Activity Report 2013

Team SPADES

Sound Programming of Adaptive Dependable Embedded Systems

RESEARCH CENTER
Grenoble - Rhône-Alpes

THEME
Embedded and Real-time Systems

Table of contents

Team SPADES

Keywords: Component Programming, Embedded Systems, Fault Tolerance, Formal Methods, Real-time

Creation of the Team: 2013 January 01.

1. Members

Research Scientists

Jean-Bernard Stefani [Team leader, Senior Researcher, en détachement du Corps des Mines]
Pascal Fradet [Inria, Researcher, HdR]
Alain Girault [Inria, Senior Researcher, HdR]
Gregor Goessler [Inria, Researcher]
Bertrand Jeannet [Inria, Researcher, until Sep 2013]
Sophie Quinton [Inria, Researcher, from Nov 2013]

Faculty Member

Gwenaël Delaval [Associate professor, Université Joseph Fourier Grenoble I, until Apr 2013]

External Collaborators

Damien Pous [CNRS, Researcher, LIP Lyon]
Xavier Nicollin [Associate Professor, Grenoble INP]

PhD Students

Vagelis Bebelis [STMicroelectronics, CIFRE grant]
Dmitry Burlyaev [Université Joseph Fourier Grenoble I]
Yoann Geoffroy [Université Joseph Fourier Grenoble I, MESR grant, from Oct 2013]
Quentin Sabah [STMicroelectronics, CIFRE grant until Jul 2013]
Gideon Smeding [DIGITEO, DIGITEO grant, until Apr 2013]

Post-Doctoral Fellows

Barbara Petit [Inria, ANR REVER project, until Oct 2013]
Wei-Tsun Sun [Inria, Inria grant]

Visiting Scientists

Sidharta Andalam [TUM Create, Singapore, Mar 2013]
Ismail Assayad [Univ. Casablanca, Morroco, Sep 2013]
Christopher Shaver [UC Berkeley, USA, from Nov 2013 until Dec 2013]
Roopak Sinha [Univ. Auckland, New Zealand, from Jul 2013 until Aug 2013]
Eugene Yip [Univ. Auckland, New Zealand, Mar 2013]

Administrative Assistants

Diane Courtiol [Inria, until May 2013]
Helen Pouchot [Inria, from Jun 2013]

Other

Khanh Huu The Dam [Master, from Feb 2013 until Jun 2013]

2. Overall Objectives

2.1. Overall Objectives

The SPADES project-team aims at contributing to meet the challenge of designing and programming dependable embedded systems in an increasingly distributed and dynamic context. Specifically, by exploiting formal methods and techniques, SPADES aims to answer three key questions:

1. How to program open networked embedded systems as dynamic adaptive modular structures?
2. How to program reactive systems with real-time and resource constraints on multicore architectures?
3. How to program reliable, fault-tolerant embedded systems with different levels of criticality?

These questions above are not new, but answering them in the context of modern embedded systems, which are increasingly distributed, open and dynamic in nature [34], makes them more pressing and more difficult to address: the targeted system properties – dynamic modularity, time-predictability, energy efficiency, and fault-tolerance – are largely antagonistic (*e.g.*, having a highly dynamic software structure is at variance with ensuring that resource and behavioral constraints are met). Tackling these questions together is crucial to address this antagonism, and constitutes a key point of the SPADES research program.

A few remarks are in order:

- We consider these questions to be central in the construction of future embedded systems, dealing as they are with, roughly, software architecture and the provision of real-time and fault-tolerance guarantees. Building a safety-critical embedded system cannot avoid dealing with these three concerns.
- The three questions above are highly connected. For instance, composability along time, resource consumption and reliability dimensions are key to the success of a component-based approach to embedded systems construction.
- For us, "Programming" means any constructive process to build a running system. It can encompass traditional programming as well as high-level design or "model-based engineering" activities, provided that the latter are supported by effective compiling tools to produce a running system.
- We aim to provide semantically sound programming tools for embedded systems. This translates into an emphasis on formal methods and tools for the development of provably dependable systems.

3. Research Program

3.1. Introduction

The SPADES research program is organized around three main themes, *Components and contracts*, *Real-time multicore programming*, and *Language-based fault tolerance*, that seek to answer the three key questions identified in Section 2.1. We plan to do so by developing and/or building on programming languages and techniques based on formal methods and formal semantics (hence the use of *"sound programming"* in the project-team title). In particular, we seek to support design where correctness is obtained by construction, relying on proven tools and verified constructs, with programming languages and programming abstractions designed with verification in mind.

3.2. Components and contracts

Component-based construction has long been advocated as a key approach to the "correct-by-construction" design of complex embedded systems [71]. Witness component-based toolsets such as UC Berkeley's Ptolemy [58], Verimag's BIP [41], or the modular architecture frameworks used, for instance, in the automotive industry (AUTOSAR) [31]. For building large, complex systems, a key feature of component-based construction is the ability to associate with components a set of *contracts*, which can be understood as rich behavioral types that can be composed and verified to guarantee a component assemblage will meet desired properties. The goal in this theme is to study the formal foundations of the component-based construction of embedded systems, to develop component and contract theories dealing with real-time, reliability and fault-tolerance aspects of components, and to develop proof-assistant-based tools for the computer-aided design and verification of component-based systems.

Formal models for component-based design are an active area of research (see *e.g.*, [32], [33]). However, we are still missing a comprehensive formal model and its associated behavioral theory able to deal *at the same time* with different forms of composition, dynamic component structures, and quantitative constraints (such as timing, fault-tolerance, or energy consumption). Notions of contracts and interface theories have been proposed to support modular and compositional design of correct-by-construction embedded systems (see *e.g.*, [43], [44] and the references therein), but having a comprehensive theory of contracts that deals with all the above aspects is still an open question [76]. In particular, it is not clear how to accomodate different forms of composition, reliability and fault-tolerance aspects, or to deal with evolving component structures in a theory of contracts.

Dealing in the same component theory with heterogeneous forms of composition, different quantitative aspects, and dynamic configurations, requires to consider together the three elements that comprise a component model: behavior, structure and types. *Behavior* refers to behavioral (interaction and execution) models that characterize the behavior of components and component assemblages (*e.g.*, transition systems and their multiple variants – timed, stochastic, etc.). *Structure* refers to the organization of component assemblages or configurations, and the composition operators they involve. *Types* refer to properties or contracts that can be attached to components and component interfaces to facilitate separate development and ensure the correctness of component configurations with respect to certain properties. Taking into account dynamicity requires to establish an explicit link between behavior and structure, as well as to consider higher-order systems, both of which have a direct impact on types.

We plan to develop our component theory by progressing on two fronts: component calculi, and semantical framework. The work on typed component calculi aims to elicit process calculi that capture the main insights of component-based design and programming and that can serve as a bridge towards actual architecture description and programming language developments. The work on the semantical framework should, in the longer term, provide abstract mathematical models for the more operational and linguistic analysis afforded by component calculi. Our work on component theory will find its application in the development of a Coq-based toolchain for the certified design and construction of dependable embedded systems, which constitutes our third main objective for this axis.

3.3. Real-time multicore programming

Programming real-time systems (i.e. systems whose correct behavior depends on meeting timing constraints) requires appropriate languages (as exemplified by the family of synchronous languages [42]), but also the support of efficient scheduling policies, execution time and schedulability analyses to guarantee real-time constraints (*e.g.*, deadlines) while making the most effective use of available (processing, memory, or networking) resources. Schedulability analysis involves analyzing the worst-case behavior of real-time tasks under a given scheduling algorithm and is crucial to guarantee that time constraints are met in any possible execution of the system. Reactive programming and real-time scheduling and schedulability for multiprocessor systems are old subjects, but they are nowhere as mature as their uniprocessor counterparts, and still feature a number of open research questions [40], [53], in particular in relation with mixed criticality systems. The main goal in this theme is to address several of these open questions.

We intend to focus on two issues: multicriteria scheduling on multiprocessors, and schedulability analysis for real-time multiprocessor systems. Beyond real-time aspects, multiprocessor environments, and multicore ones in particular, are subject to several constraints *in conjunction*, typically involving real-time, reliability and energy-efficiency constraints, making the scheduling problem more complex both for the offline and the online cases. Schedulability analysis for multiprocessor systems, in particular for systems with mixed criticality tasks, is still very much an open research area.

Distributed reactive programming is rightly singled out as a major open issue in the recent, but heavily biased (it essentially ignores recent research in synchronous and dataflow programming), survey by Bainomugisha et al. [40]. For our part, we intend to focus on two questions: devising synchronous programming languages for distributed systems and precision-timed architectures, and devising dataflow languages for multiprocessors supporting dynamicity and parametricity while enjoying effective analyses for meeting real-time, resource and energy constraints in conjunction.

3.4. Language-based fault tolerance

Tolerating faults is a clear and present necessity in networked embedded systems. At the hardware level, modern multicore architectures are manufactured using inherently unreliable technologies [47], [65]. The evolution of embedded systems towards increasingly distributed architectures highlighted in the introductory section means that dealing with partial failures, as in Web-based distributed systems, becomes an important issue. While fault-tolerance is an old and much researched topic, several important questions remain open: automation of fault-tolerance provision, composable abstractions for fault-tolerance, fault diagnosis, and fault isolation.

The first question is related to the old question of "system structure for fault-tolerance" as originally discussed by Randell for software fault tolerance [84], and concerns in part our ability to clearly separate fault-tolerance aspects from the design and programming of purely "functional" aspects of an application. The classical arguments in favor of a clear separation of fault-tolerance concerns from application code revolve around reduced code and maintenance complexity [54]. The second question concerns the definition of appropriate abstractions for the modular construction of fault-tolerant embedded systems. The current set of techniques available for building such systems spans a wide range, including exception handling facilities, transaction management schemes, rollback/recovery schemes, and replication protocols. Unfortunately, these different techniques do not necessarily compose well – for instance, combining exception handling and transactions is non trivial, witness the flurry of recent work on the topic, see *e.g.*, [70] and the references therein –, they have no common semantical basis, and they suffer from limited programming language support. The third question concerns the identification of causes for faulty behavior in component-based assemblages. It is directly related to the much researched area of fault diagnosis, fault detection and isolation [72].

We intend to address these questions by leveraging programming language techniques (programming constructs, formal semantics, static analyses, program transformations) with the goal to achieve provable fault-tolerance, i.e. the construction of systems whose fault-tolerance can be formally ensured using verification tools and proof assistants. We aim in this axis to address some of the issues raised by the above open questions by using aspect-oriented programming techniques and program transformations to automate the inclusion of fault-tolerance in systems (software as well as hardware), by exploiting reversible programming models to investigate composable recovery abstractions, and by leveraging causality analyses to study fault-ascription in component-based systems. Compared to the huge literature on fault-tolerance in general, in particular in the systems area (see *e.g.*, [67] for an interesting but not so recent survey), we find by comparison much less work exploiting formal language techniques and tools to achieve or support fault-tolerance. The works reported in [46], [48], [50], [59], [73], [83], [89] provide a representative sample of recent such works.

A common theme in this axis is the use and exploitation of causality information. Causality, i.e., the logical dependence of an effect on a cause, has long been studied in disciplines such as philosophy [78], natural sciences, law [79], and statistics [81], but it has only recently emerged as an important focus of research in computer science. The analysis of logical causality has applications in many areas of computer science. For instance, tracking and analyzing logical causality between events in the execution of a concurrent system is

required to ensure reversibility [75], to allow the diagnosis of faults in a complex concurrent system [68], or to enforce accountability [74], that is, designing systems in such a way that it can be determined without ambiguity whether a required safety or security property has been violated, and why. More generally, the goal of fault-tolerance can be understood as being to prevent certain causal chains from occurring by designing systems such that each causal chain either has its premises outside of the fault model (*e.g.*, by introducing redundancy [67]), or is broken (*e.g.*, by limiting fault propagation [86]).

4. Application Domains

4.1. Industrial Applications

Our applications are in the embedded system area, typically: transportation, energy production, robotics, 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 programming 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, we are looking to propose 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 [1]) and execution platforms (OS such as VXWORKS, QNX, real-time versions of LINUX ...) start 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 [2] (based on LUSTRE), CONTROLBUILD and RT-BUILDER (based on SIGNAL) from GEENSYS [3] (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 dynamic data-flow models of computation for streaming applications, dedicated to high definition video applications for their new STHORM manycore chip.

5. Software and Platforms

5.1. Implementations of Synchronous Programs

Participant: Alain Girault.

[1] http://www.dspaceinc.com
[2] http://www.esterel-technologies.com
[3] http://www.geensoft.com

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 SYNDEX [4]. Our first scheduling heuristic produces static multiprocessor schedules tolerant to a specified number of processor and communication link failures [62]. The basic principles upon which we rely to make the schedules fault tolerant are, on the one hand, the active replication of the operations [63], 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) [64]. Our second scheduling heuristic is multi-criteria: it produces a static multiprocessor schedule such that the reliability is maximized, the power consumption is minimized, and the execution time is minimized [12][4] [37], [38]. Our results on fault tolerance are summarized in a web page [5].

5.2. Apron and BddApron Libraries

Participant: Bertrand Jeannet.

5.2.1. Principles

The APRON library [6] 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 provides:

- 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.

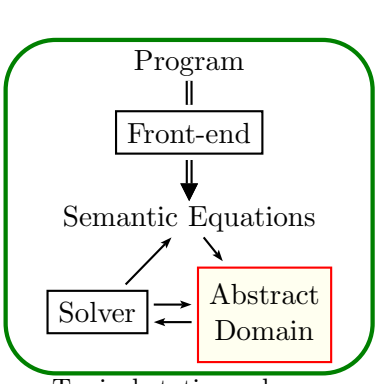
For the client static analysis community, the benefits are a unified, higher-level interface, which allows experimenting, comparing, and combining abstract domains.

The BDDAPRON library [7] 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, bit vectors) and numerical variables (integers, rationals, floating-point numbers). It first allows the manipulation of expressions that freely mix, using BDDs and MTBDDs, finite-type and numerical APRON expressions and conditions. It then provides abstract domains that combine BDDs and APRON abstract values for representing invariants holding on both finite-type variables and numerical variables.

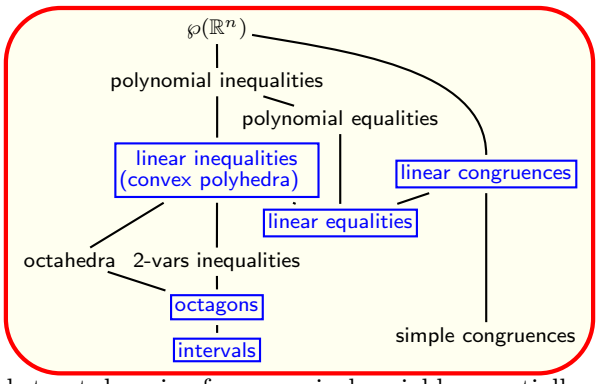
5.2.2. Implementation and Distribution

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). It is currently packaged as a REDHAT and DEBIAN package.

[4] http://www-rocq.inria.fr/syndex
[5] http://pop-art.inrialpes.fr/~girault/Projets/FT
[6] http://apron.cri.ensmp.fr/library/
[7] http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/index.html



Typical static analyser



Some abstract domains for numerical variables, partially ordered w.r.t. their expressiveness

Figure 1. Typical static analyser and examples of abstract domains

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.

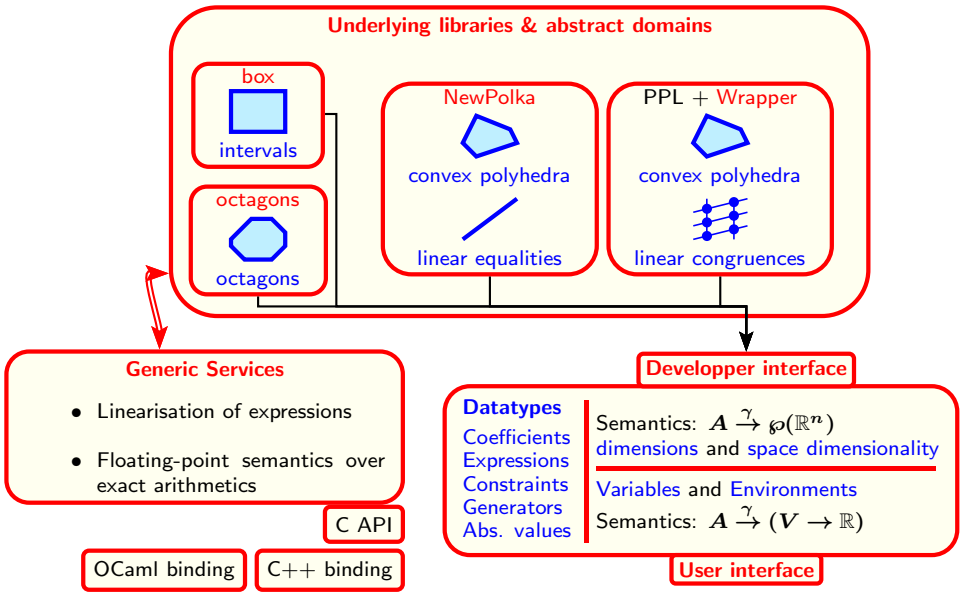


Figure 2. Organisation of the APRON *library*

5.3. ReaVer

Participant: Bertrand Jeannet.

REAVER (REActive VERifier [8]) is a tool framework for the safety verification of discrete and hybrid systems specified by logico-numerical data-flow languages, like LUSTRE, LUCID SYNCHRONE or ZELUS. It provides time-unbounded analysis based on abstract interpretation techniques.

It features partitioning techniques and several logico-numerical analysis methods based on Kleene iteration with widening and descending iterations, abstract acceleration, max-strategy iteration, and relational abstractions; logico-numerical product and power domains (based on the APRON and BddApron domain libraries) with convex polyhedra, octagons, intervals, and template polyhedra; and front-ends for the hybrid NBAC format, LUSTRE via lus2nbac, and ZELUS/LUCID SYNCHRONE. Compared to NBAC, it is connected to higher-level, more recent synchronous and hybrid languages, and provides many more options regarding analysis techniques.

It has been used for several experimental comparisons published in papers. It integrates all the methods developed by Peter Schrammel in his PhD.

5.4. Prototypes

5.4.1. Logical Causality

Participant: Gregor Goessler.

We are developing LOCA, a prototype tool written in Scala that implements the analysis of logical causality described in 6.1.1. 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.

5.4.2. Cosyma

Participant: Gregor Goessler.

We have developed COSYMA, a tool for automatic controller synthesis for incrementally stable switched systems based on multi-scale discrete abstractions. The tool accepts a description of a switched system represented by a set of differential equations and the sampling parameters used to define an approximation of the state-space on which discrete abstractions are computed. The tool generates a controller — if it exists — for the system that enforces a given safety or time-bounded reachability specification.

5.4.3. Automatic Controller Generation

Participant: Alain Girault.

We have developed a software tool chain to allow the specification of models, 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 [9] and Mode Automata [10]. It is the result of a collaboration with Emil Dumitrescu (INSA Lyon) and Eric Rutten from the CTRL-A Inria 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.

5.4.4. The Interproc family of static analyzers

Participant: Bertrand Jeannet [contact person].

[8] http://members.ktvam.at/schrammel/research/reaver
[9] http://www.irisa.fr/vertecs/Logiciels/sigali.html
[10] http://www-verimag.imag.fr

These analyzers and libraries are of general use for people working in the static analysis and abstract interpretation community.

- FIXPOINT[11]: 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 [66].
- INTERPROC[12]: 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 [13].
- CONCURINTERPROC extends INTERPROC with concurrency, for the analysis of multithreaded programs interacting via shared global variables. It is also deployed through a web-interface [14].
- PINTERPROC extends INTERPROC with pointers to local variables. It is also deployed through a web-interface [15].

5.4.5. The SIAAM virtual machine

Participants: Quentin Sabah, Jean-Bernard Stefani [contact person].

The SIAAM abstract machine is an object-based realization of the Actor model of concurrent computation. Actors can exchange arbitrary object graphs in messages while still enjoying a strong isolation property. It guarantees that each actor can only directly access objects in its own local heap, and that information between actors can only flow via message exchange [10]. The SIAAM machine has been implemented for Java as a modified Jikes virtual machine. The resulting SIAAM software comprises:

- A modified Jikes RVM that implements actors and actor isolation as specified by the SIAAM machine.
- A set of static analyses build using the Soot Java optimization framework for optimizing the execution of the SIAAM/Jikes virtual machine, and for helping programmers diagnose potential performance issues.
- A formal proof using the Coq proof assistant of the SIAAM isolation property.

6. New Results

6.1. Components and Contracts

Participants: Gregor Goessler, Quentin Sabah, Jean-Bernard Stefani.

6.1.1. Analysis of logical causality

The failure of one component may entail a cascade of failures in other components; several components may also fail independently. In such cases, elucidating the exact scenario that led to the failure is a complex and tedious task that requires significant expertise.

[11] http://http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint
[12] http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc
[13] http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
[14] http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi
[15] http://pop-art.inrialpes.fr/interproc/pinterprocweb.cgi

The notion of causality *(did an event e cause an event e'?)* has been studied in many disciplines, including philosophy, logic, statistics, and law. The definitions of causality studied in these disciplines usually amount to variants of the counterfactual test "e is a cause of e' if both e and e' have occurred, and in a world that is as close as possible to the actual world but where e does not occur, e' does not occur either". Surprisingly, the study of logical causality has so far received little attention in computer science, with the notable exception of [69] and its instantiations. However, this approach relies on a causal model that may not be known, for instance in presence of black-box components.

Improving on previous results, we have proposed in [21] an approach to enhance the fault diagnosis in black-box component-based systems, in which only events on component interfaces are observable. For such systems, we have described a causality analysis framework that helps us establish the causal relationship between component failures and system failures, given an observed system execution trace. The analysis is based on a formalization of counterfactual reasoning, and applicable to real-time systems. We have illustrated the analysis with a case study from the medical device domain.

In [5] we have proposed a formal framework for reasoning about causality, and blaming system-level failures on the component(s) that caused them. The framework is general in the sense that it applies to many different models of computation and communication (MoC), such as synchronous and asynchronous computation, and communication by messages or shared variables. We are currently instantiating the framework to specific MoC, in particular, to timed automata, and developing a refinement of our original approach that reduces the number of false positives.

6.1.2. Supporting isolation for actors in shared memory

The actor model of concurrency, as supported *e.g.*, by the Erlang programming language, is an appealing programming model for the construction of concurrent and distributed systems, and multicore programming in particular. Although much work has taken place in particular during the past ten years on efficient implementations of the actor model, the design space is far from being completely understood.

As part of Quentin Sabah's thesis [10], we have developed a variant of the actor model that, in contrast to previous works, ensures a strict isolation between actors while imposing no restriction on the form of data exchanged in messages. We have formally specified an abstract machine, called SIAAM (see Sec.5.4.5), for an extension of the Java language with our actor model, and implemented it as a modified Jikes virtual machine, a state of the art Java virtual machine. A combination of points-to and live variable analyses has been implemented using the Soot framework, that can be used to remove unnecessary read and write checks for isolation. A diagnosis tool built on top of the analyses helps programmers to pinpoint potential problems (exceptions raised indicating a potential violation of isolation). We have shown with artificial and small applicative benchmarks that, using our analyses to improve performance, our implementation is reasonably efficient and imposes low overhead for the benefit of strict isolation.

In addition, we have developed a Coq proof of the isolation property enforced by SIAAM, namely that no information between actors can take place outside of message exchanges, despite the presence of a shared heap between actors.

6.2. Real-Time multicore programming

Participants: Vagelis Bebelis, Gwenaël Delaval, Pascal Fradet, Alain Girault, Gregor Goessler, Bertrand Jeannet, Gideon Smeding, Jean-Bernard Stefani.

6.2.1. A time predictable programming language for multicores

Time predictability (PRET) is a topic that emerged in 2007 as a solution to the ever increasing unpredictability of today's embedded processors, which results from features such as multi-level caches or deep pipelines [57]. For many real-time systems, it is mandatory to compute a strict bound on the program's execution time. Yet, in general, computing a tight bound is extremely difficult [90]. The rationale of PRET is to simplify both the programming language and the execution platform to allow more precise execution times to be easily computed [39].

Following our past results on the PRET-C programming language [35], we have proposed a time predictable synchronous programming language for multicores, called FOREC. It extends C with a small set of ESTEREL-like synchronous primitives to express concurrency, interaction with the environment, looping, and a synchronization barrier [22] (like the `pause` statement in ESTEREL). FOREC threads communicate with each other via shared variables, the values of which are combined at the end of each tick to maintain deterministic execution. FOREC is compiled into threads that are then statically scheduled for a target multicore chip. Our WCET analysis takes into account the access to the shared TDMA bus and the necessary administration for the shared variables. We achieve a very precise WCET (the over-approximation being less than 2%) thanks to a reachable space exploration of the threads' states.

This work has been conducted within the RIPPES associated team.

6.2.2. WCET analysis

Our past work on the WCET analysis of PRET-C programs has led us to design static analyses, for instance to prune unfeasible paths in the control flow graph [36]. In 2013, we have worked on how to take into account direct mapped instruction caches in WCET analysis. Instruction caches are essential to address if one wants to analyze large embedded programs. Our cache analysis technique offers the same precision as the most precise techniques [80], while improving analysis time by up to 240 times. This improvement is achieved by analyzing individual blocks of the control flow graph separately, and by proposing a tailored abstract domain to represent efficiently the cache state [14], [25]. In contrast with previous abstract analysis methods [88], [85], our analysis is able to offer the same precision as the concrete approaches [80].

6.2.3. Tradeoff exploration between reliability, power consumption, and execution time

For autonomous critical real-time embedded systems (*e.g.*, satellites), 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 ready list 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 tri-criteria 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 [37], [38]. TSH implements a ready list scheduling heuristics, and we have formulated a new multi-criteria cost function such that we are able to prove rigorously that the static schedules we generate meet both the reliability constraint and the power consumption constraint [12].

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 ILP. Secondly, we have compared TSH with the ECS heuristic (Energy-Conscious Scheduling [77]); 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.

6.2.4. 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 started 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 [49]). 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. It 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).

6.2.5. Distribution of synchronous programs under real-time constraints

The goal of Gideon Smeding's PhD thesis [11] was to propose a quasi-synchronous framework encompassing constraints on the relative speed of clocks, together with a formalism for reasoning about clock-dependent properties within the model. This framework should provide a seamless link between synchronous models and their asynchronous implementation.

The quasi-synchronous approach developed in [11] considers independently clocked, synchronous components that interact via communication-by-sampling or FIFO channels. We have defined relative drift bounds on pairs of recurring events such as clock ticks or the arrival of a message. Drift bounds express constraints on the stability of clocks, *e.g.*, at least two ticks of one per three consecutive ticks of the other. We can thus move from total synchrony, where all clocks tick simultaneously, to global asynchrony by relaxing the drift bounds. As constraints are more relaxed, behavior diverges more and more from synchronous system behavior. In many systems, such as distributed control systems, occasional deviations of input and output signals of the controller from their behavior in the synchronous model may be acceptable as long as the frequency of such deviations is bounded. The approach of [11] takes as inputs a program written in a Lustre-like language extended with asynchronous communication by sampling, application requirements on the distribution in the form of weakly-hard constraints [45] bounding *e.g.*, the tolerated loss of data tokens, and platform assertions (*e.g.*, relative clock speeds, available communication resources), and verifies whether the program meets the requirements under the platform assertions.

6.2.6. Analysis and scheduling of parametric dataflow 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 or 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.

Last year, we have introduced the *schedulable parametric data-flow (SPDF)* MoC for dynamic streaming applications [60]. SPDF extends the standard dataflow model by allowing rates to be parametric. SPDF was designed to be statically analyzable while retaining sufficient expressive power.

Following the same lines, we have recently proposed the *Boolean Parametric Data Flow (BPDF)* MoC which combines integer parameters (to express dynamic rates) and boolean parameters (to express the activation and deactivation of communication channels) [15], [26], [24]. High dynamism is provided by integer parameters which can change at each basic iteration and boolean parameters which can change even within the iteration. We have presented static analyses which ensure statically the liveness and the boundedness of BDPF graphs. Our case studies are video decoders for high definition video streaming such as VC-1.

We have proposed a generic and flexible framework to generate parallel ASAP schedules targeted to the new STHORM many-core platform designed by STMicroelectronics [29], [23]. The parametric dataflow graph is associated with generic or user-defined specific constraints aimed at minimizing, timing, buffer sizes, power consumption, or other criteria. The scheduling algorithm executes with minimal overhead and can be adapted to different scheduling policies just by changing some constraints. The safety of both the dataflow graph and constraints can be checked statically and all schedules are guaranteed to be bounded and deadlock free. This parallel scheduling framework has been developed for a parametric MoC without booleans. We are now focusing on extending it to BPDF applications.

This research is the central topic of Vagelis Bebelis' PhD thesis. It is conducted in collaboration with STMicroelectronics.

6.2.7. Abstract Acceleration of general linear loops

We have investigated abstract acceleration techniques for computing loop invariants for numerical programs with linear assignments and conditionals. Whereas abstract interpretation techniques typically over-approximate the set of reachable states iteratively, abstract acceleration captures the effect of the loop with a single, non-iterative transfer function applied to the initial states at the loop head.

In contrast to previous acceleration techniques, our approach applies to any linear loop without restrictions. Its novelty lies in the use of the Jordan normal form decomposition of the loop body to derive symbolic expressions for the entries of the matrix modeling the effect of $n >= 0$ iterations of the loop. The entries of such a matrix depend on n through complex polynomial, exponential and trigonometric functions. Therefore, we introduced an abstract domain for matrices that captures the linear inequality relations between these complex expressions. This results in an abstract matrix for describing the fixpoint semantics of the loop. We also developed a technique to take into account the guard of the loop by bounding the number of loop iterations, which relies again on the Jordan normal form decomposition.

Our approach integrates smoothly into standard abstract interpreters and can handle programs with nested loops and loops containing conditional branches. We evaluate it over small but complex loops that are commonly found in control software, comparing it with other tools for computing linear loop invariants. The loops in our benchmarks typically exhibit polynomial, exponential and oscillatory behaviors that present challenges to existing approaches, that are either too unprecise (classical abstract interpretation) or limited to a restricted class of loops (*e.g.*, translation with resets in the case of abstract acceleration, or stable loops, in the sense of control theory, for ellipsoid methods). Our approach finds non-trivial invariants to prove useful bounds on the values of variables for such loops, clearly outperforming the existing approaches in terms of precision while exhibiting good performance.

A paper presenting this technique has been accepted to POPL'2014. An extended version has been published in arXiv [30].

6.2.8. 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.*, [87] 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 [82]. 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 [61].

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. We have been exploring two approaches to overcome this state-space explosion.

In [52], we have proposed a technique for the synthesis of safety controllers for switched systems using multi-scale abstractions that allow us to deal with fast switching while keeping the number of states in the abstraction at a reasonable level. 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 have implemented these results in the tool COSYMA (COntroller SYnthesis using Multi-scale Abstractions, see Sec. 5.4.2) [20]. The tool accepts a description of a switched system represented by a set of differential equations and the sampling parameters used to define an approximation of the state-space on which discrete abstractions are computed. The tool generates a controller — if it exists — for the system that enforces a given safety or time-bounded reachability specification.

In [19], we have presented an approach using mode sequences of given length as symbolic states for our abstractions. We have shown that the resulting symbolic models are approximately bisimilar to the original switched system and that an arbitrary precision can be achieved by considering sufficiently long mode sequences. The advantage of this approach over existing ones is double: first, the transition relation of the symbolic model admits a very compact representation under the form of a shift operator; second, our approach does not use lattices over the state-space and can potentially be used for higher dimensional systems. We have provided a theoretical comparison with the lattice-based approach and presented a simple criterion enabling to choose the most appropriate approach for a given switched system. We have applied the approach to a model of road traffic for which we have synthesized a schedule for the coordination of traffic lights under constraints of safety and fairness.

6.3. Language Based Fault-Tolerance

Participants: Dmitry Burlyaev, Pascal Fradet, Alain Girault, Jean-Bernard Stefani.

6.3.1. Automatic Transformations for Fault tolerant Circuits

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 automatic transformations to ensure fault-tolerance properties in digital circuits. To this aim, we consider program transformations for hardware description languages (HDL). We have designed a simple hardware description language inspired from LUSTRE and Lucid Synchrone. It is a core functional language manipulating synchronous boolean streams. We consider both single-event upsets (SEU) and single-event transients (SET) and all fault models of the form "at most 1 SEU or SET within n clock signals". The language's semantics as well as fault modes have been formalized in Coq and many basic (library) properties have been shown on that language.

We have expressed several variants of triple modular redundancy (TMR) as program transformations. We have proposed a verification-based approach to minimize the number of voters in TMR [16]. Our technique guarantees that the resulting circuit *(i)* is fault tolerant to the soft-errors defined by the fault model and *(ii)* is functionally equivalent to the initial one. Our approach operates at the logic level and takes into account the input and output interface specifications of the circuit. Its implementation makes use of graph traversal algorithms, fixed-point iterations, and BDDs. Experimental results on the ITC'99 benchmark suite indicate that our method significantly decreases the number of inserted voters which entails a hardware reduction of up to 55% and a clock frequency increase of up to 35% compared to full TMR. We address scalability issues arising from formal verification with approximations and assess their efficiency and precision.

We are currently studying the definition of other fault-tolerant techniques (*e.g.*, time redundancy, mixed time/space redundancy) as program transformations. We are also considering the use of the Coq proof assistant to certify that the transformations make the programs fault tolerant *w.r.t.* specific fault models. Our long term goal is to design an aspect-like language allowing users to specify and tune a wide range of fault tolerance techniques, while ensuring that the corresponding transformations ensure well-defined fault-tolerance properties. The advantage would be to produce fault-tolerant circuits by specifying fault-tolerant properties/strategies separately from their functional specifications.

6.3.2. Concurrent flexible reversibility

In the recent years, we have been investigating reversible concurrent computation, and investigated various reversible concurrent programming models, with the hope that reversibility can shed some light on the common semantic features underlying various forms of fault recovery techniques (including, exceptions, transactions, and checkpoint/rollback schemes).

As part of this research program, we have devised a reversible variant of the higher-order π-calculus, equipped with an imperative rollback operation that allows a concurrent program to be rolled back to a past execution state, and a primitive form of compensation to control (forward execution) after a rollback operation [18]. We have shown that these two extensions provide very powerful primitives for programming different forms of rollback/compensation schemes. We have shown in particular that they are powerful enough to provide a faithful encoding of a notion of communicating transaction proposed in the literature. We have started the development of a behavioral theory for this $\mathsf{croll}\pi$ calculus, and proved in particular a context lemma, similar to that of the π-calculus, although the reversible machinery makes its proof more involved.

This work was done in collaboration with Inria teams FOCUS in Bologna and CELTIQUE in Rennes, and as part of the ANR REVER project.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

- With ST Microelectronics: CIFRE contract for the PhD of Vagelis Bebelis. This work is described in Section 6.2.6.
- With ARGOSIM SA: "Study and transfer contract" for the development by Bertrand Jeannet and the cession to ARGOSIM of the PolyCart library. PolyCart is a library for the manipulation of cartesian products of polyhedra and intervals.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR Projects

8.1.1.1. PiCoq (ANR project)

Participants: Barbara Petit, Jean-Bernard Stefani.

The goal of the PiCoq project is to develop an environment for the formal verification of properties of distributed, component-based programs. The project's approach lies at the interface between two research areas: concurrency theory and proof assistants. Achieving this goal relies on three scientific advances, which the project intends to address:

- Finding mathematical frameworks that ease modular reasoning about concurrent and distributed systems: due to their large size and complex interactions, distributed systems cannot be analysed in a global way. They have to be decomposed into modular components, whose individual behaviour can be understood.
- Improving existing proof techniques for distributed/modular systems: while behavioural theories of first-order concurrent languages are well understood, this is not the case for higher-order ones. We also need to generalise well-known modular techniques that have been developed for first-order languages to facilitate formalisation in a proof assistant, where source code redundancies should be avoided.
- Defining core calculi that both reflect concrete practice in distributed component programming and enjoy nice properties *w.r.t.* behavioural equivalences.

The project partners include Inria (CELTIQUE and SPADES teams), LIP (PLUME team), and Université de Savoie. The project runs from November 2010 to October 2014.

8.1.1.2. REVER (ANR project)

Participants: Barbara Petit, Jean-Bernard Stefani.

The REVER project aims to develop semantically well-founded and composable abstractions for dependable distributed computing on the basis of a reversible programming model, where reversibility means the ability to undo any program execution and to revert it to a state consistent with the past execution. The critical assumption behind REVER is that by combining reversibility with notions of compensation and modularity, one can develop systematic and composable abstractions for dependable programming.

The REVER work program is articulated around three major objectives:

- To investigate the semantics of reversible concurrent processes.
- To study the combination of reversibility with notions of compensation, isolation and modularity in a concurrent and distributed setting.
- To investigate how to support these features in a practical (typically, object-oriented and functional) programming language design.

The project partners are Inria (FOCUS and SPADES teams), Université de Paris VII (PPS laboratory), and CEA (List laboratory). The project runs from December 2011 to November 2015.

8.2. International Initiatives

8.2.1. Inria Associate Teams

8.2.1.1. RIPPES

Title: RIgorous Programming of Predictable Embedded Systems

Inria principal investigator: Alain Girault

International Partner (Institution - Laboratory - Researcher):

University of California Berkeley (USA) – EECS Department, PTOLEMY group – Prof. Edward Lee.

University of Auckland (New Zealand) – ECE Department – Prof. Partha Roop.

Duration: January 2013 – December 2015

See also: https://wiki.inria.fr/rippes/Main_Page

The RIPPES associated team gathers the SPADES team from Inria Grenoble, the Ptolemy group from UC Berkeley (EECS Department), and the Embedded Systems Research group from U. of Auckland (ECE Department). The planned research seeks to reconcile two contradictory objectives of embedded systems, more predictability and more adaptivity. We propose to address these issues by exploring two complementary research directions: (1) by starting from a classical concurrent C or Java programming language and enhancing it to provide more predictability, and (2) by starting from a very predictable model of computation (SDF) and enhancing it to provide more adaptivity.

8.3. International Research Visitors

8.3.1. Visits of International Scientists

- January and February 2013: Ismail Assayad (Ass. Prof. U. Casablanca) visited Inria Grenoble to work on multi-criteria optimisation and scheduling for embedded system.
- March 2013: Eugene Yip (PhD student, U. Auckland) visited Inria Grenoble to work on the semantics of the FOREC PRET programming language (RIPPES associated team).
- March 2013: Hokeun Kim (PhD student, UC Berkeley) visited Inria Grenoble to work on the RIPPES associated team.

- March 2013: Partha Roop (Senior Lecturer, U. Auckland) visited Inria Grenoble to work on the FOREC PRET programming language (RIPPES associated team).
- July 2013: Eugene Yip (PhD student, U. Auckland) visited Inria Grenoble to work on the semantics of the FOREC PRET programming language (RIPPES associated team).
- July 2013: Matthew Kuo (PhD student, U. Auckland) visited Inria Grenoble to work on tickpad memories for PRET programs (RIPPES associated team).
- December 2013: Chris Shaver (PhD student, UC Berkeley) visited Inria Grenoble to work on parametric data-flow models of computation (RIPPES associated team).

8.3.2. Visits to International Teams

- Vagelis Bebelis visited the University of California Berkeley (USA) in October 2013 to work on a parametric dataflow models of computation and on its implementation within the Ptolemy II framework.

8.3.3. Inria International Partners

8.3.3.1. Informal International Partners

We have a long lasting informal collaboration with Prof. Ivan Lanese (U. Bologna, Italy) on component programming and reversability. He visits the team regularly.

We have a long lasting informal collaboration with Prof. Ismail Assayad (U. Casablanca, Morocco) and Prof. Hamoudi Kalla (U. Batna, Algeria) on fault-tolerant embedded systems, multi-criteria optimization, reliability, and power consumption. They both visit the team regularly.

9. Dissemination

9.1. Scientific Animation

- Pascal Fradet served in the program committees of MODULARITY 2014 and of JFLA 2013 and JFLA 2014 (*J*ournées Francophones des Langages Applicatifs).
- Alain Girault served in the program committees of the international conferences DAC 2013 and MSR 2013.
- Gregor Goessler served in the program committees of the international conferences Component-Based Software Engineering (CBSE) 2013 and Design, Automation, and Test in Europe (DATE) 2014 and the workshop Hybrid Systems and Biology (HSB) 2013.
- Jean-Bernard Stefani is the current Chair of IFIP Working Group WG6.1, that sponsors the international conference series DAIS, FORTE, ICTSS, and Middleware. He is the current chair of the FORTE Steering Committee and a member of the DISCOTEC joint international conference (hosting Coordination, DAIS and FORTE).

9.2. Teaching - Supervision - Juries

9.2.1. Supervision

PhD: Quentin Sabah, “Simple Isolation for An Abstract Actor Machine”, Grenoble University, 4/12/2013, advised by Jean-Bernard Stefani.

PhD: Gideon Smeding, “Verification of Weakly-Hard Requirements on Quasi-Synchronous Systems”, Grenoble University, 19/12/2013, co-advised by Gregor Goessler and Joseph Sifakis.

PhD in progress: Vagelis Bebelis, “Advanced dataflow programming for embedded systems”, Grenoble University, since 12/2011, co-advised by Pascal Fradet and Alain Girault.

PhD in progress: Dmitry Burlyaev, “Specification and synthesis of fault-tolerant circuits”, Grenoble University, since 12/2011, co-advised by Pascal Fradet and Alain Girault.

PhD in progress: Yoann Geoffroy, “Towards a general causality analysis framework”, Grenoble University, since 10/2013, co-advised by Gregor Goessler and Daniel Le Métayer (Privatics INRIA team).

9.2.2. Juries

- Alain Girault was referee for the PhD thesis of Hervé Yviquel (University of Rennes 1) and for the PhD thesis of Léonard Gérard (University of Orsay).
- Jean-Bernard Stefani was referee for the PhD thesis of Cédric Pasteur (University of Paris VI).

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, pp. 1–43

[2] E. BRUNETON, T. COUPAYE, M. LECLERCQ, V. QUEMA, J.-B. STEFANI. *The Fractal Component Model and its Support in Java*, in "Software - Practice and Experience", 2006, vol. 36, n^{o} 11-12

[3] S. DJOKO DJOKO, R. DOUENCE, P. FRADET. *Aspects preserving properties*, in "Science of Computer Programming", 2012, vol. 77, n^{o} 3, pp. 393-422

[4] 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, pp. 241–254, Research report Inria 6319, http://hal.inria.fr/inria-00177117

[5] G. GOESSLER, D. LE MÉTAYER. *A General Trace-Based Framework of Logical Causality*, in "FACS - 10th International Symposium on Formal Aspects of Component Software - 2013", Nanchang, Chine, 2013, http://hal.inria.fr/hal-00924048

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

[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] S. LENGLET, A. SCHMITT, J.-B. STEFANI. *Characterizing Contextual Equivalence in Calculi with Passivation*, in "Inf. Comput.", 2011, vol. 209, n^{o} 11, pp. 1390-1433

Publications of the year

Doctoral Dissertations and Habilitation Theses

[10] Q. SABAH. , *SIAAM: Isolation dynamique pour une machine abstraite à base d'acteurs*, Université de Grenoble, December 2013, http://hal.inria.fr/tel-00933072

[11] G. SMEDING. , *Vérification de propriétés faiblement dures des systèmes quasi-synchrones*, Université de Grenoble, December 2013, http://hal.inria.fr/tel-00925626

Articles in International Peer-Reviewed Journals

[12] I. ASSAYAD, A. GIRAULT, H. KALLA. *Tradeoff exploration between reliability, power consumption, and execution time for embedded systems*, in "Software Tools for Technology Transfer (STTT)", June 2013, vol. 15, n^{o} 3, pp. 229-245 [*DOI :* 10.1007/s10009-012-0263-9], http://hal.inria.fr/hal-00923926

[13] A. BENOIT, F. DUFOSSÉ, A. GIRAULT, Y. ROBERT. *Reliability and performance optimization of pipelined real-time systems*, in "Journal of Parallel and Distributed Computing", 2013, vol. 73, n^{o} 6, pp. 851-865 [*DOI :* 10.1016/j.jpdc.2013.02.009], http://hal.inria.fr/hal-00926123

International Conferences with Proceedings

[14] S. ANDALAM, R. SINHA, P. ROOP, A. GIRAULT, J. REINEKE. *Precise timing analysis for direct-mapped caches*, in "Design Automaton Conference, DAC", Austin, TX, United States, ACM, June 2013 [*DOI :* 10.1145/2463209.2488917], http://hal.inria.fr/hal-00842368

[15] V. BEBELIS, P. FRADET, A. GIRAULT, B. LAVIGUEUR. *BPDF: A Statically Analyzable Dataflow Model with Integer and Boolean Parameters*, in "International Conference on Embedded Software, EMSOFT'13", Montreal, Canada, ACM, September 2013, http://hal.inria.fr/hal-00842421

[16] D. BURLYAEV, P. FRADET, A. GIRAULT. *Verification-guided Voter Minimization in Triple-Modular Redundant Circuits*, in "Design, Automation and Test in Europe Conference, DATE'14", Dresden, Germany, November 2013, http://hal.inria.fr/hal-00911768

[17] G. GÖSSLER, D. LE MÉTAYER. *A General Trace-Based Framework of Logical Causality*, in "FACS - 10th International Symposium on Formal Aspects of Component Software - 2013", Nanchang, China, 2013, http://hal.inria.fr/hal-00924048

[18] I. LANESE, M. LIENHARDT, C. MEZZINA, A. SCHMITT, J.-B. STEFANI. *Concurrent Flexible Reversibility*, in "22nd European Symposium on Programming, ESOP 2013", Rome, Italy, M. FELLEISEN, P. GARDNER (editors), Lecture Notes in Computer Science (LNCS), Springer, March 2013, vol. 7792, pp. 370-390 [*DOI :* 10.1007/978-3-642-37036-6_21], http://hal.inria.fr/hal-00811629

[19] E. LE CORRONC, A. GIRARD, G. GÖSSLER. *Mode Sequences as Symbolic States in Abstractions of Incrementally Stable Switched Systems*, in "CDC - 52nd Conference on Decision and Control - 2013", Florence, Italy, IEEE, December 2013, http://hal.inria.fr/hal-00924815

[20] S. MOUELHI, A. GIRARD, G. GOESSLER. *CoSyMA: a tool for controller synthesis using multi-scale abstractions*, in "HSCC'13 - 16th International Conference on Hybrid systems: computation and control",

Philadelphie, United States, ACM, 2013, pp. 83-88 [*DOI :* 10.1145/2461328.2461343], http://hal.inria.fr/hal-00839613

[21] S. WANG, A. AYOUB, B. KIM, G. GÖSSLER, O. SOKOLSKY, I. LEE. *A Causality Analysis Framework for Component-based Real-time Systems*, in "RV - 4th International Conference on Runtime Verification - 2013", Rennes, France, A. LEGAY, S. BENSALEM (editors), Springer, 2013, vol. 8174, pp. 285-303, http://hal.inria.fr/hal-00919081

[22] E. YIP, P. ROOP, M. BIGLARI-ABHARI, A. GIRAULT. *Programming and Timing Analysis of Parallel Programs on Multicores*, in "International Conference on Application of Concurrency to System Design, ACSD'13", Barcelona, Spain, IEEE, July 2013, pp. 167–176, http://hal.inria.fr/hal-00842402

Conferences without Proceedings

[23] V. BEBELIS, P. FRADET, A. GIRAULT, B. LAVIGUEUR. *A Framework to Schedule Parametric Dataflow Applications on Many-Core Platforms*, in "CPC 2013, 17th Workshop on Compilers for Parallel Computing", Lyon, France, July 2013, http://hal.inria.fr/hal-00923670

[24] V. BEBELIS, P. FRADET, A. GIRAULT, B. LAVIGUEUR. *BPDF: A Statically Analyzable Dataflow Model with Integer and Boolean Parameters*, in "The Tenth Biennial Ptolemy Miniconference", Berkeley, United States, November 2013, http://hal.inria.fr/hal-00923672

Research Reports

[25] S. ANDALAM, R. SINHA, P. S. ROOP, A. GIRAULT, J. REINEKE. , *Precise Modelling of Instruction Cache Behaviour*, Inria, January 2013, n^{o} RR-8214, 62 p. , http://hal.inria.fr/hal-00781566

[26] V. BEBELIS, P. FRADET, A. GIRAULT, B. LAVIGUEUR. , *BPDF: Boolean Parametric Data Flow*, Inria, July 2013, n^{o} RR-8333, 21 p. , http://hal.inria.fr/hal-00846645

[27] Z. E. BHATTI, R. SINHA, P. ROOP. , *Unified Functional Safety Assessment of Industrial Automation Systems*, Inria, September 2013, n^{o} RR-8357, http://hal.inria.fr/hal-00858218

[28] G. GÖSSLER, D. LE MÉTAYER. , *A General Trace-Based Framework of Logical Causality*, Inria, October 2013, n^{o} RR-8378, http://hal.inria.fr/hal-00873665

Other Publications

[29] V. BEBELIS, P. FRADET, A. GIRAULT, B. LAVIGUEUR. , *A Flexible Approach for Scheduling Parametric Data Flow Applications on Sthorm*, March 2013, Poster at Workshop on Platform 2012 / STHORM embedded many-core acceleration, DATE 2013, http://hal.inria.fr/hal-00923669

[30] B. JEANNET, P. SCHRAMMEL, S. SANKARANARAYANAN. , *Abstract Acceleration of General Linear Loops*, November 2013, Extended version of the POPL'14 paper, http://hal.inria.fr/hal-00924264

References in notes

[31] , *Automotive Open System Architecture*, 2003, http://www.autosar.org

[32] G. LEAVENS, M. SITARAMAN (editors). , *Foundations of Component-Based Systems*, Cambridge University Press, 2000

[33] Z. LIU, H. JIFENG (editors). , *Mathematical Frameworks for Component Software - Models for Analysis and Synthesis*, World Scientific, 2006

[34] ARTEMIS JOINT UNDERTAKING. , *ARTEMIS Strategic Research Agenda*, 2011

[35] 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, IEEE, July 2010, pp. 159–168

[36] 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

[37] 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, pp. 437–451

[38] I. ASSAYAD, A. GIRAULT, H. KALLA. *Scheduling of Real-Time Embedded Systems under Reliability and Power Constraints*, in "International Conference on Complex Systems, ICCS'12", Agadir, Morocco, IEEE, November 2012

[39] P. AXER, R. ERNST, H. FALK, A. GIRAULT, D. GRUND, N. GUAN, B. JONSSON, P. MARWEDEL, J. REINEKE, C. ROCHANGE, M. SEBATIAN, R. VON HANXLEDEN, R. WILHELM, W. YI. *Building Timing Predictable Embedded Systems*, in "ACM Trans. Embedd. Comput. Syst.", 2014, To appear

[40] E. BAINOMUGISHA, A. CARRETON, T. VAN CUTSEM, S. MOSTINCKX, W. DE MEUTER. *A Survey on Reactive Programming*, in "ACM Computing Surveys", 2013, vol. 45, n^{o} 4

[41] A. BASU, S. BENSALEM, M. BOZGA, J. COMBAZ, M. JABER, T.-H. NGUYEN, J. SIFAKIS. *Rigorous Component-Based System Design Using the BIP Framework*, in "IEEE Software", 2011, vol. 28, n^{o} 3

[42] A. BENVENISTE, P. CASPI, S. A. EDWARDS, N. HALBWACHS, P. LE GUERNIC, R. DE SIMONE. *The synchronous languages 12 years later*, in "Proceedings of the IEEE", 2003, vol. 91, n^{o} 1

[43] A. BENVENISTE, J. RACLET, B. CAILLAUD, D. NICKOVIC, R. PASSERONE, A. SANGIOVANNI-VICENTELLI, T. HENZINGER, K. LARSEN. *Contracts for the Design of Embedded Systems Part I: Methodology and Use Cases*, in "Proceedings of the IEEE", 2012

[44] A. BENVENISTE, J. RACLET, B. CAILLAUD, D. NICKOVIC, R. PASSERONE, A. SANGIOVANNI-VICENTELLI, T. HENZINGER, K. LARSEN. *Contracts for the Design of Embedded Systems Part II: Theory*, in "Proceedings of the IEEE", 2012

[45] G. BERNAT, A. BURNS, A. LLAMOSÍ. *Weakly Hard Real-Time Systems*, in "IEEE Transactions on Computers", 2001, vol. 50, n^{o} 4, pp. 308-321, http://doi.ieeecomputersociety.org/10.1109/12.919277

[46] B. BONAKDARPOUR, S. S. KULKARNI, F. ABUJARAD. *Symbolic synthesis of masking fault-tolerant distributed programs*, in "Distributed Computing", 2012, vol. 25, n° 1

[47] S. BORKAR. *Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation*, in "IEEE Micro", 2005, vol. 25, n° 6

[48] R. BRUNI, H. C. MELGRATTI, U. MONTANARI. *Theoretical foundations for compensations in flow composition languages*, in "32nd ACM Symposium on Principles of Programming Languages (POPL)", ACM, 2005

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

[50] T. CHOTHIA, D. DUGGAN. *Abstractions for fault-tolerant global computing*, in "Theor. Comput. Sci.", 2004, vol. 322, n° 3

[51] P. COUSOT, R. COUSOT. *Abstract Interpretation and Application to Logic Programs*, in "Journal of Logic Programming", 1992, vol. 13, n° 2–3, pp. 103–179

[52] J. CÁMARA, A. GIRARD, G. GOESSLER. *Safety Controller Synthesis for Switched Systems Using Multi-Scale Symbolic Models*, in "CDC-ECC", IEEE, 2011, pp. 520-525

[53] R. I. DAVIS, A. BURNS. *A Survey of Hard Real-Time Scheduling for Multiprocessor Systems*, in "ACM Computing Surveys", 2011, vol. 43, n° 4

[54] V. DE FLORIO, C. BLONDIA. *A Survey of Linguistic Structures for Application-Level Fault-Tolerance*, in "ACM Computing Surveys", 2008, vol. 40, n° 2

[55] G. DELAVAL. , *Répartition modulaire de programmes synchrones*, INPGInria 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, pp. 101–110, ftp://ftp.inrialpes.fr/pub/bip/pub/girault/Publications/Lctes08/main.pdf

[57] S. A. EDWARDS, E. A. LEE. *The Case for the Precision Timed (PRET) Machine*, in "44th Design Automation Conference (DAC)", IEEE, 2007

[58] J. EKER, J. W. JANNECK, E. A. LEE, J. LIU, X. LIU, J. LUDVIG, S. NEUENDORFFER, S. SACHS, Y. XIONG. *Taming heterogeneity - the Ptolemy approach*, in "Proceedings of the IEEE", 2003, vol. 91, n° 1

[59] J. FIELD, C. A. VARELA. *Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments*, in "32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)", ACM, 2005

[60] P. FRADET, A. GIRAULT, P. POPLAVKO. *SPDF: A Schedulable Parametric Data-Flow MoC*, in "Design Automation and Test in Europe, DATE'12", Dresden, Germany, 2012, http://hal.inria.fr/hal-00744376

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

[62] A. GIRAULT. , *System-Level Design of Fault-Tolerant Embedded Systems*, October 2006, vol. 67, pp. 25–26

[63] 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

[64] 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, pp. 135–144

[65] D. GIZOPOULOS, M. PSARAKIS, S. V. ADVE, P. RAMACHANDRAN, S. K. S. HARI, D. SORIN, A. MEIXNER, A. BISWAS, X. VERA. *Architectures for Online Error Detection and Recovery in Multicore Processors*, in "Design Automation and Test in Europe (DATE)", 2011

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

[67] F. C. GÄRTNER. *Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments*, in "ACM Computing Surveys", 1999, vol. 31, n^{o} 1

[68] S. HAAR, E. FABRE. *Diagnosis with Petri Net Unfoldings*, in "Control of Discrete-Event Systems", Lecture Notes in Control and Information Sciences, Springer, 2013, vol. 433, chap. 15

[69] J. HALPERN, J. PEARL. *Causes and Explanations: A Structural-Model Approach. Part I: Causes*, in "British Journal for the Philosophy of Science", 2005, vol. 56, n^{o} 4, pp. 843-887

[70] D. HARMANCI, V. GRAMOLI, P. FELBER. *Atomic Boxes: Coordinated Exception Handling with Transactional Memory*, in "25th European Conference on Object-Oriented Programming (ECOOP)", Lecture Notes in Computer Science, 2011, vol. 6813

[71] T. HENZINGER, J. SIFAKIS. *The Embedded Systems Design Challenge*, in "Formal Methods 2006", Lecture Notes in Computer Science, Springer, 2006, vol. 4085

[72] I. HWANG, S. KIM, Y. KIM, C. E. SEAH. *A Survey of Fault Detection, Isolation and Reconfiguration Methods*, in "IEEE Trans. on Control Systems Technology", 2010, vol. 18, n^{o} 3

[73] V. IZOSIMOV, P. POP, P. ELES, Z. PENG. *Scheduling and Optimization of Fault-Tolerant Embedded Systems with Transparency/Performance Trade-Offs*, in "ACM Trans. Embedded Comput. Syst.", 2012, vol. 11, n^{o} 3, 61 p.

[74] R. KÜSTERS, T. TRUDERUNG, A. VOGT. *Accountability: definition and relationship to verifiability*, in "ACM Conference on Computer and Communications Security", 2010, pp. 526-535

[75] I. LANESE, C. A. MEZZINA, J.-B. STEFANI. *Reversing Higher-Order Pi*, in "21th International Conference on Concurrency Theory (CONCUR)", Lecture Notes in Computer Science, Springer, 2010, vol. 6269

[76] E. A. LEE, A. L. SANGIOVANNI-VINCENTELLI. *Component-based design for the future*, in "Design, Automation and Test in Europe, DATE 2011", IEEE, 2011

[77] 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

[78] P. MENZIES. *Counterfactual Theories of Causation*, in "Stanford Encyclopedia of Philosophy", E. ZALTA (editor), Stanford University, 2009, http://plato.stanford.edu/entries/causation-counterfactual

[79] M. MOORE. , *Causation and Responsibility*, Oxford, 1999

[80] H. NEGI, T. MITRA, A. ROYCHOUDHURY. *Accurate Estimation of Cache-Related Preemption Delay*, in "International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS'03", ACM, 2003, pp. 201–206

[81] J. PEARL. *Causal inference in statistics: An overview*, in "Statistics Surveys", 2009, vol. 3, pp. 96-146

[82] 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, pp. 206–230

[83] G. RAMALINGAM, K. VASWANI. *Fault tolerance via idempotence*, in "40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)", ACM, 2013

[84] B. RANDELL. *System Structure for Software Fault Tolerance*, in "IEEE Trans. on Software Engineering", 1975, vol. 1, n^{o} 2

[85] J. REINEKE, D. GRUND, C. BERG, R. WILHELM. *Timing Predictability of Cache Replacement Policies*, in "Real-Time Syst.", November 2007, vol. 37, n^{o} 2, pp. 99–122, http://rw4.cs.uni-saarland.de/~grund/papers/rts07-predictability.pdf

[86] J. RUSHBY. , *Partitioning for Safety and Security: Requirements, Mechanisms, and Assurance*, NASA Langley Research Center, 1999, n^{o} CR-1999-209347

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

[88] H. THEILING, C. FERDINAND, R. WILHELM. *Fast and Precise WCET Prediction by Separate Cache and Path Analyses*, in "Real-Time Syst.", May 2000, vol. 18, n^{o} 2/3, pp. 157–179

[89] D. WALKER, L. W. MACKEY, J. LIGATTI, G. A. REIS, D. I. AUGUST. *Static typing for a faulty lambda calculus*, in "11th ACM SIGPLAN International Conference on Functional Programming (ICFP)", ACM, 2006

[90] 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