the PhD thesis of Peter Schrammel.

6.5.2. Improving dynamic approximations in static analysis

Abstract interpretation [51] formalizes two kind of approximations that can be done in the static analysis of programs:

- Static approximations, defined by the choice of an abstract domain of abstract properties (for instance, intervals or convex polyhedra that approximates set of points in numerical spaces), and the definition of sound approximations in this domain of concrete operations (variable assignments, tests, ...). These abstract properties and operations are substitutes to the concrete properties and operations defined by the semantics of the analyzed program. This stage results into a abstract fixpoint equation $Y = G(Y), Y \in A$, where A is the abstract domain. The best (least) solution of this equation can be obtained by *Kleene iteration*, which consists in computing the sequence $Y_0 = \bot_A, Y_{n+1} = G(Y_n)$, where $\bot_A$ is the least element of the lattice A.
- Dynamic approximations, that makes the Kleene iteration sequence converge in finite time by applying an extrapolation operator called *widening* and denoted with ∇. This results in a sequence $Z_0 = \bot_A, Z_{n+1} = Z_n \nabla G(Z_n)$ that converges to a post-fixpoint $Z_\infty \sqsupseteq G(Z_\infty)$. For instance, for many numerical abstract domains (like octagons [86] or convex polyhedra [75]) the "standard" widening $\nabla : A \times A \to A$ consists in keeping in the result $R = P \nabla Q$ the numerical constraints of P that are still satisfied by Q.

The problem addressed here is that the extrapolation performed by widening often loses crucial information for the analysis goal.

6.5.2.1. *Widening with thresholds.*

A classical technique for improving the precision is "widening with thresholds", which bounds the extrapolation. The idea is to parameterize ∇ with a finite set $\mathcal{C}$ of *threshold constraints*, and to keep in the result $R = P \nabla_{\mathcal{C}} Q$ those constraints $c \in \mathcal{C}$ that are still satisfied by Q: $P \nabla_{\mathcal{C}} Q = (P \nabla Q) \sqcap \{c \in \mathcal{C} \mid Q \models c\}$. In practice, one extrapolates up to some threshold; in the next iteration, either the threshold is still satisfied and the result is better than with the standard widening, or it is violated and one extrapolates up to the remaining thresholds.

The benefit of this refinement strongly depends on the choice of relevant thresholds. In [33], [26] we proposed a semantic-based technique for automatically inferring such thresholds, which applies to any control graph, be it intraprocedural, interprocedural or concurrent, without specific assumptions on the abstract domain. Despite its technical simplicity, we showed that our technique is able to infer the relevant thresholds in many practical cases.

6.5.2.2. *Policy Iteration.*

Another direction we investigated for solving the fix-point equation $Y = G(Y), Y \in A$ is the use of *Policy Iteration*, which is a method for the exact solving of optimization and game theory problems, formulated as equations on min max affine expressions. In this context, a *policy* π is a strategy for the min-player, which gives rise to a simplified equation $X = F^{\pi}(X), F^{\pi} \geq F, X \in \mathbb{R}^n$ which is easier to solve that the initial equation $X = F(X), X \in \mathbb{R}^n$. *Policy iteration* iterates on policies rather than iterating the application of F (as in Kleene iteration), using the property that the least fixpoint of F corresponds to the least fixpoint of F^{π} for some π.

[50] showed that the problem of finding the least fixpoint of semantic equations on some abstract domains can be reduced to such equations on min max affine expressions, that can then be solved using Policy Iteration instead of the traditional Kleene iteration with widening described above.

We first investigated the integration of the concept of Policy Iteration in a generic way into existing numerical abstract domains. We implemented it in the APRON library (see module 5.4). This allows the applicability of Policy Iteration in static analysis to be considerably extended.

In particular we considered the verification of programs manipulating Boolean and numerical variables, and we provided an efficient method to integrate the concept of policy in the logico-numerical abstract domain BDDAPRON that mixes Boolean and numerical properties (see module 5.4). This enabled the application of the policy iteration solving method to much more complex programs, that are not purely numerical any more. This work was published in [30].

6.5.3. Analysis of imperative programs

We also studied the analysis of imperative programs. Even if it is preferable to analyze embedded systems described in higher-level languages such as synchronous languages, it is also useful to be able to analyze C programs. Moreover, it enables a wider diffusion of the analysis techniques developed in the team.

6.5.3.1. Inferring Effective Types for Static Analysis of C Programs

This work is a step in the project of connecting the C language to our analysis tool INTERPROC/CONCURINTERPROC (see section 5.5.4). The starting point is the connection made by the industrial partner EADS-IW in the context of the ANR project ASOPT (§8.1.2) from a subset of the C language to INTERPROC. This translation uses the NEWSPEAK intermediate language promoted by EADS [77].

```
typedef struct {
  int n;
} t;
int main()
{
  t x; t* y;
  int *p,*q;
  y = alloc(t); p = &(y->n);
  y = &x; q = &(y->n);
  *p = 1; *q = 2; *p = *p < 1;
  return *p;
}
```

Initial program.

```
typedef enum {
  l0=0,l1=1,l2=2
} e;
typedef struct {
  e n;
} t;
int main()
{
  t x; t* y;
  e *p,*q;
  y = alloc(t); p = &(y->n);
  y = &x; q = &(y->n);
  *p = l1; *q = l2; *p = (*p==l0)?l1:l0;
  return *p;
}
```

Transformed program.

Figure 3. Inferring finite types in C programs

The problem addressed here is that the C language does not have a specific Boolean type: Boolean values are encoded with integers. This is also true for enumerated types, that may be freely and silently cast to and from integers. On the other hand, our verification tool INTERPROC that infers the possible values of variables at each program point may benefit from the information that some integer variables are used solely as Boolean or as enumerated type variables, or more generally as finite type variables with a small domain. Indeed, specialized and efficient symbolic representations such as BDDs are used for representing properties on such variables, whereas approximated representations like intervals and octagons are used for larger domain integers and floating-points variables.

Driven by this motivation, we proposed in [25] a static analysis for inferring more precise types for the variables of a C program, corresponding to their effective use. The analysis addresses a subset of the C99 language, including pointers, structures and dynamic allocation. The principle of the method is very different from type inference techniques used in functional programming languages such as ML, where the types are inferred from the context of use. Instead, our analysis can be seen as a simple points-to analysis, followed by a disjunction version of a constant propagation analysis, and terminated by a program transformation that generates a strongly typed program. Fig. 3 illustrates this process. On this example, we discover that the program is a finite-state one, to which *exact* analysis technique can be applied.

6.5.3.2. Interprocedural analysis with pointers to the stack

This work addressed the problem of interprocedural analysis when side-effect are performed on the stack containing local variables. Indeed, in any language with procedures calls and pointers as parameters (C, Ada) an instruction can modify memory locations anywhere in the call-stack. The presence of such side effects breaks most generic interprocedural analysis methods, which assume that only the top of the stack may be modified. In [29] we presented a method that addresses this issue, based on the definition of an equivalent local semantics in which writing through pointers has a local effect on the stack. Our second contribution in this context is an adequate representation of summary functions that models- the effect of a procedure, not only on the values of its scalar and pointer variables, but also on the values contained in pointed memory locations. Our implementation in the interprocedural analyzer PINTERPROC (see §5.5.4) results in a verification tool that infers relational properties on the value of Boolean, numerical, and pointer variables.

6.6. Component-Based Construction

Participants: Lacramioara Astefanoaei, Alain Girault, Gregor Goessler [contact person], Roopak Sinha, Gideon Smeding.

6.6.1. Incremental converter synthesis

We have proposed and implemented a formal incremental converter-generation algorithm for system-on-chip (SoC) designs. The approach generates a converter, if one exists, to control the interaction between multiple intellectual property (IP) protocols with possible control and data mismatches, and allows pre-converted systems to be re-converted with additional IPs in the future. IP protocols are represented using labeled transition systems (LTS), a simple but elegant abstraction framework which can be extracted from and converted to standard IP description languages such as VHDL. The user can provide control properties, each stated as an LTS with accepting states, to describe desired aspects of the converted system, including fairness and liveness. Furthermore, data specifications can be provided to bound data channels between interacting IPs such that they do not over/under flow. The approach takes into account the uncontrollable environment of a system by allowing users to identify signals exchanged between the SoC and the environment, which the converter can neither suppress nor generate.

Given these inputs, the conversion algorithm first computes the reachable state-space of a maximal non-deterministic converter that ensures (i) the satisfaction of the given data specifications and (ii) the trace equivalence with the given control specifications, using a greatest fix-point computation. It then checks, using the standard algorithm for Büchi games, whether the converter can ensure the satisfaction of the given control specifications (reachability of accepting states) regardless of how the environment behaves. If this is found to be true, deterministic converters can be automatically generated from the maximal non-deterministic converter generated during the first step. The algorithm is proven to be sound and complete, with a polynomial complexity in the state-space sizes of given IP protocols and specifications. It is also shown that it can be used for incremental design of SoCs, where IPs and specifications are added to an SoC in steps. Incremental design allows to constrain the combinatorial explosion of the explored state-space in each step, and also reduces on-chip wire congestion by decentralizing the conversion process.

A Java implementation has been created, and experimental results show that the algorithm can handle complex IP mismatches and specifications in medium to large AMBA based SoC systems. Future work involves creating a library of commonly-encountered specifications in SoC design such as sharing of control signals between interacting IPs using buffers, signal lifespans, and the generation of optimal converters based on quantitative criteria such as minimal power usage.

This work has been done within the AFMES associated team with the Electric and Computer Engineering Department of the University of Auckland.

6.6.2. Causality Analysis in Contract Violation

Establishing liabilities in case of litigation is generally a delicate matter. It becomes even more challenging when IT systems are involved. Generally speaking, a party can be declared liable for a damage if a fault can be attributed to that party and this fault has caused the damage. The two key issues are thus to establish convincing evidence with respect to (1) the occurrence of the fault and (2) the causality relation between the fault and the damage. The first issue concerns the technique used to log the relevant events of the system and to ensure that the logs can be produced (and have some value) in court. The second issue is especially complex when several faults are detected in the logs and the impact of these faults on the occurrence of the failure has to be assessed. In [6] we have focused on this second issue and proposed a formal framework for reasoning about causality. A system based on this framework could be used to provide relevant information to the expert, the judge, or the parties themselves (in case of amicable settlement) to analyze the origin of the failure of an IT system.

The notion of causality has been studied for a long time in computer science, but with very different perspectives and goals. In the distributed systems community, causality (following Lamport's seminal paper [82]) is seen essentially as a temporal property. In our context, the temporal ordering contributes to the analysis, but it is obviously not sufficient to establish the *logical causality* required to rule on a matter of liability: the fact that an event e_1 has occurred before an event e_2 does not imply that e_1 was the cause for e_2 (or that e_2 would not have occurred if e_1 had not occurred).

Our formal model is based on components interacting according to well identified *interaction models* [5]. Each component is associated with an individual *contract* which specifies its expected behavior. The system itself is associated with a *global contract* which is assumed to be implied by the composition of the individual contracts.

In [6] we have defined several variants of logical causality. The first variant, *necessary causality*, characterizes cases when the global contract would not have been violated if the local contract had been fulfilled. The second variant, *sufficient causality*, characterizes cases when the global contract would have been violated even if all the other components had fulfilled their contracts. In other words, the violation of its contract by a single component was sufficient to violate the global contract.

We are currently extending to framework to other models of computation and communication, in particular, to timed automata.

6.6.3. Realizability of Choreographies for Services Interacting Asynchronously

Choreography specification languages describe from a global point of view interactions among a set of services in a system to be designed. Given a choreography specification, the goal is to obtain a distributed implementation of the choreography as a system of communicating peers. These peers can be given as input (*e.g.*, obtained using discovery techniques) or automatically generated by projection from the choreography. Checking whether some set of peers implements a choreography specification is called *realizability*. This check is in general undecidable if asynchronous communication is considered, that is, services interact through message buffers.

In [24] we consider conversation protocols as a choreography specification language, and leverage a recent decidability result [54] to check automatically the realizability of these specifications by a set of peers under an asynchronous communication model with a priori unbounded buffers.

6.6.4. A Theory of Fault Recovery for Component-Based Models

In [18] we have introduced a theory of fault recovery for component-based models. A model is specified in terms of a set of atomic components that are incrementally composed and synchronized by a set of glue operators. We define what it means for such models to provide a recovery mechanism, so that the model converges to its normal behavior in the presence of faults. We identify corrector (atomic or composite) components whose presence in a model is essential to guarantee recovery after the occurrence of faults. We also formalize component based models that effectively separate recovery from functional concerns.

6.7. Aspect-Oriented Programming

Participants: Henri-Charles Blondeel, Pascal Fradet [contact person], Alain Girault, Marnes Hoff.

The goal of Aspect-Oriented Programming (AOP) is to isolate aspects (such as security, synchronization, or error handling) which cross-cut the program basic functionality and whose implementation usually yields tangled code. In AOP, such aspects are specified separately and integrated into the program by an automatic transformation process called *weaving*.

Although this paradigm has great practical potential, it still lacks formalization and undisciplined uses make reasoning on programs very difficult. Our work on AOP addresses these issues by studying foundational issues (semantics, analysis, verification) and by considering domain-specific aspects (availability, fault tolerance or refinement aspects) as formal properties.

6.7.1. Aspects Preserving Properties

Aspect Oriented Programming can arbitrarily distort the semantics of programs. In particular, weaving can invalidate crucial safety and liveness properties of the base program.

We have identified categories of aspects that preserve some classes of properties [10]. Our categories of aspects comprise, among others, observers, aborters, and confiners. For example, observers do not modify the base program's state and control-flow (*e.g.*, persistence, profiling, and debugging aspects). These categories are defined formally based on a language independent abstract semantic framework. The classes of properties are defined as subsets of LTL for deterministic programs and CTL* for non-deterministic ones. We have formally proved that, for any program, the weaving of any aspect in a category preserves any property in the related class.

In a second step, we have designed for each aspect category a specialized aspect language which ensures that any aspect written in that language belongs to the corresponding category. These languages preserve the corresponding classes of properties by construction.

This work was conducted in collaboration with Rémi Douence from the ASCOLA INRIA team at École des Mines de Nantes.

6.7.2. Fault Tolerance Aspects

In the recent years, we have studied the implementation of specific fault tolerance techniques in real-time embedded systems using program transformation [1]. We are now investigating the use of fault-tolerance aspects in digital circuits. To this aim, we consider program transformations for hardware description languages (HDL). Our goal is to design an aspect language allowing users to specify and tune a wide range of fault tolerance techniques, while ensuring that the woven HDL program remains synthesizable. The advantage would be to produce fault-tolerant circuits by specifying fault-tolerant strategies separately from the functional specifications.

We have reviewed the different fault tolerant techniques used in integrated circuits: concurrent error detection, error detecting and correcting codes (Hamming, Berger codes, ...), spatial and time redundancy. We have designed a simple hardware description language inspired from Lustre and Lucid Synchrone. It is a core functional language manipulating synchronous boolean streams. Faults are represented by bit flips and we take into account all fault models of the form “at most k faults within n clock signals”. The language semantics as well as the fault model have been formalized in Coq. The next step is to express standard fault tolerance techniques as program transformations and prove that they allow to tolerate all faults of a given model.

6.7.3. Refinement Aspects

Chemical programming describes computation in terms of a *chemical solution* in which molecules (representing data) interact freely according to *reaction rules* (representing the program). Solutions are represented by multisets of elements and reactions by rewrite rules which consume and produce new elements according to conditions. This paradigm makes it possible to express programs without artificial sequentiality in a very abstract way. It bridges the gap between specification and implementation languages.

A drawback of chemical languages is that their very high-level nature usually leads to very inefficient programs. We have proposed a refinement oriented approach where the basic functionality is expressed as a chemical program whereas efficiency is achieved separately by:

- structuring the multiset with a data type defining neighborhood relations;
- describing the selection of elements according to their neighborhood;
- specifying the evaluation strategy (*i.e.*, the application of rules and termination).

Using these three implementation aspects (data structure, selection and strategy), the chemical program can then be refined automatically into an efficient low-level program. The crucial methodological advantage is that logical issues are decoupled from efficiency issues.

This research, that takes place within the AUTOCHEM project (see Section 8.1.1), is done in collaboration with Jean-Louis Giavitto (Ircam, Paris). It is the subject matter of Marnes Hoff's PhD thesis.

7. Contracts and Grants with Industry

7.1. Grants with Industry

- STMicroelectronics, starting in 12/2011: CIFRE contract for the PhD of Vagelis Bebelis.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR AutoChem: Chemical Programming

Participants: Pascal Fradet [contact person], Marnes Hoff.

The AUTOCHEM project aims at investigating and exploring the use of chemical languages (see Section 6.7.3) to program complex computing infrastructures such as grids and real-time deeply-embedded systems. The consortium includes INRIA Rennes – Bretagne Atlantique (PARIS team, Rennes), INRIA Grenoble – Rhône-Alpes (POP ART team, Montbonnot), IBISC (CNRS/Université d'Evry) and CEA List (Saclay). The project started at the end of 2007 and ended in November 2011.

8.1.2. ANR Asopt: Analyse Statique et OPTimisation

Participants: Bertrand Jeannet [contact person, coordinator], Lies Lakhdar-Chaouch, Pascal Sotin, Peter Schrammel.

The ASOPT (Analyse Statique et OPTimisation) project [end of 2008-2011] brings together static analysis (INRIA-POP ART, VERIMAG, CEA LMeASI), optimisation, and control/game theory experts (CEA LMeASI, INRIA-MAXPLUS) around some program verification problems. POP ART is the project coordinator.

Many abstract interpretations attempt to find “good” geometric shapes verifying certain constraints; this not only applies to purely numerical abstractions (for numerical program variables), but also to abstractions of data structures (arrays and more complex shapes). This problem can often be addressed by optimisation techniques, opening the possibility of exploiting advanced techniques from mathematical programming.

The purpose of ASOPT is to develop new abstract domains and new resolution techniques for embedded control programs, and in the longer run, for numerical simulation programs.

8.1.3. ANR Vedecy: Verification and Design of Cyber-physical Systems

Participants: Gregor Goessler [contact person], Bertrand Jeannet, Sebti Mouelhi.

The VEDECY project brings together hybrid systems and formal methods experts. Three partners are involved: Laboratoire Jean Kuntzmann (LJK), INRIA POP ART, and VERIMAG.

VEDECY aims at pursuing fundamental research towards the development of algorithmic approaches to the verification and design of cyber-physical systems. Cyber-physical systems result from the integration of computations with physical processes: embedded computers control physical processes which in return affect computations through feedback loops. They are ubiquitous in current technology and their impact on lives of citizens is meant to grow in the future (autonomous vehicles, robotic surgery, energy efficient buildings, ...).

Cyber-physical systems applications are often safety critical and therefore reliability is a major requirement. To provide assurance of reliability, model based approaches and formal methods are appealing. Models of cyber-physical systems are heterogeneous by nature: discrete dynamic systems for computations and continuous differential equations for physical processes. The theory of hybrid systems offers a sound modeling framework for cyber-physical systems. The purpose of VEDECY is to develop hybrid systems techniques for the verification and the design of cyber-physical systems.

8.1.4. INRIA Large Scale Action Synchronics: Language Platform for Embedded System Design

Participants: Gwenaël Delaval, Alain Girault [contact person, co-coordinator], Bertrand Jeannet, Xavier Nicollin, Peter Schrammel.

The SYNCHRONICS (Language Platform for Embedded System Design) project [mid-2008 to mid-2012] gathers 9 permanent researchers on the topic of embedded systems design: B. Caillaud (INRIA Rennes – Bretagne Atlantique), A. Cohen, L. Mandel, and M. Pouzet (INRIA-Saclay and ENS Ulm), G. Delaval, A. Girault, and B. Jeannet (INRIA Grenoble – Rhône-Alpes), E. Jahier and P. Raymond (VERIMAG).

SYNCHRONICS capitalizes on recent extensions of data-flow synchronous languages, as well as relaxed forms of synchronous composition or compilation techniques for various platform, to address two main challenges with a language-centered approach: (i) the co-simulation of mixed discrete-continuous specifications, and more generally the co-simulation of programs and properties (either discrete or continuous); (ii) the ability, inside the programming model, to account for the architecture constraints (execution time, memory footprint, energy, power, reliability, etc.).

8.1.5. Collaborations inside INRIA

- VERTECS at INRIA Rennes – Bretagne Atlantique is working with us on applications of discrete controller synthesis, and in particular on the tool SIGALI.
- P. Fradet cooperates with R. Douence (ASCOLA, École des Mines de Nantes) on aspect-oriented programming.
- A. Girault cooperates with D. Trystram (MOAIS, INRIA Grenoble – Rhône-Alpes) on scheduling and dependability, with E. Rutten (SARDES, INRIA Grenoble – Rhône-Alpes) and H. Marchand (VERTECS, INRIA Rennes – Bretagne Atlantique) on optimal discrete controller synthesis, and with A. Benoit, F. Dufossé and Y. Robert (GRAAL, INRIA Grenoble – Rhône-Alpes) on multi-criteria scheduling.

- G. Goessler cooperates with D. Le Métayer (LICIT, INRIA Grenoble – Rhône-Alpes) on logical causality and with G. Salaün (VASY, INRIA Grenoble – Rhône-Alpes) on realizability of choreographies with asynchronous communication.
- B. Jeannet cooperates with A. Miné and X. Rival (ABSTRACTION, INRIA Paris – Rocquencourt) and X. Allamigeon (MAXPLUS, INRIA Saclay – Île-de-France) on static analysis and abstract interpretation.
- G. Delaval cooperates with H. Marchand (VERTECS, INRIA Rennes – Bretagne Atlantique) and É. Rutten (SARDES, INRIA Grenoble – Rhône-Alpes) on modular controller synthesis and its applications.
- G. Delaval, A. Girault and B. Jeannet collaborate with the PARKAS team of ENS Ulm (INRIA Paris – Rocquencourt) on the distribution of higher-order synchronous data-flow programs and on static analysis of hybrid programs.

8.1.6. Cooperations with other laboratories

- P. Fradet cooperates with J.-L. Giavitto (CNRS/Ircam) on refinement of chemical programs.
- A. Girault collaborates with P. Roop, Z. Salcic, and S. Andalam (University of Auckland, New Zealand) and A. Malik (IBM Watson, USA) in the context of the AFMES associated team, with H. Kalla (University of Batna, Algeria) and I. Assayad (University of Casablanca, Morocco) on multicriteria scheduling.
- G. Goessler collaborates with A. Girard (LJK, Grenoble) on multi-scale controller synthesis, with J. Sifakis (EPFL) on distribution under real-time constraints, with J.-B. Raclet (IRIT, Toulouse) on modal contracts, with I. Lee and O. Sokolsky (U. of Pennsylvania) on causality analysis for medical devices, and with M. Bozga (VERIMAG) and B. Bonakdarpour (U. of Waterloo, Canada) on fault tolerance in component-based systems.
- A. Girault and G. Goessler collaborate with P. Roop (University of Auckland, New Zealand) on incremental converter synthesis.
- B. Jeannet collaborates with N. Halbwachs and M. Péron (VERIMAG), E. Goubault and S. Putot (CEA Saclay) on static analysis and abstract interpretation.
- G. Delaval and A. Girault collaborate with X. Nicollin (VERIMAG) on the automatic distribution of synchronous programs.

8.2. European Initiatives

8.2.1. Collaborations in European Programs, except FP7

Program: ARTEMISIA.

Project acronym: CESAR [32].

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

Duration: January 2009 – April 2012.

Partners: There are 59 partners from academia and industry (both SMEs and large companies).

Abstract: We are particularly involved in the following sub-programs:

SP1: Task Force Safety 1.5.1 (State of the art survey on safety and diagnosability for cost-efficient safety critical emebedded systems) and 1.5.2 (Identification of requirements for comon cross domain core safety and diagnosability techniques and methods).

SP2: Requirements Engineering, along with two other INRIA teams (S4 and TRISKELL, from INRIA Rennes). We shall work on contracts based design for traceability.

[32] http://www.cesarproject.eu

8.3. International Initiatives

8.3.1. INRIA Associate Teams

8.3.1.1. AFMES

Title: Advanced Formal Methods for Embedded Systems.

INRIA principal investigator: Alain Girault.

International Partner:

> Institution: University of Auckland (New Zealand).
>
> Laboratory: Department of Electrical and Computer Engineering.
>
> Principal investigator: Zoran Salcic.

Duration: January 2010 – December 2012.

See also: http://pop-art.inrialpes.fr/~girault/Projets/Afmes/

Embedded systems are characterized by several constraints, such as determinism and bounded reaction time. Accordingly, design methods for embedded systems should, when possible, guarantee these properties by construction. This allows the shifting of the burden of checking these constraints from the programmer to the design method and the associated compilers and code generation tools. In order to achieve this, our goal is to improve the existing design methods in several key directions: (1) Incremental converter synthesis. (2) Programming language for adaptive computing (SystemJ and beyond). (3) Time predictable programming language and execution architectures. Together, these advanced methods will provide a higher level of safety in the design of embedded systems.

8.3.2. Visits of International Scientists

- Hamoudi Kalla, assistant professor at University of Batna, Algeria, September 2011.
- Ismail Assayad, assistant professor at University of Casablanca, Morocco, September 2011.

8.3.2.1. Internship

- Emmanouil Komninos, 02-07/2011, co-advised by Pascal Fradet and Alain Girault, Power consumption optimization of data-flow applications on many-core systems, MSc at KTH, Sweden.

9. Dissemination

9.1. Animation of the scientific community

- Gwenaël Delaval was co-organizer of the international workshop SYNCHRON'11 (international open workshop on Synchronous Programming).
- P. Fradet served in the external review committee of PLDI'2011 (*ACM SIGPLAN conference on Programming Language Design and Implementation*). He was examiner for the PhD of Julien Tesson (University of Orléans).
- Alain Girault served in the programme committees of the international conferences DATE'2011, DAC'2011, MEMOCODE'2011, LAFT'2011, MSR'2011, and APSIPA-ASC'2011. He was referee for the PhD of Alexandru Drobila (University of Besançon) and Mohamed Fellahi (University of Paris Sud), and examiner for the PhD of Fanny Duffossé (ENS Lyon).
- Gregor Goessler served in the programme committees of the international conference DATE'2012 and of the international workshops FOCLASA'2011 and LAFT'2011.

- Bertrand Jeannet served in the programme committee of the international conference DATE'2012. He was examiner for the PhD of Assalé Adjé (Ecole Polytechnique) and of Khalil Gorbhal (Ecole Polytechnique).

9.2. Teaching

9.2.1. Advising

PhDs:

- Marnes Hoff, co-advised by P. Fradet (with J.-L. Giavitto, Université d'Evry), since 04/2008 until 05/2011, PhD in computer science, Grenoble University.
- Henri-Charles Blondeel, co-advised by P. Fradet and A. Girault, until 06/2011, PhD in computer science, Grenoble University.
- Peter Schrammel, co-advised by B. Jeannet and A. Girault since 07/2009, PhD in computer science, Grenoble University.
- Gideon Smeding, co-advised by G. Goessler and J. Sifakis since 12/2009, PhD in computer science, Grenoble University.
- Vagelis Bebelis, co-advised by P. Fradet and A. Girault, since 12/2011, PhD in computer science, Grenoble University.

Masters:

- Emmanouil Komninos, 02-07/2011, co-advised by Pascal Fradet and Alain Girault, Power consumption optimization of data-flow applications on many-core systems, MSc at KTH.

9.2.2. University Teaching

Gwenaël Delaval teaches algorithmics and programming at Université Joseph Fourier (170h in 2011–2012).

10. Bibliography

Major publications by the team in recent years

[1] T. AYAV, P. FRADET, A. GIRAULT. *Implementing Fault-Tolerance in Real-Time Programs by Automatic Program Transformations*, in "ACM Trans. Embedd. Comput. Syst.", July 2008, vol. 7, n^{o} 4, p. 1–43.

[2] S. DJOKO DJOKO, R. DOUENCE, P. FRADET. *Aspects Preserving Properties*, in "Proc. of the ACM SIGPLAN 2008 Symposium on Partial Evaluation and Program Manipulation (PEPM'08)", San Francisco, ACM, January 2008, p. 135–145.

[3] A. GIRAULT, H. KALLA. *A Novel Bicriteria Scheduling Heuristics Providing a Guaranteed Global System Failure Rate*, in "IEEE Trans. Dependable Secure Comput.", December 2009, vol. 6, n^{o} 4, p. 241–254, Research report INRIA 6319, http://hal.inria.fr/inria-00177117.

[4] A. GIRAULT, É. RUTTEN. *Automating the Addition of Fault Tolerance with Discrete Controller Synthesis*, in "Formal Methods in System Design", October 2009, vol. 35, n^{o} 2, p. 190–225, http://www.springerlink.com/content/w726262156h4822j.

[5] G. GÖSSLER, J. SIFAKIS. *Composition for Component-based Modeling*, in "Science of Computer Programming", 3 2005, vol. 55, n^{o} 1–3, p. 161–183.

[6] G. GÖSSLER, D. LE MÉTAYER, J.-B. RACLET. *Causality Analysis in Contract Violation*, in "Runtime Verification", LNCS, Springer-Verlag, 2010, p. 270-284.

[7] B. JEANNET, A. LOGINOV, T. REPS, M. SAGIV. *A Relational Approach to Interprocedural Shape Analysis*, in "ACM Trans. on Programming Languages and Systems", 2010, vol. 32, n^o 2, http://doi.acm.org/10.1145/1667048.1667050.

[8] T. LE GALL, B. JEANNET. *Lattice Automata: A Representation of Languages over an Infinite Alphabet, and some Applications to Verification*, in "Static Analysis Symposium, SAS'07", Copenhagen (Denmark), LNCS, August 2007, vol. 4634, http://pop-art.inrialpes.fr/people/bjeannet/publications/sas07.ps.

[9] A. MALIK, Z. SALCIC, P. ROOP, A. GIRAULT. *SystemJ: A GALS Language for System Level Design*, in "Elsevier Computer Languages, Systems and Structures", December 2010, vol. 36, n^o 4, p. 317–344.

Publications of the year

Articles in International Peer-Reviewed Journal

[10] S. DJOKO DJOKO, R. DOUENCE, P. FRADET. *Aspects Preserving Properties*, in "Science of Computer Programming", March 2012, vol. 77, n^o 3, p. 393-422, http://www.sciencedirect.com/science/article/pii/S0167642311001870.

[11] G. GÖSSLER. *Component-Based Modeling and Reachability Analysis of Genetic Networks*, in "IEEE/ACM Trans. Comput. Biology Bioinform.", 2011, vol. 8, n^o 3, p. 672–682.

[12] A. MALIK, A. GIRAULT, Z. SALCIC. *Formal Semantics, Compilation and Execution of the GALS Programming Language DSystemJ*, in "IEEE Trans. Parallel and Distributed Systems", 2012, to appear.

[13] P. SCHRAMMEL, B. JEANNET. *Applying Abstract Acceleration to (Co-)Reachability Analysis of Reactive Programs*, in "Journal of Symbolic Computation – Special issue on WING2010", 2011, to appear.

[14] A. B. SEBOUI, N. B. HADJ-ALOUANE, G. DELAVAL, É. RUTTEN, M. M. YEDDES. *An approach for the synthesis of decentralised supervisors for distributed adaptive systems*, in "International Journal of Critical Computer-Based Systems", 2011, vol. 2, n^o 3/4, p. 246-265.

International Conferences with Proceedings

[15] S. ABOUBEKR, G. DELAVAL, R. PISSARD-GIBOLLET, É. RUTTEN, D. SIMON. *Automatic Generation of Discrete Handlers of Real-Time Continuous Control Tasks*, in "Proceedings of the 18th IFAC World Congress", Milano, Italy, August 2011, vol. 18.

[16] S. ANDALAM, P. ROOP, A. GIRAULT. *Pruning Infeasible Paths for Tight WCRT Analysis of Synchronous Programs*, in "Design Automation and Test in Europe Conference, DATE'11", Grenoble, France, April 2011.

[17] I. ASSAYAD, A. GIRAULT, H. KALLA. *Tradeoff Exploration between Reliability, Power Consumption, and Execution Time*, in "International Conference on Computer Safety, Reliability and Security, SAFECOMP'11", Napoli, Italy, LNCS, Springer-Verlag, September 2011, vol. 6894, p. 437–451.

[18] B. BONAKDARPOUR, M. BOZGA, G. GÖSSLER. *A Theory of Fault Recovery for Component-Based Models*, in "Symposium on Reliable and Distributed Systems, SRDS'11", IEEE, 2011, p. 265–270.

[19] T. BOUHADIBA, Q. SABAH, G. DELAVAL, É. RUTTEN. *Synchronous Control of Reconfiguration in Fractal Component-based Systems: a Case Study*, in "Proceedings of the ninth ACM international conference on Embedded software", New York, NY, USA, EMSOFT'11, ACM, 2011, p. 309–318, http://doi.acm.org/10.1145/2038642.2038690.

[20] F. BOYER, G. DELAVAL, N. DE PALMA, O. GRUBER, É. RUTTEN. *Case Studies in Discrete Control of Autonomic Computing Systems*, in "Proc. of the Sixth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2011)", Karlsruhe, Germany, June 2011.

[21] J. CÁMARA, A. GIRARD, G. GÖSSLER. *Safety Controller Synthesis for Switched Systems Using Multi-Scale Symbolic Models*, in "CDC-ECC", 2011, to appear.

[22] J. CÁMARA, A. GIRARD, G. GÖSSLER. *Synthesis of switching controllers using approximately bisimilar multiscale abstractions*, in "Hybrid Systems Computation and Control, HSCC'11", M. CACCAMO, E. FRAZZOLI, R. GROSU (editors), ACM, 2011, p. 191–200.

[23] P. FRADET, A. GIRAULT, P. POPLAVKO. *SPDF: A Schedulable Parametric DataFlow MoC*, in "Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE'12)", March 2012, to appear.

[24] G. GÖSSLER, G. SALAÜN. *Realizability of Choreographies for Services Interacting Asynchronously*, in "Formal Aspects of Computer Software, FACS'11", LNCS, Springer, 2012, to appear.

[25] B. JEANNET, P. SOTIN. *Inferring Effective Types for Static Analysis of C Programs*, in "Int. Workshop on Numerical and Symbolic Abstract Domains, NSAD'11", Venice (Italy), ENTCS, 2011, to appear.

[26] L. LAKHDAR-CHAOUCH, B. JEANNET, A. GIRAULT. *Widening with Thresholds for Programs with Complex Control Graphs*, in "Automated Technology for Verification and Analysis, ATVA'11", Taipei (Taiwan), LNCS, 2011, vol. 6996, p. 492–502.

[27] A. MALIK, A. GIRAULT, Z. SALCIC. *A GALS Language for Dynamic Distributed and Reactive Programs*, in "International Conference on Application of Concurrency to System Design, ACSD'11", Newcastle, UK, IEEE, June 2011.

[28] P. SCHRAMMEL, B. JEANNET. *Logico-Numerical Abstract Acceleration and Application to the Verification of Data-Flow Programs*, in "Static Analysis Symposium, SAS'11", Venice (Italy), LNCS, 2011, vol. 6887, p. 233–248.

[29] P. SOTIN, B. JEANNET. *Precise Interprocedural Analysis in the Presence of Pointers to the Stack*, in "European Symposium on Programming, ESOP'11", Sarrebrück, LNCS, 2011, vol. 6602, p. 459–479.

[30] P. SOTIN, B. JEANNET, F. VÉDRINE, É. GOUBAULT. *Policy Iteration within Logico-Numerical Abstract Domains*, in "Automated Technology for Verification and Analysis, ATVA'11", LNCS, 2011, vol. 6996, p. 290–305.

National Conferences with Proceeding

[31] G. Delaval, É. Rutten, H. Marchand. *Intégration de la synthèse de contrôleurs discrets dans un langage de programmation*, in "Actes du 8ème Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR 2011)", Lille, France, November 2011, p. 125–140.

Research Reports

[32] P. Fradet, A. Girault, P. Poplavko. *SPDF: A Schedulable Parametric Dataflow Graph Model (extended version)*, INRIA, December 2011, n^o 7828.

[33] L. Lakhdar-Chaouch, B. Jeannet, A. Girault. *Widening with Thresholds for Programs with Complex Control Graphs*, INRIA, July 2011, n^o RR-7673, http://hal.inria.fr/inria-00606961/en/.

Other Publications

[34] P. Fradet, A. Girault, P. Poplavko. *A Statically Analyzable Dataflow Model for Dynamic Streaming*, in "5th Workshop on Mapping of Applications to MPSoCs", 2011, one-page abstract.

[35] P. Fradet, A. Girault, P. Poplavko. *A Statically Analyzable Dataflow Model for Dynamic Streaming*, in "Platform 2012 Developers' Conference", 2011, one-page abstract.

[36] P. Fradet, A. Girault, P. Poplavko, A.-E. Ozcan. *A Dataflow Model for Interactive Data-dependent Streaming Applications*, in "Workshop on Designing for Embedded Parallel Computing Platforms: Architectures, Design Tools, and Applications", 2011, poster.

References in notes

[37] *Norme Internationale – Automates programmables : Langages de programmation*, CEI (Commission Électrotechnique Internationale), 1993.

[38] X. Allamigeon, S. Gaubert, É. Goubault. *Inferring Min and Max Invariants Using Max-Plus Polyhedra*, in "Static Analysis Symposium, SAS'08", LNCS, 2008, vol. 5079, p. 189–204.

[39] C. Ancourt, F. Coelho, F. Irigoin. *A Modular Static Analysis Approach to Affine Loop Invariants Detection*, in "Numerical and Symbolic Abstract Domains", ENTCS, Elsevier, 2010, vol. 267, p. 3–16.

[40] S. Andalam, P. Roop, A. Girault. *Predictable Multithreading of Embedded Applications Using PRET-C*, in "International Conference on Formal Methods and Models for Codesign, MEMOCODE'10", Grenoble, France, July 2010.

[41] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*, Masson, 1992.

[42] E. Asarin, O. Bournez, T. Dang, O. Maler, A. Pnueli. *Effective Synthesis of Switching Controllers for Linear Systems*, in "Proceedings of the IEEE", 2000, vol. 88, n^o 7, p. 1011–1025.

[43] A. Benveniste, T. Bourke, B. Caillaud, M. Pouzet. *Divide and recycle: types and compilation for a hybrid synchronous language*, in "LCTES", ACM, 2011, p. 61-70.

[44] R. Bryant. *Graph-based algorithms for boolean function manipulation*, in "IEEE Trans. on Computers", 1986, vol. C-35, n^{o} 8, p. 677–692.

[45] P. Caspi, M. Pouzet. *Synchronous Kahn Networks*, in "ACM SIGPLAN International Conference on Functional Programming, ICFP'96", Philadelphia (PA), USA, ACM Press, May 1996.

[46] C. Cassandras, S. Lafortune. *Introduction to Discrete Event Systems*, Kluwer, 1999.

[47] D. Chase, M. Wegman, F. Zadeck. *Analysis of Pointers and Structures*, in "Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation", ACM Press, 1990, p. 296–310, http://doi.acm.org/10.1145/93542.93585.

[48] E. Clarke, E. Emerson, A. Sistla. *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications*, in "ACM Trans. Programming Languages and Systems", 4 1986, vol. 8, n^{o} 2, p. 244-263, Introduction of Model-checking; impartiality, justice, fairness.

[49] D. Clarke, T. Jéron, V. Rusu, E. Zinovieva. *STG: a Symbolic Test Generation tool*, in "(Tool paper) Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)", LNCS, 2002, vol. 2280.

[50] A. Costan, S. Gaubert, É. Goubault, M. Martel, S. Putot. *A Policy Iteration Algorithm for Computing Fixed Points in Static Analysis of Programs*, in "Computer Aided Verification, CAV'05", LNCS, 2005, vol. 3576, p. 462–475.

[51] P. Cousot, R. Cousot. *Abstract Interpretation and Application to Logic Programs*, in "Journal of Logic Programming", 1992, vol. 13, n^{o} 2–3, p. 103–179.

[52] P. Cousot, N. Halbwachs. *Automatic discovery of linear restraints among variables of a program*, in "5th ACM Symposium on Principles of Programming Languages, POPL'78", Tucson (Arizona), January 1978.

[53] P. D'Argenio, B. Jeannet, H. Jensen, K. Larsen. *Reduction and Refinement Strategies for Probabilistic Analysis*, in "Process Algebra and Probabilistic Methods - Performance Modelling and Verification, PAPM-PROBMIV'02", Copenhagen (Denmark), LNCS, July 2002, vol. 2399.

[54] P. Darondeau, B. Genest, P. Thiagarajan, S. Yang. *Quasi-static scheduling of communicating tasks*, in "Inf. Comput.", 2010, vol. 208, n^{o} 10, p. 1154-1168.

[55] G. Delaval. *Répartition modulaire de programmes synchrones*, INPG, INRIA Grenoble Rhône-Alpes, July 2008, PhD thesis.

[56] G. Delaval, A. Girault, M. Pouzet. *A Type System for the Automatic Distribution of Higher-order Synchronous Dataflow Programs*, in "International Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES'08", Tucson (AZ), USA, ACM, June 2008, p. 101–110, ftp://ftp.inrialpes.fr/pub/bip/pub/girault/Publications/Lctes08/main.pdf.

[57] G. Delaval, H. Marchand, É. Rutten. *Contracts for Modular Discrete Controller Synthesis*, in "ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2010)", Stockholm, Sweden, April 2010, http://pop-art.inrialpes.fr/people/delaval/pub/lctes2010.pdf.

[58] D. DELMAS, É. GOUBAULT, S. PUTOT, J. SOUYRIS, K. TEKKAL, F. VÉDRINE. *Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software*, in "Formal Methods for Industrial Critical Systems, FMICS'09", LNCS, 2009, vol. 5825.

[59] E. DUMITRESCU, A. GIRAULT, H. MARCHAND, É. RUTTEN. *Multicriteria Optimal Reconfiguration of Fault-Tolerant Real-Time Tasks*, in "Workshop on Discrete Event Systems, WODES'10", Berling, Germany, IFAC, New-York, September 2010.

[60] A. DUNKELS, O. SCHMIDT, T. VOIGT, M. ALI. *Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems*, in "Conference on Embedded Networked Sensor Systems, SenSys'06", Boulder (CO), USA, ACM, November 2006.

[61] F. GAUCHER, E. JAHIER, B. JEANNET, F. MARANINCHI. *Automatic State Reaching for Debugging Reactive Programs*, in "5th Int. Workshop on Automated and Algorithmic Debugging, AADEBUG'03", September 2003.

[62] K. GHORBAL, É. GOUBAULT, S. PUTOT. *The Zonotope Abstract Domain Taylor1+*, in "Computer Aided Verification, CAV'09", LNCS, 2009, vol. 5643.

[63] A. GIRARD, G. PAPPAS. *Approximation metrics for discrete and continuous systems*, in "IEEE Trans. on Automatic Control", 2007, vol. 52, n^o 5, p. 782–798.

[64] A. GIRAULT. *System-Level Design of Fault-Tolerant Embedded Systems*, October 2006, vol. 67.

[65] A. GIRAULT, H. KALLA, M. SIGHIREANU, Y. SOREL. *An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules*, in "International Conference on Dependable Systems and Networks, DSN'03", San-Francisco (CA), USA, IEEE, June 2003.

[66] A. GIRAULT, H. KALLA, Y. SOREL. *Transient Processor/Bus Fault Tolerance for Embedded Systems*, in "IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'06", Braga, Portugal, Springer, October 2006, p. 135–144.

[67] A. GIRAULT, É. RUTTEN. *Automating the Addition of Fault Tolerance with Discrete Controller Synthesis*, in "Formal Methods in System Design", October 2009, vol. 35, n^o 2, p. 190–225, http://www.springerlink.com/content/w726262156h4822j.

[68] L. GONNORD. *Accélération abstraite pour l'amélioration de la précision en Analyse des Relations Linéaires*, Université Joseph Fourier, Grenoble, October 2007.

[69] L. GONNORD, N. HALBWACHS. *Combining widening and acceleration in linear relation analysis*, in "Static Analysis Symposium (SAS)", Seoul, Korea, Aug 2006, p. 144–160.

[70] D. GOPAN, T. REPS. *Guided Static Analysis*, in "Static Analysis Symposium, SAS'07", LNCS, August 2007, vol. 4634, p. 349–365, http://dx.doi.org/10.1007/978-3-540-74061-2_22.

[71] G. GÖSSLER. *Component-based Design of Heterogeneous Reactive Systems in* PROMETHEUS, INRIA, 2006, n^o 6057.

[72] G. GÖSSLER, J. SIFAKIS. *Priority Systems*, in "Proc. FMCO'03", F. DE BOER, M. BONSANGUE, S. GRAF, W.-P. DE ROEVER (editors), LNCS, Springer-Verlag, 2004, vol. 3188, p. 314-329.

[73] N. HALBWACHS. *Synchronous Programming of Reactive Systems*, Kluwer, 1993.

[74] N. HALBWACHS. *Synchronous Programming of Reactive Systems – a Tutorial and Commented Bibliography*, in "Proc. of the Int. Conf. on Computer-Aided Verification, CAV'98, Vancouver, Canada", Springer-Verlag, 1998, LNCS Vol. 1427.

[75] N. HALBWACHS, Y. PROY, P. ROUMANOFF. *Verification of real-time systems using linear relation analysis*, in "Formal Methods in System Design", August 1997, vol. 11, n^o 2.

[76] T. HENZINGER, P. HO, H. WONG-TOÏ. *HyTech: The Next Generation*, in "RTSS'95", 1995.

[77] C. HYMANS, O. LEVILLAIN. *Newspeak, Doubleplussimple Minilang for Goodthinkful Static Analysis of C*, EADS, 2008, http://penjili.org.

[78] B. JEANNET, P. D'ARGENIO, K. LARSEN. *RAPTURE: A tool for verifying Markov Decision Processes*, in "Tools Day, International Conference on Concurrency Theory, CONCUR'02", Brno (Czech Republic), August 2002, Technical Report, Faculty of Informatics at Masaryk University Brno.

[79] B. JEANNET. *Dynamic Partitioning in Linear Relation Analysis. Application To The Verification Of Reactive Systems*, in "Formal Methods in System Design", July 2003, vol. 23, n^o 1, p. 5–37.

[80] B. JEANNET, T. JÉRON, V. RUSU, E. ZINOVIEVA. *Symbolic Test Selection based on Approximate Analysis*, in "11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)", Edinburgh (UK), LNCS, April 2005, vol. 3440.

[81] E. KOMNINOS. *Power consumption optimization of data-flow applications on many-core systems*, KTH, August 2011.

[82] L. LAMPORT. *Time, Clocks, and the Ordering of Events in a Distributed System*, in "CACM", 1978, vol. 21, n^o 7, 558 565.

[83] J.-Y. LE BOUDEC, P. THIRAN. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, Lecture Notes in Computer Science, Springer, 2001, vol. 2050.

[84] Y. LEE, A. ZOMAYA. *Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling*, in "IEEE/ACM International Symposium on Cluster Computing and the Grid, SCCG'09", 2009.

[85] O. MALER, A. PNUELI, J. SIFAKIS. *On the Synthesis of Discrete Controllers for Timed Systems*, in "Proc. of STACS'95", LNCS, Springer Verlag, 1995, vol. 900.

[86] A. MINÉ. *The Octagon Abstract Domain*, in "Higher-Order and Symbolic Computation", 2006, vol. 19, http://www.di.ens.fr/~mine/publi/article-mine-HOSC06.pdf.

[87] J.-P. QUEILLE, J. SIFAKIS. *Specification and Verification of Concurrent Systems in CESAR*, in "Proc. International Symposium on Programming", LNCS, Springer-Verlag, 1982, vol. 137, p. 337–351.

[88] P. RAMADGE, W. WONHAM. *Supervisory Control of a Class of Discrete Event Processes*, in "SIAM Journal on control and optimization", January 1987, vol. 25, n^{o} 1, p. 206–230.

[89] P. RAMADGE, W. WONHAM. *The Control of Discrete Event Systems*, in "Proceedings of the IEEE", 1989, vol. 77, n^{o} 1.

[90] P. SCHRAMMEL, B. JEANNET. *Extending Abstract Acceleration to Data-Flow Programs with Numerical Inputs*, in "ENTCS", 2010, vol. 267, n^{o} 1, p. 101–114.

[91] P. SCHRAMMEL, B. JEANNET. *Extending Abstract Acceleration to Data-Flow Programs with Numerical Inputs*, in "ENTCS", 2010, vol. 267, n^{o} 1, p. 101–114.

[92] P. TABUADA. *Verification and Control of Hybrid Systems - A Symbolic Approach*, Springer, 2009.

[93] R. WILHELM, J. ENGBLOM, A. ERMEDAHL, N. HOLSTI, S. THESING, D. B. WHALLEY, G. BERNAT, C. FERDINAND, R. HECKMANN, T. MITRA, F. MUELLER, I. PUAUT, P. P. PUSCHNER, J. STASCHULAT, P. STENSTRÖM. *The Determination of Worst-Case Execution Times — Overview of the Methods and Survey of Tools*, in "ACM Trans. Embedd. Comput. Syst.", April 2008, vol. 7, n^{o} 3.

[94] R. VON HANXLEDEN. *SyncCharts in C – A Proposal for Light-Weight Deterministic Concurrency*, in "International Conference on Embedded Software, EMSOFT'09", Grenoble, France, October 2009.