Activity Report 2014

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, Inria, Senior Researcher]
Pascal Fradet [Inria, Researcher, HdR]
Alain Girault [Inria, Senior Researcher, HdR]
Gregor Goessler [Inria, Researcher, HdR]
Sophie Quinton [Inria, Researcher]

Faculty Member
Xavier Nicollin [Univ. Grenoble I, Associate Professor, from Sep 2014]

PhD Students
Vagelis Bebelis [STMicroelectronics, Cifre grant]
Dmitry Burlyaev [Univ. Grenoble I]
Yoann Geoffroy [Univ. Grenoble I, MESR grant from Oct. 2013]
Vagelis Bebelis [STMicroelectronics, CIFRE grant]
Dmitry Burlyaev [Université Joseph Fourier Grenoble I]
Yoann Geoffroy [Université Joseph Fourier Grenoble I, MESR grant, from Oct 2013]

Post-Doctoral Fellows
Adnan Bouakaz [Inria, from Oct 2014]
Wei-Tsun Sun [Inria, until May 2014]

Visiting Scientists
Eugene Yip [PhD, April 2014]
Ismail Assayad [Professor, September 2014]
Imen Boudabous [PhD, from Sep 2014 until Oct 2014]
Lilia Sfaxi [Professeur, September 2014]
Hugh Wang [PhD, November 2014]
Parthasarathi Roop [Professor, November 2014]

Administrative Assistant
Helen Pouchot [Inria]

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 [28], 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 [53]. Witness component-based toolsets such as UC Berkeley's Ptolemy [44], Verimag's BIP [30], or the modular architecture frameworks used, for instance, in the automotive industry (AUTOSAR) [25]. 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.*, [26], [27]). 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.*, [32], [33] and the references therein), but having a comprehensive theory of contracts that deals with all the above aspects is still an open question [58]. 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 [31]), 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 [29], [41], 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 for both 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. [29]. 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 [36], [48]. 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 [65], 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 [42]. 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.*, [52] 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 [54].

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.*, [49] 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 [34], [37], [39], [46], [55], [64], [69] 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 [60], natural sciences, law [61], and statistics [62], 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 [57], to allow the diagnosis of faults in a complex concurrent system [50], or to enforce accountability [56], 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 [49]), or is broken (*e.g.*, by limiting fault propagation [66]).

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. New Software and Platforms

5.1. Prototypes

5.1.1. Logical Causality

Participant: Gregor Goessler.

We are developing LOCA, a prototype tool written in Scala that implements the analysis of logical causality described in 6.3.3. LOCA currently supports causality analysis in BIP and networks of timed automata. 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.

5.1.2. Cosyma

Participant: Gregor Goessler.

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

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.

5.1.3. The SIAAM virtual machine

Participant: Jean-Bernard Stefani.

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. 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.

The SIAAM machine is the subject of Quentin Sabah's PhD thesis [67].

5.1.4. pyCPA_TCA

Participant: Sophie Quinton.

We are developing PYCPA_TCA, a PYCPA plugin for Typical Worst-Case Analysis as described in Section 6.2.2. PYCPA is an open-source Python implementation of Compositional Performance Analysis developed at TU Braunschweig, which allows in particular response-time analysis. PYCPA_TCA is an extension of this tool that is co-developed by Sophie Quinton and Zain Hammadeh at TU Braunschweig. It allows in particular the computation of weakly-hard guarantees for real-time tasks, i.e. number of deadline misses out of a sequence of executions. So far, PYCPA_TCA is restricted to uniprocessor systems of independent tasks, scheduled according to static priority scheduling.

6. New Results

6.1. Components and Contracts

Participant: Jean-Bernard Stefani.

6.1.1. Location graph model

The design of configurable systems can be streamlined and made more systematic by adopting a component-based structure, as demonstrated with the Fractal component model [2]. However, the formal foundations for configurable component-based systems, featuring higher-order capabilities where components can be dynamically instantiated and passivated, and non-hierarchical structures where components can be contained in different composites at the same time, are still an open topic. We have developed recently the location graph model [15], where components are understood as graphs of locations hosting higher-order processes, and where component structures can be arbitrary graphs. We have developed a compositional operational semantics for the location graph model, which is parametric with respect to the family of processes. We have shown that the location graph model constitutes a conservative extension of a previous model, called CAB, that captures the key features of the BIP component model [5]. We have further worked on the behavioral theory of the location graph model, characterizing contextual equivalence in the model by means of a higher-order bisimularity relation, and begun the study of the encoding of different models, including the Synchronized Hyperedge Replacement model [45].

6.2. Real-Time multicore programming

Participants: Vagelis Bebelis, Adnan Bouakaz, Pascal Fradet, Alain Girault, Gregor Goessler, Jean-Bernard Stefani, Sophie Quinton, Partha Roop, Eugene Yip.

6.2.1. 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.

Recently, we have introduced the *schedulable parametric data-flow* (SPDF) MoC for dynamic streaming applications [47]. SPDF extends the standard dataflow model by allowing rates to be parametric. Last year, we have 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). High dynamicity 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 the liveness and the boundedness of BDPF graphs.

Recently, we have proposed a generic and flexible framework to generate parallel schedules for BPDF applications [16]. The parametric dataflow graph is associated with 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. Our case studies are video decoders for high definition video streaming such as VC-1. One of the target architectures is the STHORM many-core platform designed by STMicroelectronics.

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

6.2.2. Typical Worst-Case Analysis of real-time systems

Weakly hard time constraints have been proposed for applications where occasional deadline misses are permitted. We have recently developed Typical Worst Case Analysis (TWCA) to exploit similar constraints and bound response times of systems with sporadic overload. This year, we have applied this approach to a real-life automotive network [14]. Additionally, we have extended the approach for static priority preemptive (SPP) and static priority non-preemptive (SPNP) scheduling to determine the maximum number of deadline misses of a given task [21]. The approach is based on an optimization problem which trades off higher priority interference versus miss count. We formally derived a lattice structure for these combinations that lays the ground for an integer linear programming (ILP) formulation. The ILP solution was evaluated and provided far better results than previous TWCA.

In parallel, we have contributed to a systematic co-engineering approach that integrates TWCA into functional analysis [19]. We combine physical, control and timing models by representing them as a network of hybrid automata. Closed-loop properties can then be verified on this hybrid automata network by using standard model checkers for hybrid systems. The use of the Logical Execution Time (LET) semantics where data is written back deterministically at the typical worst-case response time (rather than the usual worst-case bound) is a new and particularly powerful approach for addressing the computational complexity of the model checking problem.

6.2.3. Time predictable programming

In the context of the RIPPES associated team with UC Berkeley and U Auckland, we have finalized ongoing work on our synchronous programming language for time predictability PRET-C [10]. PRET-C extends C with synchronous constructs inspired by ESTEREL, to allow an easy programming of concurrent reactive programs.

These constructs allow the programmer to express concurrency, interaction with the environment, looping, and a synchronization barrier (like the `pause` statement in ESTEREL). PRET-C's semantics is deterministic, and it can be efficiently compiled towards sequential code, either executed on a dedicated processor for the best predictability of the program's Worst-Case Reaction Time (WCRT), or executed on a generic processor.

We have also continued our work on FOREC, a time predictable synchronous programming language for multi-core chips. Like PRET-C, it extends C with a small set of ESTEREL-like synchronous primitives. 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 multi-core chip. This is the main difference with PRET-C. We have finalized the semantics of FOREC, which led us to propose several ways to combine shared variables at the tick boundaries, such that the semantics remains deterministic. This part was inspired by the so-called concurrent revisions [38].

Finally, with colleagues from the former ARTISTDESIGN European Network of Excellence, we have also participated in a survey on predictable embedded systems [11].

6.2.4. Tradeoff exploration between energy consumption and execution time

We have continued our work on multi-criteria scheduling, in the particular context of dynamic applications that are launched and terminated on an embedded multi-core chip, under execution time and energy consumption constraints. We have proposed a two layer adaptive scheduling method. In the first layer, each application (represented as a DAG of tasks) is scheduled statically on sets of cores: 2 cores, 3 cores, 4 cores, and so on. For each size of these sets (2, 3, 4, ...), there may be only one topology or several topologies. For instance, for 2 or 3 cores there is only one topology (a "line"), while for 4 cores there are three distinct topologies ("line", "square", and "T shape"). Moreover, for each topology, we generate statically several schedules, each one subject to a different total energy consumption constraint, and consequently with a different Worst-Case Reaction Time (WCRT). Coping with the energy consumption constraints is achieved thanks to Dynamic Frequency and Voltage Scaling (DVFS). In the second layer, we use these pre-generated static schedules to reconfigure dynamically the applications running on the multi-core each time a new application is launched or an existing one is stopped. The goal of the second layer is to perform a global optimization of the configuration, such that each running application meets a pre-defined quality-of-service constraint (translated into an upper bound on its WCRT) and such that the total energy consumption is minimized. For this, we (1) allocate a sufficient number of cores to each active application, (2) allocate the unassigned cores to the applications yielding the largest gain in energy, and (3) choose for each application the best topology for its subset of cores (i.e., better than the by default "line" topology).

This is a joint work with Ismail Assayad (U. Casablanca, Morocco) who visits the team regularly.

6.3. Language Based Fault-Tolerance

Participants: Dmitry Burlyaev, Pascal Fradet, Alain Girault, Yoann Geoffroy, Gregor Goessler, Jean-Bernard Stefani.

6.3.1. Automatic transformations for fault tolerant circuits

In the past 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 consider both single-event upsets (SEU) and single-event transients (SET) and fault models of the form "at most 1 SEU or SET within n clock signals".

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 [17]. 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 have proposed novel fault-tolerance transformations based on time-redundancy. In particular, we have presented a transformation using double-time redundancy (DTR) coupled with micro-checkpointing, rollback and a speedup mode [18]. The approach is capable to mask any SET every 10 cycles and keeps the same input/output behavior regardless error occurrences. Experimental results on the ITC'99 benchmark suite indicate that the hardware overhead is 2.7 to 6.1 times smaller than full TMR with double loss in throughput. It is an interesting alternative to TMR for logic intensive designs.

We have also designed a transformation that allows the circuit to change its level of time-redundancy. This feature permits to dynamically and temporarily give up (resp. increase) fault-tolerance and speed up (resp. slow down) the circuit. The motivations for such changes can be based on the observed change in radiation environment or the processing of (non)critical data. These different time redundancy transformations have been patented [23]

We have started the formal certification of such transformations using the Coq proof assistant [40]. The transformations are described on a simple gate-level hardware description language inspired from μFP [68]. The fault-model is described in the operational semantics of the language. The main theorem states that, for any circuit, for any input stream and for any SET allowed by the fault-model, its transformed version produces a correct output. A TMR and triple time redundancy transformations have already been proved correct. The proof of the DTR transformation is in progress.

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).

We have revisited our encoding of our reversible higher-order π-calculus in (a variant of) the higher-order π-calculus, in order to obtain a much tighter result than our original encoding. In essence, we now have a form of strong bisimilarity (modulo administrative reductions) between a reversible higher-order π-calculus process and its translation in higher-order π. We have also studied the relation between the causality information used in our reversible higher-order π and a causal higher-order π-calculus, inspired by the causal π-calculus [35]. This work has been submitted for publication [24]. This work was done in collaboration with Inria teams FOCUS in Bologna, as part of the ANR REVER project.

6.3.3. *Blaming in component-based systems*

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.

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 [51] and its instantiations. However, this approach relies on a causal model that may not be known, for instance in presence of black-box components. For such systems, we have been developing a framework for blaming 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. We have shown in [12] how our approach can be used for log analysis to help establishing liability in the context of legal contracts.

We have proposed in [6] an approach for blaming in component-based real-time systems whose component specifications are given as timed automata. The analysis is based on a single execution trace violating a safety property P. We have formalized blaming using counterfactual reasoning to distinguish component failures that actually contributed to the outcome from failures that had no impact on the violation of P. We have shown how to effectively implement blaming by reducing it to a model-checking problem for timed automata. The approach has been implemented in LOCA (Section 5.1.1). We have further demonstrated the feasibility of our approach on the model of a dual-chamber implantable pacemaker.

6.3.4. Synthesis and implementation of fault-tolerant embedded systems

We have integrated a complete workflow to synthesize and implement correct-by-construction fault tolerant distributed embedded systems consisting of real-time periodic tasks. Correct-by-construction is provided by the use of discrete controller synthesis [63] (DCS), a formal method thanks to which we are able to guarantee that the synthe-sized controlled system satisfies the functionality of its tasks even in the presence of processor failures. For this step, our workflow uses the Heptagon domain specific language [43] and the Sigali DCS tool [59]. The correct implementation of the resulting distributed system is a challenge, all the more since the controller itself must be tolerant to the processor failures. We achieve this step thanks to the libDGALS real-time library [22] (1) to generate the glue code that will migrate the tasks upon processor failures, maintaining their internal state through migration, and (2) to make the synthesized controller itself fault-tolerant. We have demonstrated the feasibility of our work-flow on a multi-tasks multi-processor fault-tolerant distributed system.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

- With Orange Labs: software architecture for GlobalOS

7.2. Bilateral Grants with Industry

- ST Microelectronics: CIFRE contract for the PhD of Vagelis Bebelis. This work is described in Section 6.2.1.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR Projects

8.1.1.1. PiCoq (ANR project)

Participant: 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)

Participant: 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. European Initiatives

8.2.1. Collaborations in European Programs, except FP7 & H2020

Program: COST

Project acronym: IC1405

Project title: Reversible Computation

Duration: 2015-2019

Coordinator: I. Ulidowski (U. Leicester, UK)

Abstract: This recently launched COST Action aims to establisjh a research network of excellence on reversible commputation. Reversible computation is an emerging paradigm that extends the standard forward-only mode of computation with the ability to execute in reverse, so that computation can run backwards as naturally as it can go forwards. It aims to deliver novel computing devices and software. the potential benefits include the design of new reversible logic gates and circuits – leading to low-power computing –, and new conceptual frameworks, language abstractions and software tools for reliable and recovery-oriented distributed systems.

8.3. International Initiatives

8.3.1. Inria Associate Teams

8.3.1.1. RIPPES

Title: RIgorous Programming of Predictable Embedded Systems

International Partner (Institution - Laboratory - Researcher):

University of California Berkeley (USA)

University of Auckland (New Zealand)

Duration: 2013 - 2015

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

The RIPPES associated teams gathers the SPADES team from Inria Grenoble Rhône-Alpes, the Ptolemy group from UC Berkeley (EECS Department), and the Embedded Systems Research group from U. 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.

8.3.2. Inria International Partners

8.3.2.1. Informal International Partners

University of Bologna, Department of Computer Science (Italy)

Topics: reversibility in concurrent languages

TU Braunschweig, (Germany)

Topics: typical worst-case schedulability analysis

8.4. International Research Visitors

8.4.1. Visits of International Scientists

- April 2014: Eugene Yip (PhD student, U. Auckland) visited Inria Grenoble to work on the semantics of the FOREC PRET programming language (RIPPES associated team).
- April 2014: David Broman (Ass. Prof. KTH Stockholm and UC Berkeley) visited Inria Grenoble to attend the RePP'14 workshop and to work on PRET programming (RIPPES associated team).
- September 2014: Ismail Assayad (Ass. Prof. U. Casablanca) visited Inria Grenoble to work on multi-criteria optimization and scheduling for embedded system.
- September 2014: Lilia Sfaxi (Ass. Prof. ENSI Tunis) and Imen Boudabous (PhD student, ENSI Tunis) visited Inria Grenoble to work on scheduling and energy optimization of data-flow applications on multi-core chips.
- November and December 2014: Partha Roop (Senior Lecturer, U. Auckland) and Hugh Wang (PhD student, U. Auckland) visited Inria Grenoble to work on the FOREC PRET programming language (RIPPES associated team).

8.4.2. Visits to International Teams

- Alain Girault visited UC Berkeley (USA) in February 2014 to work on the parametric dataflow model of computation and on PRET programming (RIPPES associated team).

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organisation

9.1.1.1. general chair, scientific chair

- Sophie Quinton was PC co-chair and organizing committee member of the *First Workshop on Formal Methods for Timing Verification* (FMTV'14).
- Alain Girault was PC co-chair and co-organizer of the *Second Workshop on Reconciling Performance with Predictability* (RePP'14), which took place on March 5th 2014, in Grenoble, France.
- Alain Girault was co-chair and co-organizer of the *Workshop on Synchronous Programming* (SYNCHRON'14), which took place in December 2014 in Aussois, France.

9.1.2. Scientific events selection

9.1.2.1. Participation in conference steering committees

- Jean-Bernard Stefani is the current chair of the Steering Committee of the *IFIP FORTE* conference series, and a member of the Stering Committee of the *DisCoTec* federated conference series.

9.1.2.2. Participation in conference program committees

- Pascal Fradet served in the program committees of the *13th International Conference on Modularity* (MODULARITY'14) and of *Journées Francophones des Langages Applicatifs* (JFLA'14).
- Gregor Goessler served in the program committees of the *17th International Symposium on Component-Based Software Engineering* (CBSE'14) and *Design, Automation, and Test in Europe* (DATE'14).
- Sophie Quinton served in the program committees of the *26th Euromicro Conference on Real-Time Systems* (ECRTS'14), *34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems* (FORTE'14) and *12th IEEE International Symposium on Parallel and Distributed Processing with Applications* (ISPA'14) conferences as well as the *First International Workshop on Multi-Objective Many-Core Design* (MOMAC'14) and *2nd International Workshop on Mixed Criticality Systems* (WMC'14) workshops and the Work-in-Progress session of ECRTS'14.
- Alain Girault served on the program committees of the *International Conference on Design and Test in Europe* (DATE'14), the *Design Automation Conference* (DAC'14), the *International Symposium on Industrial Embedded Systems* (SIES'14), and the *International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS'14).

9.1.2.3. Reviewer

- Gregor Goessler reviewed articles for CDC'14, POST'14, and TACAS'14.
- Sophie Quinton reviewed an article for the *51th Design Automation Conference* (DAC'14).

9.1.3. Journal

9.1.3.1. Member of the editorial board

- Jean-Bernard Stefani is a member of the editorial board of Annals of Telecommunications.
- Alain Girault is a member of the editorial board of the EURASIP Journal on Embedded Systems.

9.1.3.2. Reviewer

- Jean-Bernard Stefani reviewed articles for *Theoretical Computer Science*.
- Gregor Goessler reviewed articles for *Science of Computer Programming* and *ACM Transactions on Embedded Computing Systems*.
- Sophie Quinton reviewed articles for *ACM Transactions on Embedded Computing*, *ACM Transactions on Design Automation of Electronic Systems* and *Real-Time Systems*.
- Alain Girault reviewed articles for *IEEE Trans. on Parallel and Distributed Systems* and *ACM Trans. on Design Automation of Electronic Systems*

9.2. Teaching - Supervision - Juries

9.2.1. Supervision

PhD in progress: Vagelis Bebelis, “Boolean Parametric Data Flow. Modeling - Analyses - Implementation”, 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, advised by Gregor Goessler.

9.2.2. Juries

- Alain Girault was president of the PhD defense jury of Pranav Tendulkar (University Grenoble Alpes).
- Alain Girault was president of the PhD defense jury of Christian von Essen (University Grenoble Alpes).
- Alain Girault was referee for the computer science PhD thesis of Guillaume Aupy (ENS-Lyon).
- Jean-Bernard Stefani was president of the jury for Bertrand Jeannet’s HDR defense (University Grenoble Alpes).
- Jean-Bernard Stefani was rapporteur for Helene Martorell’s PhD defense (Toulouse University)

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

[6] G. GÖSSLER, L. ASTEFANOAEI. *Blaming in component-based real-time systems*, in "Proceedings of the 14th International Conference on Embedded Software - EMSOFT’14", Delhi, India, ACM, October 2014 [*DOI :* 10.1145/2656045.2656048], https://hal.inria.fr/hal-01078214

[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

Articles in International Peer-Reviewed Journals

[10] S. ANDALAM, P. ROOP, A. GIRAULT, C. TRAULSEN. *A Predictable Framework for Safety-Critical Embedded Systems*, in "IEEE Transactions on Computers", July 2014, 13 p. , https://hal.inria.fr/hal-01095468

[11] P. AXER, R. ERNST, H. FALK, A. GIRAULT, D. GRUND, N. GUAN, B. JONSSON, P. MARWEDEL, J. REINEKE, C. ROCHANGE, M. SEBASTIAN, R. VON HANXLEDEN, R. WILHELM, W. YI. *Building Timing Predictable Embedded Systems*, in "ACM Transactions in Embedded Computing Systems", February 2014, vol. 13, n^{O} 4, 38 p. , https://hal.inria.fr/hal-01095461

[12] G. GÖSSLER, D. LE MÉTAYER, E. MAZZA, M.-L. POTET, L. ASTEFANOAEI. *Apport des méthodes formelles pour l'exploitation de logs informatiques dans un contexte contractuel*, in "Technique et Science Informatiques (TSI)", 2014, pp. 63-84 [*DOI :* 10.3166/TSI.33.63-84], https://hal.inria.fr/hal-01078220

[13] R. SINHA, A. GIRAULT, G. GÖSSLER, P. ROOP. *A Formal Approach to Incremental Converter Synthesis for System-on-Chip Design*, in "ACM Transactions on Design Automation of Electronic Systems (TODAES)", 2014, vol. 20, 30 p. [*DOI :* 10.1145/2663344], https://hal.inria.fr/hal-01092255

Invited Conferences

[14] S. QUINTON, J. HENNIG, T. BONE, M. NEUKIRCHNER, R. ERNST, M. NEGREAN. *Typical Worst Case Response-Time Analysis and its Use in Automotive Network Design*, in "The 51st Annual Design Automation Conference 2014", San Francisco, CA, United States, June 2014 [*DOI :* 10.1145/2593069.2602977], https://hal.inria.fr/hal-01097619

[15] J.-B. STEFANI. *Components as Location Graphs*, in "11th International Symposium on Formal Aspects of Component Software", Bertinoro, Italy, Lecture Notes in Computer Science, September 2014, vol. 8997, https://hal.inria.fr/hal-01094208

International Conferences with Proceedings

[16] V. BEBELIS, P. FRADET, A. GIRAULT. *A framework to schedule parametric dataflow applications on many-core platforms*, in "Proceedings of the 2014 SIGPLAN/SIGBED conference on Languages, Compilers and Tools for Embedded Systems- LCTES", Edinburgh, United Kingdom, June 2014 [*DOI :* 10.1145/2666357.2597819], https://hal.inria.fr/hal-01095767

[17] D. BURLYAEV, P. FRADET, A. GIRAULT. *Verification-guided Voter Minimization in Triple-Modular Redundant Circuits*, in "DATE - Design, Automation and Test in Europe Conference", Dresden, Germany, March 2014, https://hal.inria.fr/hal-00911768

[18] D. BURLYAEV, P. FRADET, A. GIRAULT. *Automatic Time-Redundancy Transformation for Fault-Tolerant Circuits*, in "FPGA'15 - 23rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", Monterey, United States, February 2015, forthcoming [*DOI :* 10.1145/2684746.2689058], https://hal.inria.fr/hal-01095747

[19] G. FREHSE, A. HAMANN, S. QUINTON, M. WÖHRLE. *Formal Analysis of Timing Effects on Closed-loop Properties of Control Software*, in "35th IEEE Real-Time Systems Symposium 2014 (RTSS)", Rome, Italy, December 2014, https://hal.inria.fr/hal-01097622

[20] G. GÖSSLER, L. ASTEFANOAEI. *Blaming in component-based real-time systems*, in "Proceedings of the 14th International Conference on Embedded Software - EMSOFT'14", Delhi, India, ACM, October 2014 [*DOI :* 10.1145/2656045.2656048], https://hal.inria.fr/hal-01078214

[21] Z. A. H. HAMMADEH, S. QUINTON, R. ERNST. *Extending typical worst-case analysis using response-time dependencies to bound deadline misses*, in "14th International Conference on Embedded Software 2014 (EMSOFT)", New Delhi, India, October 2014 [*DOI :* 10.1145/2656045.2656059], https://hal.inria.fr/hal-01097621

[22] W.-T. SUN, A. GIRAULT, Z. SALCIC, A. MALIK. *libDGALS: A Library-based Approach to Design Dynamic GALS Systems*, in "9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)", Pisa, Italy, June 2014, https://hal.inria.fr/hal-00996978

Patents and standards

[23] D. BURLYAEV, P. FRADET, A. GIRAULT. *Procédé de fabrication automatisée d'un circuit électronique adapté pour détecter ou masquer des fautes par redondance temporelle, programme d'ordinateur et circuit électronique associés*, June 2014, n^{o} 1456080, https://hal.inria.fr/hal-01096054

Other Publications

[24] I. LANESE, C. A. MEZZINA, J.-B. STEFANI. *Reversibility in the higher-order π-calculus*, June 2014, https://hal.inria.fr/hal-01081714

References in notes

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

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

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

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

[29] 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

[30] 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

[31] 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

[32] 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

[33] 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

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

[35] M. BOREALE, D. SANGIORGI. *A Fully Abstract Semantics for Causality in the π-Calculus*, in "Acta Informatica", 1998, vol. 35, n^{o} 5

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

[37] 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

[38] S. BURCKHARDT, D. LEIJEN. *Semantics of Concurrent Revisions*, in "European Symposium on Programming, ESOP'11", Saarbrucken, Germany, LNCS, Springer, March 2011, n^{o} 6602, pp. 116–135

[39] T. CHOTHIA, D. DUGGAN. *Abstractions for fault-tolerant global computing*, in "Theor. Comput. Sci.", 2004, vol. 322, n^{o} 3

[40] COQ DEVELOPMENT TEAM. *The Coq proof assistant, 1989-2014*, http://coq.inria.fr/

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

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

[43] G. DELAVAL, E. RUTTEN, H. MARCHAND. *Integrating Discrete Controller Synthesis into a Reactive Programming Language Compiler*, in "Discrete Event Dynamic System", December 2013, vol. 23, n^{o} 4, pp. 1–34

[44] 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^{o} 1

[45] G. L. FERRARI, D. HIRSCH, I. LANESE, U. MONTANARI, E. TUOSTO. *Synchronised Hyperedge Replacement as a Model for Service Oriented Computing*, in "FMCO 2005", Lecture Notes in Computer Science, Springer, 2005, vol. 4111

[46] 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

[47] 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

[48] 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

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

[50] 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

[51] 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

[52] 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

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

[54] 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

[55] 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.

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

[57] 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

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

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

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

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

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

[63] 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

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

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

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

[67] Q. SABAH. *SIAAM:Simple Isolation for an Actor Abstract Machine*, Université de Grenoble, December 2013

[68] M. SHEERAN. *muFP, A Language for VLSI Design*, in "LISP and Functional Programming", 1984, pp. 104–112

[69] 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