RESEARCH CENTRE

Rennes - Bretagne Atlantique

2020

ACTIVITY REPORT

Project-Team

TEA

Time, Events and Architectures

IN COLLABORATION WITH: Institut de recherche en informatique et systèmes aléatoires (IRISA)

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Embedded and Real-time Systems

Contents

Project-Team TEA

Creation of the Team: 2014 January 01, updated into Project-Team: 2015 January 01

Keywords

Computer sciences and digital sciences

A1.2.5. – Internet of things

A1.2.7. – Cyber-physical systems

A1.5.2. – Communicating systems

A2.1.1. – Semantics of programming languages

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.9. – Synchronous languages

A2.1.10. – Domain-specific languages

A2.2.1. – Static analysis

A2.2.4. – Parallel architectures

A2.3. – Embedded and cyber-physical systems

A2.3.1. – Embedded systems

A2.3.2. – Cyber-physical systems

A2.3.3. – Real-time systems

A2.4. – Formal method for verification, reliability, certification

A2.4.1. – Analysis

A2.4.2. – Model-checking

A2.4.3. – Proofs

A2.5. – Software engineering

A2.5.1. – Software Architecture & Design

A2.5.2. – Component-based Design

A4.4. – Security of equipment and software

A4.5. – Formal methods for security

A7.2. – Logic in Computer Science

A7.2.3. – Interactive Theorem Proving

A7.3. – Calculability and computability

A8.1. – Discrete mathematics, combinatorics

A8.3. – Geometry, Topology

Other research topics and application domains

B5.1. – Factory of the future

B6.1.1. – Software engineering

B6.4. – Internet of things

B6.6. – Embedded systems

1 Team members, visitors, external collaborators

Research Scientists

- Jean-Pierre Talpin [Team leader, Inria, Senior Researcher, HDR]
- Thierry Gautier [Inria, Researcher]
- Rajesh Kumar Gupta [University of California San Diego, Advanced Research Position]

Post-Doctoral Fellow

- Xiong Xu [Inria, from Sep 2020]

PhD Students

- Lucas Franceschino [Inria]
- Stephane Kastenbaum [Mitsubishi Electric, CIFRE]
- Jean Joseph Marty [Inria]
- Shenghao Yuan [Inria, from Oct 2020]
- Liangcong Zhang [ENS Rennes]

Technical Staff

- Loïc Besnard [CNRS, Engineer]

Administrative Assistant

- Armelle Mozziconacci [CNRS]

Visiting Scientist

- Shuvra Bhattacharyya [University of Maryland, From 2020, Cominlabs International Chair, HDR]

2 Overall objectives

2.1 Introduction

An embedded architecture is an artifact of heterogeneous constituents and at the crossing of several design viewpoints: software, embedded in hardware, interfaced with the physical world. Time takes different forms when observed from each of these viewpoints: continuous or discrete, event-based or time-triggered: modeling and programming formalisms that represent software, hardware and physics significantly alter this perception of time. Therefore, time reasoning in system design is usually isolated to a specific design problem: simulation, profiling, performance, scheduling, parallelization, simulation. The aim of project-team TEA is to define conceptually unified frameworks for reasoning on composition and integration in cyber-physical system design, and to put this reasoning to practice by revisiting analysis and synthesis issues in real-time system design with soundness and compositionality gained from formalization.

2.2 Context

In the construction of complex systems, information technology (IT) has become a central force of revolutionary changes, driven by the exponential increase of computational power. In the field of telecommunication, IT provides the necessary basis for systems of networked distributed applications. In the field of control engineering, IT provides the necessary basis for embedded control applications. The combination of telecommunication and embedded systems into networked embedded systems opens up a new range of systems, capable of providing more intelligent functionalities, thanks to information and communication (ICT). Networked embedded systems have revolutionized several application domains: energy networks, industrial automation and transport systems.

20th-century science and technology brought us effective methods and tools for designing both computational and physical systems, such as for instance Simulink and Matlab. But the design of cyber-physical systems (CPS) is much more than the union of those two fields. Traditionally, information scientists only have a hazy notion of requirements imposed by the physical environment of computers. Similarly, mechanical, civil, and chemical engineers view computers strictly as devices executing algorithms. CPS design is, to date, mostly executed in this ad-hoc manner, without sound, mathematically grounded, integrative methodology. A new science of CPS design will allow to create machines with complex dynamics and high control reliability, and apply to new industries and applications, such as IoT or edge devices, in a reliable and economically efficient way. Progress requires nothing less than the construction of a new science and technology foundation for CPS that is simultaneously physical and computational.

2.3 Motivations

Beyond the buzzword, a CPS is an ubiquitous object of our everyday life. CPSs have evolved from individual independent units (e.g an ABS brake) to more and more integrated networks of units, which may be aggregated into larger components or sub-systems. For example, a transportation monitoring network aggregates monitored stations and trains through a large scale distributed system with relatively high latency. Each individual train is being controlled by a train control network, each car in the train has its own real-time bus to control embedded devices. More and more, CPSs are mixing real-time low latency technology with higher latency distributed computing technology.

In the past 15 years, CPS development has moved towards Model Driven Engineering (MDE). With MDE methodology, first all requirements are gathered together with use cases, then a model of the system is built (sometimes several models) that satisfy the requirements. There are several modeling formalisms that have appeared in the past ten years with more or less success. The most successful are the *executable* models [1] [2] [3], i.e., models that can be simulated, exercised, tested and validated. This approach can be used for both software and hardware.

A common feature found in CPSs is the ever presence of concurrency and parallelism in models. Large systems are increasingly mixing both types of concurrency. They are structured hierarchically and comprise multiple synchronous devices connected by buses or networks that communicate asynchronously. This led to the advent of so-called GALS (Globally Asynchronous, Locally Synchronous) models, or PALS (Physically Asynchronous, Logically Synchronous) systems, where reactive synchronous objects are communicating asynchronously. Still, these infrastructures, together with their programming models, share some fundamental concerns: parallelism and concurrency synchronization, determinism and functional correctness, scheduling optimality and calculation time predictability.

Additionally, CPSs monitor and control real-world processes, the dynamics of which are usually governed by physical laws. These laws are expressed by physicists as mathematical equations and formulas. Discrete CPS models cannot ignore these dynamics, but whereas the equations express the continuous behavior usually using real numbers (irrational) variables, the models usually have to work with discrete time and approximate floating point variables.

[1] *Matlab/Simulink*, https://fr.mathworks.com/products/simulink.html
[2] *Ptolemy*, http://ptolemy.eecs.berkeley.edu
[3] *SysML*, http://www.uml-sysml.org

2.4 Challenges

A cyber-physical, or reactive, or embedded system is the integration of heterogeneous components originating from several design viewpoints: reactive software, some of which is embedded in hardware, interfaced with the physical environment through mechanical parts. Time takes different forms when observed from each of these viewpoints: it is discrete and event-based in software, discrete and time-triggered in hardware, continuous in mechanics or physics. Design of CPS often benefits from concepts of multiform and logical time(s) for their natural description. High-level formalisms used to model software, hardware and physics additionally alter this perception of time quite significantly.

In model-based system design, time is usually abstracted to serve the purpose of one of many design tasks: verification, simulation, profiling, performance analysis, scheduling analysis, parallelization, distribution, or virtual prototyping. For example in non-real-time commodity software, timing abstraction such as number of instructions and algorithmic complexity is sufficient: software will run the same on different machines, except slower or faster. Alternatively, in cyber-physical systems, multiple recurring instances of meaningful events may create as many dedicated logical clocks, on which to ground modeling and design practices.

Time abstraction increases efficiency in event-driven simulation or execution (i.e SystemC simulation models try to abstract time, from cycle-accurate to approximate-time, and to loosely-time), while attempting to retain functionality, but without any actual guarantee of valid accuracy (responsibility is left to the model designer). Functional determinism (a.k.a. conflict-freeness in Petri Nets, monotonicity in Kahn PNs, confluence in Milner's CCS, latency-insensitivity and elasticity in circuit design) allows for reducing to some amount the problem to that of many schedules of a single self-timed behavior, and time in many systems studies is partitioned into models of computation and communication (MoCCs). Multiple, multiform time(s) raises the question of combination, abstraction or refinement between distinct time bases. The question of combining continuous time with discrete logical time calls for proper discretization in simulation and implementation. While timed reasoning takes multiple forms, there is no unified foundation to reasoning about multi-form time in system design.

The objective of project-team TEA is henceforth to define formal models for timed quantitative reasoning, composition, and integration in embedded system design. Formal time models and calculi should allow us to revisit common domain problems in real-time system design, such as time predictability and determinism, memory resources predictability, real-time scheduling, mixed-criticality and power management; yet from the perspective gained from inter-domain timed and quantitative abstraction or refinement relations. A regained focus on fundamentals will allow to deliver better tooled methodologies for virtual prototyping and integration of embedded architectures.

3 Research program

3.1 Previous Works

The challenges of team TEA support the claim that sound Cyber-Physical System design (including embedded, reactive, and concurrent systems altogether) should consider multi-form time models as a central aspect. In this aim, architectural specifications found in software engineering are a natural focal point to start from. Architecture descriptions organize a system model into manageable components, establish clear interfaces between them, collect domain-specific constraints and properties to help correct integration of components during system design. The definition of a formal design methodology to support heterogeneous or multi-form models of time in architecture descriptions demands the elaboration of sound mathematical foundations and the development of formal calculi and methods to instrument them.

System design based on the "synchronous paradigm" has focused the attention of many academic and industrial actors on abstracting non-functional implementation details from system design. This elegant design abstraction focuses on the logic of interaction in reactive programs rather than their timed behavior, allowing to secure functional correctness while remaining an intuitive programming model for embedded systems. Yet, it corresponds to embedded technologies of single cores and synchronous buses from the 90s, and may hardly cover the semantic diversity of distribution, parallelism, heterogeneity, of cyber-physical systems found in 21st century Internet-connected, true-time^{TM}-synchronized clouds, of tomorrow's grids.

By contrast with a synchronous hypothesis, yet from the same era, the polychronous MoCC is inherently capable of describing multi-clock abstractions of GALS systems. Polychrony is implemented in the data-flow specification language Signal, available in the Eclipse project POP [4] and in the CCSL standard [5] available from the TimeSquare project. Both provide tooled infrastructures to refine high-level specifications into real-time streaming applications or locally synchronous and globally asynchronous systems, through a series of model analysis, verification, and synthesis services. These tool-supported refinement and transformation techniques can assist the system engineer from the earliest design stages of requirement specification to the latest stages of synthesis, scheduling and deployment. These characteristics make polychrony much closer to the required semantic for compositional, refinement-based, architecture-driven, system design.

While polychrony was a step ahead of the traditional synchronous hypothesis, CCSL is a leap forward from synchrony and polychrony. The essence of CCSL is “multi-form time” toward addressing all of the domain-specific physical, electronic and logical aspects of cyber-physical system design.

3.2 Timed Modeling

To formalize timed semantics for system design, we shall rely on algebraic representations of time as clocks found in previous works and introduce a paradigm of "time system": refinement types that represent timed behaviors. Just as a type system abstracts data carried along operations in a program, a “time system” abstracts the causal interaction of that program module or hardware element with its environment, its pre- and post-conditions, its assumptions and guarantees, either logical or numerical, discrete or continuous. Some fundamental concepts of thelogics we envision are present in the clock calculi found in data-flow synchronous languages like Signal or Lustre, yet bound to a particular model of timed concurrency.

In particular, the principle of refinement type systems [6], is to associate information (data-types) inferred from programs and models with properties pertaining, for instance, to the algebraic domain on their value, or any algebraic property related to its computation: effect, memory usage, pre-post condition, value-range, cost, speed, time, temporal logic [7]. Being grounded on type and domain theories, such type systems system should naturally be equipped with program analysis techniques based on type inference (for data-type inference) or abstract interpretation (for program properties inference) to help establish formal relations between heterogeneous component “types”.

Gaining scalability requires the capacities to modularly decompose systems which can be obtained using Abadi and Lamport's concepts of “*Composing Specifications*” and implemented by the notion of assume-guarantee contracts or Dijkstra monads. Verification problems encompassing heterogeneously timed specifications are common and of great variety: checking correctness between abstract (e.g. the synchronous hypothesis) and concrete time models (e.g. real-time architectures) relates to desynchronisation (from synchrony to asynchrony) and scheduling analysis (from synchronous data-flow to hardware). More generally, they can be perceived from heterogeneous timing viewpoints (e.g. mapping a synchronous-time software on a real-time middle-ware or hardware).

This perspective demands capabilities to use abstraction and refinement mechanisms for time models (using simulation, refinement, bi-simulation, equivalence relations) but also to prove more specific properties (synchronization, determinism, endochrony). All this formalization effort will allow to effectively perform the tooled validation of common cross-domain properties (e.g. cost v.s. power v.s. performance v.s. software mapping) and tackle problems such as these integrating constraints of battery capacity, on-board CPU performance, available memory resources, software schedulability, to logical software correctness and plant controllability.

[4] *Polychrony on Polarsys*, `https://www.polarsys.org/projects/polarsys.pop`
[5] *Clock Constraints in UML/MARTE CCSL.* C. André, F. Mallet. RR-6540. INRIA, 2008. `http://hal.inria.fr/inria-00280941`
[6] *Abstract Refinement Types.* N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.
[7] *LTL types FRP.* A. Jeffrey. Programming Languages meets Program Verification.

3.3 Modeling Architectures

To address the formalization of such cross-domain case studies, modeling the architecture formally plays an essential role. An architectural model represents components in a distributed system as boxes with well-defined interfaces, connections between ports on component interfaces, and specifies component properties that can be used in analytical reasoning about the model. Several architectural modeling languages for embedded systems have emerged in recent years, including the SAE AADL [8], SysML [9], UML MARTE [10].

In system design, an architectural specification serves several important purposes. First, it breaks down a system model into components of manageable size and complexity, to establish clear interfaces between components. In this way, complexity becomes manageable by hiding details that are not relevant at a given level of abstraction. Clear, formally defined, and semantically rich component interfaces facilitate integration by allowing most validation efforts to be conducted modularly. Connections between components, which specify how components interact with each other, help propagate the guaranteed effects of a component to the assumptions of linked components.

Most importantly, an architectural model is a repository to share knowledge about the system being designed. This knowledge can be represented as requirements, design artifacts, component implementations, held together by a structural backbone. Such a repository enables automatic generation of analytical models for different aspects of the system, such as timing, reliability, security, performance, energy, etc. Since all the models are generated from the same source, the consistency of assumptions w.r.t. guarantees, of abstractions w.r.t. refinements, used for different analyses becomes easier, and can be properly ensured in a design methodology based on formal verification and synthesis methods.

Related works in this aim, and closer in spirit to our approach (to focus on modeling time) are domain-specific languages such as Prelude [11] to model the real-time characteristics of embedded software architectures. Conversely, standard architecture description languages could be based on algebraic modeling tools, such as interface theories with the ECDAR tool [12]. Project TEA contributed a formal semantics of the AADL standard and the proposal of a time specification annex, using CCSL to model concurrency, time and physical properties, and PSL to model timed traces.

3.4 Scheduling Theory

Based on sound formalization of time and CPS architectures, real-time scheduling theory provides tools for predicting the timing behavior of a CPS which consists of many interacting software and hardware components. Expressing parallelism among software components is a crucial aspect of the design process of a CPS. It allows for efficient partition and exploitation of available resources.

The literature about real-time scheduling [13] provides very mature schedulability tests regarding many scheduling strategies, preemptive or non-preemptive scheduling, uniprocessor or multiprocessor scheduling, etc. Scheduling of data-flow graphs has also been extensively studied in the past decades.

A milestone in this prospect is the development of abstract affine scheduling techniques [14]. It consists, first, of approximating task communication patterns (e.g. between Safety-Critical Java threads) using cyclo-static data-flow graphs and affine functions. Then, it uses state of the art ILP techniques to find optimal schedules and to concretize them as real-time schedules in the program implementations [15] [16].

Abstract scheduling, or the use of abstraction and refinement techniques in scheduling borrowed to the theory of abstract interpretation [17] is a promising development toward tooled methodologies to orchestrate thousands of heterogeneous hardware/software blocks on modern CPS architectures (just

[8] *Architecture Analysis and Design Language*, AS-5506. SAE, 2004. `http://standards.sae.org/as5506b`
[9] *System modeling Language*. OMG, 2007. `http://www.omg.org/spec/SysML`
[10] *UML Profile for MARTE*. OMG, 2009. `http://www.omg.org/spec/MARTE`
[11] *The Prelude language*. LIFL and ONERA, 2012. `http://www.lifl.fr/~forget/prelude.html`
[12] *PyECDAR, timed games for timed specifications*. INRIA, 2013. `https://project.inria.fr/pyecdar`
[13] *A survey of hard real-time scheduling for multiprocessor systems*. R. I. Davis and A. Burns. *ACM Computing Survey* 43(4), 2011.
[14] *Buffer minimization in EDF scheduling of data-flow graphs*. A. Bouakaz and J.-P. Talpin. LCTES, ACM, 2013.
[15] *ADFG for the synthesis of hard real-time applications*. A. Bouakaz, J.-P. Talpin, J. Vitek. ACSD, IEEE, June 2012.
[16] *Design of SCJ Level 1 Applications Using Affine Abstract Clocks*. A. Bouakaz and J.-P. Talpin. SCOPES, ACM, 2013.
[17] *La vérification de programmes par interprétation abstraite*. P. Cousot. Séminaire au Collège de France, 2008.

consider modern cars or aircrafts). It is an issue that simply defies the state of the art and known bounds of complexity theory in the field, and consequently requires a particular focus.

3.5 Verified programming for system design

The IoT is a network of devices that sense, actuate and change our immediate environment. Against this fundamental role of sensing and actuation, design of edge devices often considers actions and event timings to be primarily software implementation issues: programming models for IoT abstract even the most rudimentary information regarding timing, sensing and the effects of actuation. As a result, applications programming interfaces (API) for IoT allow wiring systems fast without any meaningful assertions about correctness, reliability or resilience.

We make the case that the "API glue" must give way to a logical interface expressed using contracts or refinement types. Interfaces can be governed by a calculus – a refinement type calculus – to enable reasoning on time, sensing and actuation, in a way that provides both deep specification refinement, for mechanized verification of requirements, and multi-layered abstraction, to support compositionality and scalability, from one end of the system to the other.

Our project seeks to elevate the “function as type” paradigm to that of “system as type”: to define a refinement type calculus based on concepts of contracts for reasoning on networked devices and integrate them as cyber-physical systems [18]. An invited paper [19] outlines our progress with respect to this aim and plans towards building a verified programming environment for networked IoT devices: we propose a type-driven approach to verifying and building safe and secure IoT applications.

Accounting for such constrains in a more principled fashion demands reasoning about the composition of all the software and hardware components of the application. Our proposed framework takes a step in this direction by (1) using refinement types to make physical constraints explicit and (2) imposing an event-driven programming discipline to simplify the reasoning of system-wide properties to that of an event queue. In taking this approach, our approach would make it possible for a developer to build a verified IoT application by ensuring that a well-typed program cannot violate the physical constraints of its architecture and environment.

4 Application domains

4.1 Automotive and Avionics

From our continuous collaboration with major academic and industrial partners through projects TOPCASED, OPENEMBEDD, SPACIFY, CESAR, OPEES, P and CORAIL, our experience has primarily focused on the aerospace domain. The topics of time and architecture of team TEA extend to both avionics and automotive. Yet, the research focuses on time in team TEA is central in any aspect of, cyber-physical, embedded system design in factory automation, automotive, music synthesis, signal processing, software radio, circuit and system on a chip design.

Multi-scale, multi-aspect time modeling, analysis and software synthesis will greatly contribute to architecture modeling in these domains, with applications to optimized (distributed, parallel, multi-core) code generation for avionics (project Corail with Thales avionics) as well as modeling standards, real-time simulation and virtual integration in automotive (project with Toyota ITC, section 8).

Together with the importance of open-source software, one of these projects, the FUI Project P (section 8), demonstrated that a centralized model for system design could not just be a domain-specific programming language, such as discrete Simulink data-flows or a synchronous language. Synchronous languages implement a fixed model of time using logical clocks that are abstraction of time as sensed by software. They correspond to a fixed viewpoint in system design, and in a fixed hardware location in the system, which is not adequate to our purpose and must be extended.

In project P, we first tried to define a centralized model for importing discrete-continuous models onto a simplified implementation of SIMULINK: P models. Certified code generators would then be developed

[18]Refinement types for system design. Jean-Pierre Talpin. FDL’18 keynote.

[19]Steps toward verified programming of embedded computing systems. Jean-Pierre Talpin, Jean-Joseph Marty, Deian Stefan, Shravan Nagarayan, Rajesh Gupta, DATE’18.

from that format. Because this does not encompass all aspects being translated to P, the P meta-model is now being extended to architecture description concepts (of the AADL) in order to become better suited for the purpose of system design. Another example is the development of System modeler on top of SCADE, which uses the more model-engineering flavored formalism SysML to try to unambiguously represent architectures around SCADE modules.

An abstract specification formalism, capable of representing time, timing relations, with which heterogeneous models can be abstracted, from which programs can be synthesized, naturally appears better suited for the purpose of virtual prototyping. RT-Builder, based on the data-flow language Signal and developed by TNI, was industrially proven and deployed for that purpose at Peugeot. It served to develop the virtual platform simulating all on-board electronics of PSA cars. This 'hardware in the loop" simulator was used to test equipments supplied by other manufacturers for virtual prototyping of cars. In the advent of the related automotive standard, RT-Builder then became AUTOSAR-Builder.

4.2 Factory Automation

In collaboration with Mitsubishi R&D, we explore another application domain where time and domain heterogeneity are prime concerns: factory automation. In factory automation alone, a system is conventionally built from generic computing modules: PLCs (Programmable Logic Controllers), connected to the environment with actuators and detectors, and linked to a distributed network. Each individual, physically distributed, PLC module must be timely programmed to perform individually coherent actions and fulfill the global physical, chemical, safety, power efficiency, performance and latency requirements of the whole production chain. Factory chains are subject to global and heterogeneous (physical, electronic, functional) requirements whose enforcement must be orchestrated for all individual components.

Model-based analysis in factory automation emerges from different scientific domains and focuses on different CPS abstractions that interact in subtle ways: logic of PLC programs, real-time electro-mechanical processing, physical and chemical environments. This yields domain communication problems that render individual domain analysis useless. For instance, if one domain analysis (e.g. software) modifies a system model in a way that violates assumptions made by another domain (e.g. chemistry) then the detection of its violation may well be impossible to explain to either the software or chemistry experts. As a consequence, cross-domain analysis issues are discovered very late during system integration and lead to costly fixes. This is particularly prevalent in multi-tier industries, such as avionic, automotive, factories, where systems are prominently integrated from independently-developed parts.

5 Highlights of the year

Our teammate Loïc Besnard retired at the end of 2020. He had been promoted to the rank of Senior Engineer Exceptional Class by CNRS last year, to acknowledge his remarkable career of research engineer as principal developer of Signal and Polychrony, and as project manager and integrator with project teams EPATR (Signal), ESPRESSO (Polychrony), TEA (ADFG) and PACAP (Heptane).

As the bibliography of our annual report shows, we obvisouly decreased our efforts to publish papers at (online) international conferences in 2020 and, instead, focused on publishing project results with international journals or, sometimes, as full papers with major conferences. As of early 2021, seven such project articles are under submission, one of which is to appear in Science of Computer Programming as of February 2021.

6 New software and platforms

6.1 New software

6.1.1 ADFG

Name: Affine data-flow graphs schedule synthesizer

Keywords: Code generation, Scheduling, Static program analysis

Functional Description: ADFG is a synthesis tool of real-time system scheduling parameters: ADFG computes task periods and buffer sizes of systems resulting in a trade-off between throughput maximization and buffer size minimization. ADFG synthesizes systems modeled by ultimately cyclo-static dataflow (UCSDF) graphs, an extension of the standard CSDF model.

Knowing the WCET (Worst Case Execute Time) of the actors and their exchanges on the channels, ADFG tries to synthezise the scheduler of the application. ADFG offers several scheduling policies and can detect unschedulable systems. It ensures that the real scheduling does not cause overflows or underflows and tries to maximize the throughput (the processors utilization) while minimizing the storage space needed between the actors (i.e. the buffer sizes).

Abstract affine scheduling is first applied on the dataflow graph, that consists only of periodic actors, to compute timeless scheduling constraints (e.g. relation between the speeds of two actors) and buffering parameters. Then, symbolic schedulability policies analysis (i.e., synthesis of timing and scheduling parameters of actors) is applied to produce the scheduller for the actors.

ADFG, initially defined to synthesize real-time schedulers for SCJ/L1 applications, may be used for scheduling analysis of AADL programs.

Authors: Thierry Gautier, Jean-Pierre Talpin, Adnan Bouakaz, Alexandre Honorat, Loïc Besnard

Contacts: Jean-Pierre Talpin, Loïc Besnard

6.1.2 POLYCHRONY

Keywords: Code generation, AADL, Proof, Optimization, Multi-clock, GALS, Architecture, Cosimulation, Real time, Synchronous Language

Functional Description: Polychrony is an Open Source development environment for critical/embedded systems. It is based on Signal, a real-time polychronous data-flow language. It provides a unified model-driven environment to perform design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony. It can be included in heterogeneous design systems with various input formalisms and output languages. The Polychrony tool-set provides a formal framework to: validate a design at different levels, by the way of formal verification and/or simulation, refine descriptions in a top-down approach, abstract properties needed for black-box composition, compose heterogeneous components (bottom-up with COTS), generate executable code for various architectures. The Polychrony tool-set contains three main components and an experimental interface to GNU Compiler Collection (GCC):

* The Signal toolbox, a batch compiler for the Signal language, and a structured API that provides a set of program transformations. It can be installed without other components and is distributed under GPL V2 license.

* The Signal GUI, a Graphical User Interface to the Signal toolbox (editor + interactive access to compiling functionalities). It can be used either as a specific tool or as a graphical view under Eclipse. It has been transformed and restructured, in order to get a more up-to-date interface allowing multi-window manipulation of programs. It is distributed under GPL V2 license.

* The POP Eclipse platform, a front-end to the Signal toolbox in the Eclipse environment. It is distributed under EPL license.

URL: `https://www.polarsys.org/projects/polarsys.pop`

Authors: Loïc Besnard, Paul Le Guernic, Thierry Gautier

Contacts: Loïc Besnard, Thierry Gautier

Participants: Loïc Besnard, Paul Le Guernic, Thierry Gautier

Partners: CNRS, Inria

6.1.3 Polychrony AADL2SIGNAL

Keywords: Real-time application, Polychrone, Synchronous model, Polarsys, Polychrony, Signal, AADL, Eclipse, Meta model

Functional Description: This polychronous MoC has been used previously as semantic model for systems described in the core AADL standard. The core AADL is extended with annexes, such as the Behavior Annex, which allows to specify more precisely architectural behaviors. The translation from AADL specifications into the polychronous model should take into account these behavior specifications, which are based on description of automata.

For that purpose, the AADL state transition systems are translated as Signal automata (a slight extension of the Signal language has been defined to support the model of polychronous automata).

Once the AADL model of a system transformed into a Signal program, one can analyze the program using the Polychrony framework in order to check if timing, scheduling and logical requirements over the whole system are met.

We have implemented the translation and experimented it using a concrete case study, which is the AADL modeling of an Adaptive Cruise Control (ACC) system, a highly safety-critical system embedded in recent cars.

URL: `http://www.inria.fr/equipes/tea`

Authors: Yue Ma, Loïc Besnard, Paul Le Guernic, Thierry Gautier, Huafeng Yu

Contacts: Loïc Besnard, Alexandre Honorat, Thierry Gautier, Jean-Pierre Talpin

Participants: Huafeng Yu, Loïc Besnard, Paul Le Guernic, Thierry Gautier, Yue Ma

Partner: CNRS

6.1.4 POP

Name: Polychrony on Polarsys

Keywords: Synchronous model, Model-driven engineering

Functional Description: The Eclipse project POP is a model-driven engineering front-end to our open-source toolset Polychrony. The Eclipse project POP is a model-driven engineering front-end to our open-source toolset Polychrony. It was finalised in the frame of project OPEES, as a case study: by passing the POLARSYS qualification kit as a computer aided simulation and verification tool. This qualification was implemented by CS Toulouse in conformance with relevant generic (platform independent) qualification documents. Polychrony is now distributed by the Eclipse project POP on the platform of the POLARSYS industrial working group. Team TEA aims at continuing its dissemination to academic partners, as to its principles and features, and industrial partners, as to the services it can offer.

Project POP is composed of the Polychrony tool set, under GPL license, and its Eclipse framework, under EPL license. SSME (Syntactic Signal-Meta under Eclipse), is the meta-model of the {Signal} language implemented with Eclipse/Ecore. It describes all syntactic elements specified in Signal Reference Manual: all {Signal} operators (e.g. arithmetic, clock synchronization), model (e.g. process frame, module), and construction (e.g. iteration, type declaration). The meta-model primarily aims at making the language and services of the Polychrony environment available to inter-operation and composition with other components (e.g. AADL, Simulink, GeneAuto, P) within an Eclipse-based development tool-chain. Polychrony now comprises the capability to directly import and export Ecore models instead of textual Signal programs, in order to facilitate interaction between components within such a tool-chain. The Polarsys download site for project POP has opened in 2015.

URL: `https://www.polarsys.org/projects/polarsys.pop`

Contacts: Thierry Gautier, Loïc Besnard, Jean-Pierre Talpin

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

7 New results

7.1 A logical framework to verify requirements of hybrid system models

Participants Jean-Pierre Talpin, Benoit Boyer, Stéphane Kastenbaum.

The goal of this PhD project is to build on the previous work done in Simon Lunel's PhD Thesis. The goal is to ensure the correctness-by-design of cyber-physical system models. The accent is put on developing a compositional contract framework that can help decomposing the systems into multiple sub-sytems easier to verify, leading to a proof of the global system.

We gained interest in the work of Benveniste et al. on multiple viewpoint contracts, and more particularly Pierluigi Nuzzo's PhD Thesis. It provides us a framework for decomposing systems into subsystems regardless of the formalism chosen to design it. In his Thesis, Pierliugi Nuzzo shows how this framework can also be used to show the compatibility of two subsystems designed in different formalism.

Another work that interested us is the framework developped by Naijun Zhan and its team at ISCAS. They worked on Hybrid Hoare Logic (HHL), Duration Calculus (DC) and Hybrid Communicating Sequential Process (HCSP). HCSP is a language to denote hybrid system, DC is a logic to describe traces of theses systems and HHL is a logic to show that DC traces corresponds to HCSP programs. They implemented all this components in MARS a software suite designed to translate Simulink systems into HCSP, and then prove some safety properties of the system with Isabelle. Our goal is to augment the MARS suite with a framework of multi-viewpoints contracts, implement it first in Isabelle/HOL, then plug it into MARS. This way, users will be able to design components in Simulink then export them with MARS into a collection of modular contracts and components. Which can be used to prove the safety of the system.

7.2 Semantic foundation for cyber-physical systems using higher-order UTP

Participants Jean-Pierre Talpin, Xiong Xu, Naijun Zhan.

The arrival of Xiong with team TEA strenghtened our collaboration with Naijun Zhan's group at ISCAS.

We proposed a higher-order unifying theory of programming (HUTP) for the semantics foundation of cyber-physical systems (CPSs). It supports the specification of discrete, real-time and continuous dynamics, concurrency and communication, and higher-order quantification. Within HUTP, we defined a calculus of hybrid designs to model, analyse, compose and refine heterogeneous CPSs. Some properties, such as deadlock, divergence, and non-termination, of hybrid designs were analysed algebraically. HUTP provides a uniform semantics domain capable of capturing all component models, at different abstraction levels, across heterogeneous domain concerns, into the same analysis framework.

Since there are natural links between hybrid designs and assume/guarantee contracts, we are trying to equip HUTP with compositional assume/guarantee reasoning and implement HUTP in Isabelle for automated verification. In addition, we wish to construct the HUTP semantics for the joint model AADL+Simulink/Stateflow and check the semantic equivalence between the informal data-flow model and its formal HCSP format within the HUTP framework.

We designed a higher-order extension of Hoare's UTP (Unified Theory of Programming) to modularly model hybrid and reactive systems. We demonstrated the expressiveness of the higher-order UTP (HUTP) by defining an encoding of two major logical frameworks for hybrid system design, namely the hybrid communicating sequential processes (HCSP) and hybrid programs with differential dynamic logic (dL). We introduced of a timed calculus of monadic refinement types based on an encoding of the duration calculus in HUTP to support contract-based reasoning using Dijkstra monads (in a similar manner as in the F* programming language). Finally, we started the development of higher-order UTP using the Isabelle/HOL theorem prover.

- "Semantic foundation for cyber-physical systems using higher-order UTP". Xiong Xu, Jean-Pierre Talpin, Shuling Wang, Naijun Zhan. Draft, submitted for publication, 20 pages, 2021.

7.3 Verified programming and secure integration of operating system libraries in F*

Participants Shenghao Yuan, Jean-Pierre Talpin, Frédéric Besson.

The dynamics of Inria's challenge RIOT-fp accelerated with the arrival of Shenghao, which broadened to additional collaborations with teams Celtique (Frédéric Besson) and Prosecco (Denis Mérigoux).

The Ph.D. project of Shenghao is about formal verification and security properties of embedded operation system RIOT libraries. There are many approaches and languages to support formal methods, such as timed automata (UPPAAL), model-driven approaches (AADL and Synchronous Languages), proof systems (Coq and F*).

In this project, we focus on using the state-of-the-art theorem prover F* to model and verify the external libraries, because we will benefit a lot of F* features, for instance, strong type system, refinement type, precondition/postcondition proof style, and automatic theorem proof. We rely on the static analyzer infer to automatically generate some separation logic properties which are used to refine the F* model of the libraries. The safety and security of these libraries will be enforced by using the Steel compiler with the proof of safety and security strategies to perform C program transformations.

Currently, we have modelled the RIOT bootloader library in Low* (a subset of F*), the bootloader includes the fletcher32 algorithm and some ARM assembly code. The former is modelled in Low* because of the pointer type and some memory operations (e.g. memory copy), while the latter is too low-level. To solve that problem, we have formalized the syntax and semantics of the kernel ARM instruction set and planned to extend the vale project to support the ARM architecture for the F* verification framework.

In 2020, we implemented a verified implementation of RIOT's bootloader that garantees the image integrity and the ARM assembly code that boots it, using F*'s Low* C code generator and by extending F*'s VALE framework to the verification of the ARM instructions of the bootloader responsible for jumping to the operating system's image.

- "Verified implementation of a secure bootloader using F* and VALE". Shenghao Yuan, Jean-Pierre Talpin. Draft submitted for publication, 25 pages, 2021.

7.4 Abstract interpretation as a Dijkstra monad transformer

Participants Lucas Franceschino, Jean-Pierre Talpin, David Pichardie.

This work aims at hybridizing abstract interpretation and weakest-preconditions, to enhance the programmer's experience working with certain kinds of dependently typed languages.

In such dependently typed language, the user is required to annotate terms with manual proofs or hints. While some automation facilities are provided by certain languages, i.e. to ease proofs, dependents types might be seen as requirements yielding proof obligations. On the other side, abstract interpretation is a sound theory of program approximation: an abstract interpreter gives rise to invariants by reading a source code.

These two theories are complementary: abstract interpretation sometimes produces invariants that one should have written explicitly as an annotation in its program. We particularly focus on bringing this hybridization to F*, by constructing a Dijkstra monad transformer.

This project aims at constructing an abstract interpreter along with a theorem of soundness, directly in F*. Most of the state of the art's abstract interpreter formalizations are written in Coq; in contrast, choosing F* makes our proof much lighter, thus our implementation is more concise. F* syntax is close to OCaml, and thus extracts to OCaml more or less transparently.

In collaboration with team Celtique, Lucas finalized the proof of soundness of Dijkstra monad transformer for hybridizing a (deductive) weakest pre-condition calculus with an abstract interpreter (for inferring pre- and post-conditions). This work lays the foundation for integrating abstract interpretations directly within F*'s dependent-type system to facilitate automated annotation and verification of programs:

> "Abstract interpretation as a Dijkstra monad transformer for low-level verification ". Lucas Franceschino, Jean-Pierre Talpin, David Pichardie. Draft submitted for publication, 25 pages, 2021.

7.5 Verified information flow of embedded programs

Participants Jean-Joseph Marty, Lucas Franceschino, Niki Vazou, Jean-Pierre Talpin.

This PhD project is about applying refinement types theory to verified programming of applications and modules of library operating systems, such as unikernels, for embedded devices of the Internet of Things (IoT): TinyOS, Riot, etc. Our topic has focused on developing a model of information flow control using labeled input-outputs (LIO) implemented using F⋆: project Lio⋆.

As part of the development of Lio⋆, we implemented a library that, thanks to static verification, ensures the containment of information in relation to a parameterized policy for information flow control. In collaboration with Niki Vazou (IMDEA) and Lucas Franceschino we have formalized and developed an automatic method to prove non-interference in Meta⋆. Using the Kremlin code generator, programs using Lio⋆ can be compiled into C code and run natively on embedded low-resource-constrained devices, without the need for additional runtime system. We completed a layered implementation of the LIO framework for information flow control of embedded applications developped with the Low* code generator and submitted our results to an A* conference:

> "LIO*: Information Flow Control for IoT devices with F*". Jean-Joseph Marty, Lucas Francechino, Jean-Pierre Talpin, Niki Vazou. Draft submission, 20 pages, 2020. An earlier version is available on arxiv.org

In parallel we continued our collaboration with the ProgSys team on a second, now discontinued, project: Gluco⋆. The goal of this project was to evaluate the capabilities to use the F* programming language to program an entire system by taking into account its software, hardware and physical constraints using type refinements [20].

7.6 The polychronous model, its incarnation language Signal and its open-source software toolset Polychrony

Participants Loïc Besnard, Thierry Gautier.

We have recently compared the model of asynchronous dataflow represented by Kahn Process Networks and that of synchronous dataflow represented by the polychronous model (a paper is still in review process). In line with our old and recent work on the polychronous model, we have produced a new version of the Signal language reference manual [13]. The previous version was about ten years old. Among the main changes made in this new version, we can note the addition of new classes of process models with in particular the automaton models, the addition of guarded processes (that may be compared, for instance, to "clocked guarded actions" that exist in some formalisms), a new definition of the *tick* of a

[20]Towards verified programming of embedded devices. J.-P. Talpin, J.-J. Marty, S. Narayan, D. Stefan, R. Gupta. Design, Automation and Test in Europe (DATE'19). IEEE, 2019.

process, and the addition of specific “pragmas” (pragmas allow to associate specific information that will be used by a given compiler or another tool). At the same time, we finalized a new version of the open-source software Polychrony, released online in December 2020 (`http://polychrony.inria.fr`). The development of this new version is now managed under git. It integrates the latest functionalities developed in the Signal compiler, in particular a code generation technique for so-called “polyendochronous processes” (which may be seen as particular networks of processes exhibiting polyhierarchies of clocks). New compiler options have been created for the new features and all previously available options have been reviewed (they are described in the online help obtained by the “signal -h” command).

7.7 ADFG: Affine data-flow graphs scheduler synthesis

Participants Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin, Shuvra Bhattacharyya, Alexandre Honorat, Hai Nam Tran.

We continued our research involving the development of advanced signal processing dataflow methods for the ADFG tool. This research is collaboratively developed with the TEA Team, Hai Nam Tran (Lab-STICC/UBO), Alexandre Honorat (INSA) and Shuvra Bhattacharyya (UMD/INSA/INRIA). Emphasis during this reporting period has been on the development of a framework to integrate the scheduling synthesized by ADFG into a scheduling simulator and a code generator targeting the RTEMS (Real-Time Executive for Multiprocessor Systems) RTOS `https://www.rtems.org`.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

Inria – Mitsubishi Electric framework program (2018+)

Title: Inria – Mitsubishi Electric framework program

INRIA principal investigator: Jean-Pierre Talpin

International Partner: Mitsubishi Electric R&D Europe (MERCE)

Duration: 2018+

Abstract: Following up the fruitful collaboration of TEA with the formal methods group at MERCE, Inria and Mitsubishi Electric signed an Inria-wide collaboration agreement, which currently hosts projects with project-teams Sumo and Tea, as well as Tocata.

8.2 Bilateral grants with industry

Mitsubishi Electric R&D Europe (2019-2022)

Title: A logical framework to verify requirements of hybrid system models

INRIA principal investigator: Jean-Pierre Talpin, Stéphane Kastenbaum

International Partner: Mitsubishi Electric R&D Europe

Duration: 2015 - 2018

Abstract: The goal of this doctoral project is to verify and build cyber-physical systems (CPSs) with a correct-by-construction approach in order to validate system requirements against the two facets of the cyber and physical aspects of such designs. Our approach is based on components augmented with formal contracts that can be composed, abstracted or refined. It fosters on the proof of system-level requirements by composing individual properties proved at component level. While semantically grounded, the tooling of this methodology should be usable by regular engineers (i.e. not proof theory specialists).

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Inria International Labs: Sino-European Laboratory in Computer Science, Automation and Applied Mathematics

CONVEX

Title: Compositional Verification of Cyber-Physical Systems

Duration: 2018 - 2021

Coordinator: Thierry GAUTIER

Partners:

State Key Laboratory of Computer Science, CAS (China)

Inria contact: Thierry GAUTIER

Summary: Formal modeling and verification methods have successfully improved software safety and security in vast application domains in transportation, production and energy. However, formal methods are labor-intensive and require highly trained software developers. Challenges facing formal methods stem from rapid evolution of hardware platforms, the increasing amount and cost of software infrastructures, and from the interaction between software, hardware and physics in networked cyber-physical systems.

Automation and expressivity of formal verification tools must be improved not only to scale functional verification to very large software stacks, but also verify non-functional properties from models of hardware (time, energy) and physics (domain). Abstraction, compositionality and refinement are essential properties to provide the necessary scalability to tackle the complexity of system design with methods able to scale heterogeneous, concurrent, networked, timed, discrete and continuous models of cyber-physical systems.

Project CONVEX wants to define a CPS architecture design methodology that takes advantage of existing time and concurrency modeling standards (MARTE, AADL, Ptolemy, Matlab), yet focuses on interfacing heterogeneous and exogenous models using simple, mathematically-defined structures, to achieve the single goal of correctly integrating CPS components.

Project CONVEX has been renewed in 2021.

9.1.2 Inria International Chair

Title: End-to-end system co-design

Duration: 2017 - 2021

Coordinator: Thierry GAUTIER

Partner:

University of California, San Diego (United States) - Rajesh Gupta

9.1.3 Insa-Inria Cominlabs Chair

Title: System design methodologies for real-time signal and information processing

Duration: 2020 - 2023

Partner:

University of Maryland (United States) - Shuvra Bhattacharyya

9.2 National initiatives

Inria Challenge RIOT-fp

Title: Future-proof IoT

Duration: 4 years

Coordinator: Emmanuel Bachelli

Partners:

Infine, Eva, Grace, Prosecco, Tea, Freie Universität Berlin and Fujitsu.

Summary: RIOT-fp https://future-proof-iot.github.io RIOT-fp is a research project on cyber-security targeting low-end, microcontroller-based IoT devices, on which operating systems such as RIOT run, and the development of a low-power network stack. Taking a global and practical approach, RIOT-fp gathers partners planning to enhance RIOT with an array of security mechanisms. The main challenges tackled by RIOT-fp are: 1/ developing high-speed, high-security, low-memory IoT crypto primitives, 2/ providing guarantees for software execution on low-end IoT devices, and 3/ enabling secure IoT software updates and supply-chain, over the network. Beyond academic outcomes, the output of RIOT-fp is open source code published, maintained and integrated in the open source ecosystem around RIOT. As such, RIOT-fp strives to contribute usable building blocks for an open source IoT solution improving the typical functionality vs. risk tradeoff for end-users.

The goal of project-team TEA in RIOT-fp is to build a trusted operating system library for IoT devices using composites of security modules, developed and verified in F⋆, and legacy C modules of the open-source RIOT distribution, interfaced to F⋆ using verified refinement types obtained from certified static analyzers (Verasco, MemCAD).

10 Dissemination

General chair, scientific chair Jean-Pierre Talpin serves the steering committee of the ACM-IEEE MEMOCODE conference since 2003, of which he was nominated Chair for three years in 2021.

Member of the conference program committees Jean-Pierre Talpin participated to the program committees of ACM LCTES'20, ACM SAC'20, SCOPES'20, SAMOS'20.

Member of the editorial boards Jean-Pierre Talpin served the ACM Transactions on Embedded Computing Systems (TECS) from 2013 to 2020 under the Chief Editorship of Sandeep Kumar Shukla.

10.1 Teaching - Supervision - Juries

Jean-Pierre Talpin served as "rapporteur" at the PhD Thesis defense of Guillaume Dupont, entitled "Correct-by-Construction Design of Hybrid Systems Based on Refinement and Proof".

10.2 Popularization

We published an article on Inria blog "Emergence" presenting the topic of our collaboration with Mitsubishi Electric (MERCE) in the PhD project of Stéphane Kastenbaum, and on the occasion of the recently signed Inria-MERCE framework program, that additionally hosts collaborations with teams Sumo and Toccata.

11 Scientific production

11.1 Major publications

[1] L. Besnard, T. Gautier, P. Le Guernic, C. Guy, J.-P. Talpin, B. Larson and E. Borde. 'Formal Semantics of Behavior Specifications in the Architecture Analysis and Design Language Standard'. In: *Cyber-Physical System Design from an Architecture Analysis Viewpoint*. Cyber-Physical System Design from an Architecture Analysis Viewpoint. Springer, Jan. 2017. DOI: 10.1007/978-981-10-4436-6_3. URL: https://hal.inria.fr/hal-01615143.

[2] L. Besnard, T. Gautier, P. Le Guernic and J.-P. Talpin. 'Compilation of Polychronous Data Flow Equations'. In: *Synthesis of Embedded Software*. Ed. by S. K. Shukla and J.-P. Talpin. Springer, 2010, pp. 1–40. DOI: 10.1007/978-1-4419-6400-7_1. URL: https://hal.inria.fr/inria-00540493.

[3] A. Bouakaz and J.-P. Talpin. 'Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks'. In: *International Workshop on Software and Compilers for Embedded Systems*. St. Goar, Germany, June 2013, pp. 58–67. DOI: 10.1145/2463596.2463600. URL: https://hal.inria.fr/hal-00916487.

[4] A. Gamatié, T. Gautier and P. Le Guernic. 'Synchronous design of avionic applications based on model refinements'. In: *Journal of Embedded Computing (IOS Press)* 2.3-4 (2006), pp. 273–289. URL: https://hal.archives-ouvertes.fr/hal-00541523.

[5] A. Honorat, H. N. Tran, L. Besnard, T. Gautier, J.-P. Talpin and A. Bouakaz. 'ADFG: a scheduling synthesis tool for dataflow graphs in real-time systems'. In: *International Conference on Real-Time Networks and Systems*. Grenoble, France, Oct. 2017, pp. 1–10. DOI: 10.1145/3139258.3139267. URL: https://hal.inria.fr/hal-01615142.

[6] S. Lunel, B. Boyer and J.-P. Talpin. 'Compositional proofs in differential dynamic logic dL'. In: *17th International Conference on Application of Concurrency to System Design*. Zaragoza, Spain, June 2017. URL: https://hal.inria.fr/hal-01615140.

[7] S. Lunel, S. Mitsch, B. Boyer and J.-P. Talpin. 'Parallel Composition and Modular Verification of Computer Controlled Systems in Differential Dynamic Logic'. In: *FM 2019 - 23rd International Symposium on Formal Methods*. Long version of an article accepted to the conference FM'19. Porto, Portugal, Oct. 2019, pp. 1–22. URL: https://hal.inria.fr/hal-02193642.

[8] S. Nakajima, J.-P. Talpin, M. Toyoshima and H. Yu. *Cyber-Physical System Design from an Architecture Analysis Viewpoint*. Communications of NII Shonan Meetings. Springer, Jan. 2017. DOI: 10.1007/978-981-10-4436-6. URL: https://hal.inria.fr/hal-01615144.

[9] V. Papailiopoulou, D. Potop-Butucaru, Y. Sorel, R. De Simone, L. Besnard and J.-P. Talpin. *From concurrent multi-clock programs to concurrent multi-threaded implementations*. Research Report RR-7577. INRIA, Mar. 2011, p. 22. URL: https://hal.inria.fr/inria-00578585.

[10] O. Sankur and J.-P. Talpin. 'An Abstraction Technique For Parameterized Model Checking of Leader Election Protocols: Application to FTSP'. In: *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Ed. by A. Legay and T. Margaria. Vol. 10206. Lecture Notes in Computer Science. Uppsala, Sweden: Springer Berlin Heidelberg, Apr. 2017, pp. 23–40. URL: https://hal.archives-ouvertes.fr/hal-01431472.

[11] H. Yu, J. Prashi, J.-P. Talpin, S. K. Shukla and S. Shiraishi. 'Model-Based Integration for Automotive Control Software'. In: *Digital Automation Conference*. ACM. San Francisco, United States, June 2015. URL: https://hal.inria.fr/hal-01148905.

11.2 Publications of the year

International journals

[12] D. Yue, V. Joloboff and F. Mallet. 'TRAP: trace runtime analysis of properties'. In: *Frontiers of Computer Science* 14.3 (June 2020), pp. 1–15. DOI: 10.1007/s11704-018-7217-7. URL: https://hal.inria.fr/hal-02402957.

Other scientific publications

[13] L. Besnard, T. Gautier and P. Le Guernic. *SIGNAL V4 – INRIA version: Reference Manual(revised working version)*. 1st Apr. 2020. URL: `https://hal.archives-ouvertes.fr/hal-02928452`.

[14] J.-J. Marty, L. Franceschino, J.-P. Talpin and N. Vazou. *LIO*: Low Level Information Flow Control in F**. 1st Apr. 2020. URL: `https://hal.inria.fr/hal-03137132`.