RESEARCH CENTRE

Rennes - Bretagne Atlantique

2021
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 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]

PhD Students

- Lucas Franceschino [Inria]
- Stephane Kastenbaum [Mitsubishi Electric, CIFRE]
- Shenghao Yuan [Inria]
- Liangcong Zhang [ENS Rennes, until Sep 2021]

Administrative Assistant

- Armelle Mozziconacci [CNRS]

2 Overall objectives

2.1 Introduction

An embedded architecture is an artifact f 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 a 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.

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 (irrational) variables, the models usually have to work with discrete time and approximate floating point variables.

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 system 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 reason 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-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 (Polychrony on Polarsys) and in the CCSL standard [1] 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,

[1] *Clock Constraints in UML/MARTE CCSL*. C. André, F. Mallet. RR-6540. INRIA, 2008

discrete or continuous. Some fundamental concepts 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 [2], 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 [3]. 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 capacity to modularly decompose systems which can be obtained using Abadi and Lamport's "*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 middleware 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.

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.

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.

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.

[2] *Abstract Refinement Types.* N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.
[3] *LTL types FRP.* A. Jeffrey. Programming Languages meets Program Verification.

The literature about real-time scheduling [4] 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 [5]. 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 [6] [7].

Abstract scheduling, or the use of abstraction and refinement techniques in scheduling borrowed to the theory of abstract interpretation [8] is a promising development toward tooled methodologies to orchestrate thousands of heterogeneous hardware/software blocks on modern CPS architectures (just 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 [9]. An invited paper [10] 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 constraints 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, a developer could 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

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

[4] *A survey of hard real-time scheduling for multiprocessor systems.* R. I. Davis and A. Burns. *ACM Computing Survey* 43(4), 2011.
[5] *Buffer minimization in EDF scheduling of data-flow graphs.* A. Bouakaz and J.-P. Talpin. LCTES, ACM, 2013.
[6] *ADFG for the synthesis of hard real-time applications.* A. Bouakaz, J.-P. Talpin, J. Vitek. ACSD, IEEE, June 2012.
[7] *Design of SCJ Level 1 Applications Using Affine Abstract Clocks.* A. Bouakaz and J.-P. Talpin. SCOPES, ACM, 2013.
[8] *La vérification de programmes par interprétation abstraite.* P. Cousot. Séminaire au Collège de France, 2008.
[9] Refinement types for system design. Jean-Pierre Talpin. FDL’18 keynote.
[10] Steps toward verified programming of embedded computing systems. Jean-Pierre Talpin, Jean-Joseph Marty, Deian Stefan, Shravan Nagarayan, Rajesh Gupta, DATE’18.

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

5.1 Awards

Shenghao Yuan participated to Hackatech 2021 from Sep. 30 to Oct. 2 and joined the JIMI team with the project to use AI techniques for automatically generating new (and hence copyright-free) melodies (Automatiser la création de mélodies). The JIMI project won the prize 'Challenge Ouest France' among three other projects awarded.

6 New software and platforms

Over the past two decades, projects ESPRESSO and TEA developed the following softwares 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., relations 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 scheduler for the actors.

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

URL: `http://polychrony.inria.fr/ADFG/`

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

Contact: 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

Contact: Loïc Besnard

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

Contact: Loïc Besnard

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 toolchain. The Polarsys download site for project POP opened in 2015.

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

Contact: Loïc Besnard

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

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. It deals with cyber-physical systems (CPS), which are assemblies of networked, heterogeneous, hardware, and software components sensing, evaluating, and actuating a physical environment. This heterogeneity induces complexity that makes CPSs challenging to model correctly. Since CPSs often have critical functions, it is however of utmost importance to formally verify them in order to provide the highest guarantees of safety. Faced with CPS complexity, model abstraction becomes paramount to make verification attainable. To this end, assume/guarantee contracts enable component model abstraction to support a sound, structured, and modular verification process. While abstractions of models by contracts are usually proved sound, none of the related contract frameworks themselves have, to the best of our knowledge, been formally proved correct so far. In this aim, we present the formalization of a generic assume/guarantee contract theory in the Coq proof assistant. We identify and prove theorems that ensure its correctness. Our theory is generic, or parametric, in that it can be instantiated and used with any given logic, in particular hybrid logics, in which highly complex cyber-physical systems can uniformly be described. This year, an article was published [15], presenting this parameterized formalization of a meta-theory of contract in the proof assistant Coq. We are now working on instantiating this theory with differential dynamic logic, a language to specify cyber-physical sytems. This instantiation provides a use case for the meta-theory of contracts. To this end, we are using CoqDL, a formalization of differential dynamic logic that was developed by the Logical Systems Lab of Carnegie Mellon University. We are currently exploring two compatible methods of instantiation. They will provide feedback and yield interesting properties.

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

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

The arrival of Xiong Xu with team TEA strenghtened our collaboration with Naijun Zhan's group at ISCAS, Beijing, in the context of Inria associate-project CONVEX.

Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow. We propose in [13] a framework combining AADL and Simulink/Stateflow, named AADL+S/S, to co-model CPSs and present a method to uniformly analyze and verify this combination. Compared with the existing co-modeling formalisms, AADL+S/S takes into account the three design aspects of CPSs, i.e., physics, architecture and functionalities, uniformly. We present a formal semantics of AADL+S/S by translating it to Hybrid Communicating Sequential Processes (HCSP), which yields a deductive verification framework for AADL+S/S models based on Hybrid Hoare Logic (HHL). A weak bisimulation between any AADL+S/S model and the translated HCSP process was proved, which guarantees the correctness of the translation. Finally, we developed a simulation tool for AADL+S/S, based on which the effectiveness of our approach is illustrated by the realistically scaled case study of an automatic cruise control system.

Semantic foundations for cyber-physical systems using higher-order UTP. We extended Hoare and He's Unifying Theories of Programming (UTP) with higher-order quantification and provide a formal semantics for modeling and verifying hybrid systems. Higher-order UTP (HUTP) provides a semantics foundation for cyber-physical systems (CPSs). It separates the concerns in CPS design into time, state and trace, which support the specification of discrete, real-time and continuous dynamics, concurrency and communication, and higher-order quantification. Within HUTP, we defined a calculus of normal hybrid designs to model, analyze, compose, refine and verify heterogeneous hybrid system models. In addition, we defined respective formal semantics for Hybrid Communicating Sequential Processes (HCSP) and Simulink using HUTP and justified the correctness of the translation from Simulink to HCSP by translation validation as an application of HUTP.

This work is to appear in the ACM Transactions on Software Engineering Methods, 48 pages, 2022: "Semantic foundations for cyber-physical systems using higher-order UTP". Xiong Xu, Jean-Pierre Talpin, Shuling Wang, Bohua Zhan, Naijun Zhan. .

Next : a session-typed hybrid pi-calculus. We are going to study the hybrid extension of Milner's pi-calculus to model, analyze and verify mobile hybrid systems. First, based on our previous work on higher-order unifying theories of programming (HUTP), we will investigate the mechanism of mobile hybrid systems, i.e., the HUTP extended with mobility. Then, we will focus on the operational semantics of the hybrid pi-calculus. The syntax should be expressive but succinct, and the operational semantics should be equivalent with the HUTP semantics of hybrid pi-calculus. Finally, we will build a session type system for hybrid pi-calculus and equip it with type checking capability.

7.3 Verified programming and secure integration of operating system libraries in RIOT-fp

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

Our project aims at the formal verification of safety and security properties for embedded operating system libraries in RIOT, focusing on its bootloader and a virtual machine to host runtime applications and services in the eBPF (extended Berkeley packet filter).

In 2021, Shenghao designed and verified RIOT's bootloader in Low* (a subset of F*). The model consists of the fletcher32 image checksum algorithm and an ARM assembly code sequence for booting. Both have been proved functionally correct and memory safe in F*. The work has been published [17].

In collaboration with teams Celtique (Frédéric Besson) and TRiBE (Koen Zandberg), FU Berlin (Emmanuel Baccelli) and Université de Lille (Samuel Hym), Shenghao implemented a verified rBPF virtual machine for RIOT's "femto-containers" in Coq. This work consists of 1) modelling an executable Coq interpreter of the rBPF virtual machine in Coq; 2) formally proving the rBPF interpreter satisfies sandboxing properties (software fault isolation) 3) defining a monadic Coq rBPF model and extract it to CompCert Clight ; 4) proving a forward simulation between the monadic Coq model and the Clight implementation.

This work is being submitted for publication: "End-to-end mechanised proof of an eBPF virtual machine for micro-controllers". Shenghao Yuan, Frédéric Besson, Jean-Pierre Talpin, 25 pages, 2022.

7.4 Verified Functional Programming of an Abstract Interpreter

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

We presented a verified abstract interpreter written in a functional style with the proof-oriented programming language F*. An abstract interpreter is a static analyser that implements algorithms from abstract interpretation. Abstract interpretation is a theory for copmputing sound approximations of program properties. However, most of abstract interpreters are themselves not verified and possibly subject to implementation errors, jeopardizing the expected mathematical guarantees: one slight bug might result in a bad inference. A solution to this problem is *verified programming*: to develop the program alongside with its correctness proof. In the scope of this work, this amounts to programming an abstract interpreter and proving its soundness formally, at the same time.

Such verified abstract interpreters already exist: their development and proofs were conducted with the proof assistant Coq. While their soundness guarantees are great, their require tremendous manual proof efforts, proofs that demand skill and expertise to understand, write and maintain, especially for abstract interpretation specialists (often non proof-assistant experts). For example, the Verasco abstract interpreter required about 17k lines of manual Coq proofs. By contrast with a handwritten proof, a Coq proof can be very verbose, and often does not convey a good intuition for the idea behind a proof. Leveraging the type system and automation of the proof-oriented programming language F*, we designed and presented a very small (only about 527 lines of codes) and self-contained abstract interpreter. Our paper [14] presents 95% of its source code. Its F* implementation resembles OCaml code and is very accessible to a public with no expertise in proof assistants: it required less than 40 lines of explicit proofs.

Our work highlights how our style of development can be helpful for writing accessible yet formally verified static analyser. As further work, we aim at enriching the target language of our analyser in order to progressively get closer to a real-world programming language such as C. It would also be very interesting to see how the lines of code and lines of proof ratio evolves while adding more complex abstract domains.

7.5 Using Polychrony for programming artificial neural networks

Participants: Thierry Gautier.

As part of a long-standing collaboration with Abdoulaye Gamatié, Senior Researcher at CNRS in the Microelectronics department of the LIRMM laboratory in Montpellier, we followed an exploratory work by a student from City University of Hong Kong, supervised by Abdoulaye Gamatié, on the use of the Signal language and the Polychrony toolset to build Convolutional Neural Networks and Recurrent Neural Networks. The internship resulted in a report by Jierui Liu entitled "Towards Synchronous Dataflow Programming of Artificial Neural Networks". In particular, the C code generated automatically from Polychrony was evaluated against the corresponding Python or hand-coded C descriptions of CNNs. It was generally observed that the Signal model allows to build a relevant CNN model for inference in a compositional manner. Though this work is preliminary, this study could open further perspectives on leveraging the design facilities associated with Signal for generating high-quality code for embedded systems dedicated to artificial intelligence.

7.6 ADFG: Affine data-flow graphs scheduler synthesis

Participants: 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. Our latest results were published in [16]

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 Toccata.

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: 2019 - 2022

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 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program

CONVEX

Title: Compositional Verification of Cyber-Physical Systems

Duration: 2018 - 2023

Coordinator: Naijun ZHAN (znj@ios.ac.cn)

Partners:

- Chinese Academy of Sciences

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.

Despite the pandemic, the CONVEX collaboration continued with weekly meeting across the year and did yield numerous exciting results in schedule verification, protocol verification and on modeling hybrid system abstractions using theorem provers[12, 18, 13].

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 Celtique and Fujitsu.

Summary: 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 proof-oriented programming techniques in Coq and F*, such as the actual bootloader of RIOT and its femto-containers: virtual machines isolating the execution of user applications and services using the eBPF virtual ISA.

10 Dissemination

10.1 Promoting scientific activities

Jean-Pierre Talpin served the program committee of SAMOS'21: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation.

10.2 Teaching - Supervision - Juries

Jean-Pierre Talpin served as rapporteur for the defense of Guillaume Dupont's Ph.D. Thesis at INP Toulouse: "Correct-by-construction design of hybrid systems based on refinement and proof".

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] K. Hu, T. Zhang, Y. Ding, J. Zhu and J.-P. Talpin. 'Verification of concurrent code from synchronous specifications'. In: *Science of Computer Programming* 206 (June 2021), pp. 1–19. DOI: 10.1016/j.scico.2021.102625. URL: https://hal.inria.fr/hal-03487784.

[13] X. Xu, S. Wang, B. Zhan, X. Jin, J.-P. Talpin and N. Zhan. 'Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow'. In: *Theoretical Computer Science* (Nov. 2021), pp. 1–25. DOI: 10.1016/j.tcs.2021.11.008. URL: https://hal.inria.fr/hal-03488546.

International peer-reviewed conferences

[14] L. Franceschino, J.-P. Talpin and D. Pichardie. 'Verified Functional Programming of an Abstract Interpreter'. In: SAS 2021 - 28th Static Analysis Symposium. Chicago, United States, 17th Oct. 2021, pp. 1–20. URL: https://hal.inria.fr/hal-03342997.

[15] S. Kastenbaum, B. Boyer and J.-P. Talpin. 'A Mechanically Verified Theory of Contracts'. In: ICTAC 2021 - 18th International Colloquium on Theoretical Aspects of Computing. Nur-Sultan, Kazakhstan, 20th Aug. 2021, pp. 134–151. DOI: 10.1007/978-3-030-85315-0_9. URL: https://hal.inria.fr/hal-03329311.

[16] H. N. Tran, A. Honorat, S. S. Bhattacharyya, J.-P. Talpin, T. Gautier and L. Besnard. 'A Framework for Fixed Priority Periodic Scheduling Synthesis from Synchronous Data-flow Graphs'. In: SAMOS XXI 2021 - 21st International Conference on embedded computer Systems: Architectures, MOdeling and Simulation. Virtual, France, 4th July 2021, pp. 1–12. URL: https://hal.inria.fr/hal-03488217.

[17] S. Yuan and J.-P. Talpin. 'Verified functional programming of an IoT operating system's bootloader'. In: MEMOCODE 2021 - 19th ACM-IEEE International Conference on Formal Methods and Models for System Design. Beijing, China: ACM, 20th Nov. 2021, pp. 1–16. URL: https://hal.inria.fr/hal-03343002.

[18] J. Zhu, K. Hu, M. Filali, J.-P. Bodeveix, J.-P. Talpin and H. Cao. 'Formal Simulation and Verification of Solidity contracts in Event-B'. In: COMPSAC 2021 - IEEE 45th Annual Computers, Software, and Applications Conference. Madrid, France: IEEE, 12th July 2021, pp. 1309–1314. DOI: 10.1109/COMPSAC51774.2021.00183. URL: https://hal.inria.fr/hal-03488328.