Activity Report 2014

Project-Team TEA

Time, Events and Architectures

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

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Embedded and Real-time Systems

Table of contents

Project-Team TEA

Keywords: Embedded Systems, Formal Methods, Time Modelling, Concurrency Theory, Programming Language, Program Analysis, Type Theory, Code Generation, Data-Flow Networks, Synchronous Modelling, Model-Driven Engineering, Architecture Modelling, Software Engineering

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

1. Members

Research Scientists

Jean-Pierre Talpin [Team leader, Inria, Senior Researcher, HdR]
Thierry Gautier [Inria, Researcher]
Paul Le Guernic [Inria, Senior Researcher]

Faculty Member

Adnan Bouakaz [Univ. Rennes I, until Sep 2014]

Engineers

Loïc Besnard [SED/CNRS, Senior Research Engineer]
Christophe Junke [Inria, granted by FUI project P]

PhD Students

Van-Chan Ngo [Inria, until Dec 2014, granted by ANR VeriSync project]
Ke Sun [Inria, until Oct 2014, granted by the Regional Council of Brittany]

Visiting Scientists

Tak Kuen John Koo [Univ. Rennes I, until July 2014]
Imré Frotier de La Messelière [PhD student with Mines ParisTech under co-supervision]

Administrative Assistant

Stéphanie Lemaile [Inria]

2. Overall Objectives

2.1. Introduction

An embedded architecture is an artefact of heterogeneous constituants 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. Unfortunately, modelling and programming formalisms that represent software, hardware and physics significantly alter this perception of time. Moreover, time reasoning in system design is usually isolated to a specific design problem: simulation, profiling, performance, scheduling, parallelisation, simulation. All these tasks would benefit from modularity and compositionally gained by globally reasoning about time. The aim of project-team TEA is to define a conceptually unified semantic framework for time reasoning in embedded system design, and to put it to practice by revisiting common analysis and synthesis issues in real-time system design with the compositionality gained from that formalisation.

2.2. Motivations

Electronic appliances, embedded systems, or, more generally, Cyber-Physical Systems, abbreviated CPS, are systems that comprise sensors, to sense physical data; electronics, to digitise the sensed physical information; computing units, to monitor the physical process; actuators, to activate devices reacting with the physical world; and, finally, a mean of communication, interconnecting these components.

As Lee acknowledges on his website [1], the term cyber-physical system (CPS) was introduced by Helen Gill at the NSF referring to the integration of computation and physical processes. In CPS, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. The principal challenges in system design lie in this perpetual interaction of software, hardware and physics.

Beyond the buzzword, a CPS is nothing new. In fact, it 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.

CPS safety is often critical for society. Their failure may entail threatening human beings life in many applications such as transportations (whether automotive, trains or airplanes), power distribution, medical equipment and tele-medicine. Whether or not life is threatened, failures may have huge economic impact (e.g. Toyota's defect car equipment). The development of reliable CPS has become a critical issue for the industry and society. Safety and security requirements must be satisfied by using strong validation tools. Requirements for quality of service, safety and security imply to have formally proved the required properties of the system before it is deployed.

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 modelling formalisms that have appeared in the past ten years with more or less success. The most successful are the *executable* models, models that can be 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. Development of concurrent and parallel systems has traditionally been clearly split in two different families. The first family is based on synchronous models, primarily targeting design of hardware circuits and/or embedded and reactive systems, often safety-critical. Esterel, Lustre, Signal and SCADE are examples of existing technologies of that nature, and in many places these have been connected with models of environments as required for CPS modelling. The second family addresses more loosely coupled systems, where communication between distributed entities is asynchronous by nature. Analysis of asynchronous systems has often greater complexity, because of the greater size of state spaces; process algebras such as CSP and CCS, or component models such as Fractal and GCM are more relevant here.

Large systems are increasingly mixing both types of concurrency. Large systems 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 synchronisation, determinism and functional correctness, scheduling optimality and calculation time predictability.

It should also be noted that CPSs are used essentially to monitor and control real-world processes, the dynamics of which are usually governed by well known 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 behaviour usually using real numbers (irrational) variables, the models usually have to work with discrete time and approximate floating point variables.

We consider that there are two key research directions, respectively, one for the theoretical basis underlying CPSs and one for the practical aspect of developing future applications that could be a major vector for scientific projects, developed in the next sections.

[1] *Cyber-physical systems*. E. A. Lee. Research Project, 2012. http://cyberphysicalsystems.org

2.3. Challenge

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 modelling and programming formalisms used to represent software, hardware and physic significantly alter this perception of time. In the model of the environment, the continuous evolution of time is represented by differential equations whose computerised resolution is inherently discrete. In hardware models, the system clock is an abstraction of the electrical behaviour of the circuit. It is usually further approximated by coarser-grain abstractions: register transfer level (RTL), transaction-level modelling (TLM) or system-level modelling.

In system design, time is usually abstracted to serve the purpose of one of many design problem: simulation, profiling, performance analysis, scheduling analysis, parallelisation, distribution, simulation, 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 extensions, multiple recurring instances of meaningful events may create as many dedicated logical clocks, on which to ground modelling and design practices.

Time reasoning is further complicated by the inadequacy of conventional programming models for modern hardware, such as Network-On-Chips. As pointed out by Edward Lee in his position paper [2], anyone experienced with multi-threaded programming can easily acknowledge the difficulty of designing and implementing concurrent software. Resolving concurrency, synchronisation, and coordination issues, and tackling the nondeterminism germane in multi-threaded software is extremely difficult. Ensuring software correctness not only with respect to its specification, but also with regards to target hardware and environment, is a necessary yet even more challenging task.

This challenge explains why the mitigation of time constraints arising from heterogeneous time models or domains is equally isolated to one specific design problem. For instance,

- scheduling analysis aims at reconciling software logical time with discrete hardware resources;
- desynchronisation aims at reconciling the synchronous abstraction of software concurrency with the asynchronous abstraction of a distributed architecture;
- virtual prototyping aims at simulating hardware events using software;
- hybrid simulation mixes software time and simulated physical time.

None of these problems demand system-level timed reasoning. All these issues are usually addressed in isolation. Yet, all would benefit from modularity and compositionally gained by coordinated time reasoning. Proper handling of time requires a precise semantic foundations and the establishment of formal correctness properties. It allows powerful analysis and error-proof verification of functional behaviours and quantitative characteristics.

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 behaviour, 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

[2] *The Problem with Threads*. E. A. Lee. Technical Report UCB/EECS-2006-1. UC Berkeley, 2006. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.html

continuous time with discrete logical time calls for proper discretisation 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 aim of project-team TEA is to develop formal calculi for reasoning about time in embedded system design. Equipped with these calculi, we will revisit typical problems and application in real-time system design, such as time determinism, memory ressources predictability, real-time scheduling, mixed-criticality and power management. Eventually, this will allow to prototype and deliver a tooled methodology for virtual prototyping embedded architectures.

3. Research Program

3.1. State of the Art

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 design abstraction focuses on the logic of interaction in reactive programs rather than their timed behaviour, allowing to secure functional correctness while remaining an intuitive programming model for embedded systems.

Maintaining the “synchronous hypothesis” on software at runtime, however, demands a quasi-synchronous model of execution (hardware or middleware) in order to be effectively implemented [3]. Strong software constraints to ensure functional correctness imply strong runtime restrictions and simple hardware. If we look at recent features found in synchronous programming languages such as Quartz [4], Lucid [5] departing from the simpler semantics of Esterel [6] and Lustre [7], we observe that all try to cope in a way or another with the availability of more general execution architectures: clock domains [8], pipelining [9], streaming [10]. Unfortunately, attempts to scale the simple "typed programming language" approach of the 90’s [11] to the above purpose hit inherent computational complexity limits. For example, a periodic clock operation like $0^{(1920*(1080-480))}\{0^{1200}1^{720}\}^{480}$ in Lucy-n (0^n means n zeros) yields an exponentially larger term [12]. This explains why team TEA opts for focusing on the semantics of time and concurrency in system design and on implementing the implied design methodologies using program analysis and abstract interpretation.

By contrast with a synchronous hypothesis, the polychronous MoCC implemented in the specification language Signal, available in the Eclipse project POP [13] and in the CCSL standard [14], is inherently capable of describing circuits and systems with multiple clocks.

The Eclipse project POP provides a tooled infrastructure to refine high-level specifications into real-time streaming applications or locally synchronous and globally asynchronous systems, through a series of model analysis and synthesis libraries. 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.

[3] *A protocol for loosely time-triggered architectures*. A. Benveniste et al. Embedded Software Conference. ACM, 2002

[4] The Averest System http://www.averest.org.

[5] Lucid synchrone http://www.di.ens.fr/~pouzet/lucid-synchrone.

[6] *The Esterel synchronous programming language*. G. Berry, G. Gonthier. Science of Computer Programming, v. 19(2). Elsevier, 1992.

[7] *The synchronous data flow programming language Lustre*. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D. Proceedings of the IEEE v. 79(9), 1991.

[8] *A formal semantics of clock refinement in imperative synchronous languages*. Gemünde, M., Brandt, J., Schneider, K. Application of Concurrency to System Design. IEEE Press, 2010.

[9] *Parallelism with futures in Lustre*. Cohen, A., Gérard, L., Pouzet, M. Embedded Software Conference. ACM, 2012.

[10] *N-synchronous Kahn networks*. Cohen, A., et al. Principles of Programming Languages. ACM, 2006.

[11] A. Benveniste et al. *The Synchronous Languages Twelve Years Later*. Proceedings of the IEEE v. 91(1), 2003.

[12] http://www.di.ens.fr/~guatto/slides_parkas_14_05_12.pdf, page 15.

[13] *Polychrony on POLARSYS (POP)*, an Eclipse project in the POLARSYS Industry Working Group, 2013. https://www.POLARSYS.org/projects/POLARSYS.pop

[14] *Clock Constraints in UML/MARTE CCSL*. C. André, F. Mallet. Technical Report RR-6540. Inria, 2008. http://hal.inria.fr/inria-00280941

3.2. Modelling Time

The elegant abstraction offered by the "synchronous hypothesis" [15] has translated in famous leitmotivs like *"computation takes no time"* and *"communication is instantaneous"* and contributed to the impact and commercial success of Esterel Studio [16] and SCADE [17].

Meanwhile, proposals and standards have appeared to push the technical boundaries of synchronous concurrency, in order to address a larger spectrum of concerns related to modern, heterogeneous, many-core architectures. The challenge becomes more largely about representing time in system design, alongside with many, so called, non-functional properties: cost, power, heat, speed, throughput.

One reference for the purpose of modelling timed hardware behaviour is PSL [18]. PSL is a formal specification language based on Kleene algebras that was originally designed to model regular hardware signal traces. The duality between automata and this formalism also makes it suitable to express requirements, formal properties and abstraction of program behaviours. It is widely used for modelling and verification of hardware systems.

A more recent reference of broader spectrum is CCSL [19], the clock constraints specification language of UML Marte. CCSL's core specification formalism is based on the Signal MoCC, it is synchronous and multi-clocked. Yet, CCSL supports extensions to model multi-rate, multi-periodic systems, that are adequate to represent hardware clocks, as well as asynchronous and continuous extensions (although largely unexploited in the related work). Another well-developed model is that of Ptolemy [20], which represents time as a first-class citizen alongside data carried by streams in the modelled system. It relates to the notion of PRET [21], (precision time machine) to support real-time simulation.

In the meantime, and from a totally different perspective, type theory has made considerable advances since the avent of effect systems [22] to formally represent formal properties alongside with values. Hybrid types [23] (linked to interface and contract theories), refinement types [24], value-dependant types, allow formal program properties, logical or temporal, to flow alongside with data-types during program analysis and verification. While a combination of all the above is yet unexplored, it offers an exciting venue for contributing in either/both of these fields with new theoretical developments on modelling time using principles of type theory.

3.3. Modelling Architectures

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. Models are hierarchically organised, so that each box can contain another system with its own set of boxes and connections between them. An architecture description language for embedded systems, for which timing and resource availability form an important part of the requirements, must in addition describe resources of the system platform, such as processors, memories, communication links, etc. Several architectural modelling languages for embedded systems have emerged in recent years, including the SAE AADL [25], SysML [26], UML MARTE [27].

[15] *The synchronous languages 12 years later*. A. Benveniste, et al. Proceedings of IEEE, 91(1), 2003.
[16] *Esterel Studio*, Sinfora. http://www.synfora.com/products/esterelStudio.html.
[17] *Scade System*, ANSYS. http://www.esterel-technologies.com/products/scade-system
[18] *IEEE Standard for Property Specification Language*. IEEE, 2005. http://dx.doi.org/10.1109/IEEESTD.2005.97780.
[19] *CCSL: specifying clock constraints with UML/MARTE*, OMG, 2008. http://www.omgmarte.org/node/66.
[20] *Ptolemy*, UC Berkeley. http://ptolemy.eecs.berkeley.edu.
[21] *Precision Timed Computation in Cyber-Physical Systems*. E. A. Lee and S. A. Edwards, 2007. http://ptolemy.eecs.berkeley.edu/publications/papers/07/PRET.
[22] *Polymorphic effect systems*. J. M. Lucassen , D. K. Gifford. Principles of Programming Languages. ACM, 1988.
[23] *Hybrid type checking*. K.W. Knowles and C. Flanagan. ACM Transactions on Programming languages and systems, 32(2). ACM, 2010
[24] *Abstract Refinement Types*. N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.
[25] *Architecture Analysis and Design Language*, AS-5506. SAE, 2004. http://standards.sae.org/as5506b
[26] *System Modelling Language*. OMG, 2007. http://www.omg.org/spec/SysML
[27] *UML Profile for MARTE*. OMG, 2009. http://www.omg.org/spec/MARTE

An architectural specification serves several important purposes. First, it breaks down a system model into manageable components 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, component interfaces allow us to avoid integration problems at the implementation phase. Connections between components, which specify how components affect each other, help propagate the effects of a change in one component to the 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 artefacts, 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 modelling time) are domain-specific languages such as Prelude [28] to model the real-time characteristics of embedded software architectures. Conversely, standard architecture description languages could be based on algebraic modelling tools, such as interface theories with the ECDAR tool [29].

3.4. Time Scheduling

Cyber-physical systems are reactive systems whose correctness not only depends on a deterministic behavior but also on timing predictability. The timing parameters of a CPS are requirements that arise from the system's specification (e.g. minimum throughput, maximum latency, deadlines) or timing properties of the physical and cyber-parts that restrict the CPS implementation. The design of a CPS must ensure that these timing requirements will be met, even in the worst-case scenario, through the different components and their timing properties.

3.4.1. Scheduling theory

Real-time scheduling theory provides tools for predicting the timing behaviour 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. In the real-time scheduling theory literature, many models of computation have been proposed to express such parallelism, for instance:

- Set of independent periodic, sporadic, or aperiodic tasks where each real-time task is generally characterised with some timing parameters: deadline, period, first start time, jitter, etc. The periodic and sporadic task models [30] are very well studied task models since they allow to analytically reason about the timing behaviour of tasks. More expressive task models [31] such as the multi-frame and the recurring real-time task models have also emerged.
- Task graph models [32] where precedence constraints among real-time tasks may exist.
- Data-flow graph models such as synchronous data-flow (SDF [33]) and cyclo-static dataflow (CSDF [34]. IEEE, 1996.) models where the set of tasks (also called actors) communicate with each other

[28] *The Prelude language*. LIFL and ONERA, 2012. http://www.lifl.fr/~forget/prelude.html
[29] *PyECDAR, timed games for timed specifications*. Inria, 2013. https://project.inria.fr/pyecdar
[30] *Scheduling algorithms for multiprogramming in a hard-real-time environment*. C. L. Liu and J. W. Layland. Journal of the ACM 20(1), 1973.
[31] *The digraph real-time task model*. M. Stigge, P. Ekberg, N. Guan, and W. Yi. Real-Time and Embedded Technology and Applications Symposium. IEEE, 2011.
[32] *Task graph scheduling using timed automata*. Y. Abdeddaïm, A. Kerbaa, and O. Maler. International Symposium on Parallel and Distributed Processing. IEEE, 2003.
[33] *Synchronous data-flow*. E. A. Lee and D. G. Messerschmitt. Proceedings of the IEEE, 1987.
[34] *Cycle-static dataflow*. G. Blisen, M. Engels, R. Lauwereins, and J. Peperstraete. Transactions on Signal Processing

through FIFO channels. When it fires, an actor consumes a predefined number of tokens from its inputs and produces a predefined number of tokens on its outputs. The scheduling problem is hence more complex since data dependencies must be satisfied.

The literature about real-time scheduling of sets of independent real-time tasks [35] provides very mature schedulability tests regarding many scheduling strategies, preemptive or non-preemptive scheduling, uniprocessor or multiprocessor scheduling, etc. Historically, real-time systems where scheduled by cyclic executives (i.e. static scheduling). However, since this approach produces rigid and difficult to maintain systems and handles only periodic tasks, the research community has proposed many dynamic scheduling algorithms, which can be classified as fixed-priority scheduling (e.g. rate-monotonic scheduling, deadline monotonic scheduling) and dynamic priority scheduling (e.g. earliest-deadline first scheduling, least laxity scheduling). Multiprocessor scheduling can be further classified as partitioned scheduling (each task is allocated to a processor and no migration is allowed), global scheduling (a single job can migrate to and execute on different processors), or hybrid.

Scheduling of data-flow graphs has also been extensively studied in the past decades. Static-periodic scheduling is the main scheduling approach, which consists in infinitely repeating a firing sequence of actors. This problem has been addressed with respect to many performance criteria: throughput maximisation [36], latency minimisation [37], buffer minimisation [38], code size minimisation [39], etc. Recently, real-time dynamic scheduling (fixed-priority and earliest-deadline first scheduling) of data-flow graphs has been addressed where actors are mapped to periodic real-time tasks and existing schedulability tests are adapted to synthesise the timing characteristics of actors [40] [41]

3.5. Virtual Prototyping

Virtual Prototyping is the technology of developing realistic simulators from models of a system under design; that is, an emulated device that captures most, if not all, of the required properties of the real system, based on its specifications. A virtual prototype should be run and tested like the real device. Ideally, the real application software would be run on the virtual prototyping platform and produce the same results as the real device with the same sequence of outputs and reported performance measurements. This may be true to some extent only. Some trade-offs have often to be made between the accuracy of the virtual prototype, and time to develop accurate models.

A virtual prototyping platform must include operating system or hardware emulation technology since the hardware functions must be simulated at least to a minimum extent in order to run the software and evaluate the design alternatives. The hardware simulation engine is a key component of a virtual prototyping platform, which makes it possible to run the application software and produce output that can be analysed by other tools. Because electronic design tools (EDAs) simulate the hardware in every detail, it is possible to verify that the circuit operates properly and also to measure how many clock cycles will be required to achieve an operation. But because they simulate very low-level operations, simulation is much too slow to be usable for virtual prototyping. The authors of the FAST system [42] and SocLib project reports [43] speed-ups with a

[35] *A survey of hard real-time scheduling for multiprocessor systems*. R. I. Davis and A. Burns. *ACM Computing Survey* 43(4), 2011.

[36] *Throughput analysis of synchronous data-flow graphs*. Ghamarian, A.H. et al. Application of Concurrency to System Design. IEEE, 2006

[37] *Latency minimization for synchronous data flow graphs*. A. H. Ghamarian, et al. Conference on Digital System Design Architectures, Methods and Tools. Euromicro, 2007.

[38] *Minimal memory schedules for data-flow networks*. M. Cubric and P. Panangaden. International Conference on Concurrency Theory. Springer, 1993.

[39] *Looped schedules for dataflow descriptions of multirate signal processing algorithms*. S. S. Bhattacharyya and E. A. Lee. Journal of Formal Methods in System Design. Kluwer, 1994.

[40] *Affine data-flow graphs for the synthesis of hard real-time applications*. A. Bouakaz, J.-P. Talpin, and J. Vitek. International Conference on Application of Concurrency to System Design. IEEE Press, 2012.

[41] *Hard-real-time scheduling of data-dependent tasks in embedded streaming applications*. M. Bamakhrama and T. Stefanov. International Conference on Embedded Software. ACM, 2011.

[42] *The fast methodology for high-speed SOC simulation.* D. Chiou, et al. International conference on Computer-aided design. IEEE, 2007.

factor of several hundreds in a comparison between their cycle accurate simulator and their virtual prototyping framework. A factor of the order of 100 times faster than EDA tools is required for virtual prototyping.

In order to speed-up simulation time, the virtual prototype must trade-off with something. Depending upon the application designers goals, one may be interested in trading some loss of accuracy in exchange for simulation speed, which leads to constructing simulation models that focus on some design aspects and provide abstraction of others. A simulation model can provide an abstraction of the simulated hardware in three directions:

- *Computation abstraction*. A hardware component computes a high level function by carrying out a series of small steps executed by composing logical gates. In a virtual prototyping environment, it is often possible to compute the high level function directly by using the available computing resources on the simulation host machine, thus abstracting the hardware function.
- *Communication abstraction*. Hardware components communicate together using some wiring, and some protocol to transmit the data. Simulation of the communication and the particular protocol may be irrelevant for the purpose of virtual prototyping: communication can be abstracted into higher level data transmission functions.
- *Timing Abstraction*. In a cycle accurate simulator, there are multiple simulation tasks, and each task makes some progress on each clock cycle, but this is slowing down the simulation. In a virtual prototyping experiment, one may not need to so precise timing information: coarser time abstractions can be defined allowing for faster simulation.

The cornerstone of a virtual prototyping platform is the component that simulates the processor(s) of the platform, and its associated peripherals. Such simulation can be *static* or *dynamic*.

3.6. Research Objectives

The challenges addressed by team TEA support the claim that sound Cyber-Physical System design (including embedded, reactive, and concurrent systems altogether) should consider (logical, formal) time modelling as a central aspect.

In this aim, architectural specifications found in software engineering are a natural focal point to start from. Architecture descriptions organise a system model into manageable components, establish clear interfaces between them, and help correct integration of these components during system design.

The definition of a formal design methodology to support the heterogeneous modelling of time in architecture descriptions demands the elaboration of sound mathematical foundations and the development of formal calculi methods to instrument them that constitute the research program of team TEA.

3.6.1. Objective n. 1 – Semantics and specification of time in system design

Time systems. To mitigate and generalise algebraic representations of time, we propose to introduce the paradigm of "time system" (type systems to represent time). 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. Instances of the concept of time system we envision are the clock calculi found in data-flow synchronous languages like Signal, Lustre and its different incarnations. All are bound to a particular model of time.

To gain generality and compositionality, we wish to proceed from recent developments on hybrid types [44] (linked to interface and contract theories), refinement types [45], value-dependant type [46] theories, to formally define a time system.

[43] *Using binary translation in event driven simulation for fast and flexible MPSOC simulation.* M. Gligor, N. Fournel, and F. Pétrot. In CODES+ISSS, IEEE, 2009.

[44] *Hybrid type checking*. K.W. Knowles and C. Flanagan. ACM Transactions on Programming languages and systems, 32(2). ACM, 2010

[45] *Abstract Refinement Types*. N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.

[46] *Secure distributed programming with value-dependent types*. N. Swamy, et al. International Conference on Functional Programming. Springer, 2011.

The principle of these type systems is to allow data-types inferred in the program with properties, possibly temporal, pertaining, for instance, to the algebraic domain on their value, or any algebraic property related to its computation: effect, memory usage [47], pre-post condition, value-range, cost, speed, time.

In the quest of an appropriate algebra for time, we are studying both the CCSL and PSL standards and, more generally, Kleene algebras [48] which offer greater expressivity in the prospect of timed specification as well as refinement checking and verification [49] [50].

Being grounded on type and domain theories, a time system can naturally be equipped with program analysis techniques based on type inference (for data-type inference) or abstract interpretation (for program properties inference) [51]. We intend to use and learn from existing open-source implementations in this field of research [52] in order to prototype our solution.

Relating time systems. Just as a time system formally represents the timed behaviour of a given component, timing relations (abstraction and refinement) represent interaction among components. Logically, their specification should be the role of a module system, and verifying their conformance that of a module checking algorithm.

Scalability and compositionality dictate the use of assume-guarantee reasoning, as found in interface automata and contract algebra, in order to facilitate composition by behavioural sub-typing, in the spirit of the (static) contract-based formalism proposed by Passerone et al. [53] [54].

To further elaborate a formal verification approach, we will additionally consider notions of refinement calculi based on temporal logic [55], in order to possibly extend our interface and contract theories with liveness properties. The definition of a module/interface for timed architectures should hence proceed directly from the definition of its time system, using mostly existing theoretical results on the matter of module systems, interface and contract theories.

Conformance of time relations. Verification problems encompassing heterogeneously timed specifications are common and of great variety: checking correctness between abstract and concrete time models relates to desynchronisation (from synchrony to asynchrony) and scheduling analysis (from synchrony 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 not only to inject time models one into the other (by abstract interpretation, using refinement calculi), to compare time abstractions one another (using simulation, refinement, bisimulation, equivalence relations) but also to prove more specific properties (synchronisation, determinism, endochrony).

---

[47] *Region-based memory management*. Tofte, M., Talpin, J.-P. Information and Computation, 132(2). Academic Press, 1997.

[48] *Automated reasoning in Kleene algebra*. P. Höfner and G. Struth. Conference on Automated Reasoning. Springer, 2007.

[49] *Algebraic Verification Method for SEREs Properties via Groebner Bases Approaches*. N. Zhou, J. Wu, X. Gao. Journal of Applied Mathematics. Hindawi, 2013

[50] *From monadic logic to PSL*. M. Y. Vardi. Pillars of Computer Science, 2008.

[51] *Timed polyhedra analysis for synchronous languages*. Besson, F., Jensen, T., Talpin, J.-P. Static Analysis Symposium. Springer, 1999.

[52] The Microsoft F* project, https://research.microsoft.com/en-us/projects/fstar.

[53] *A contract-based formalism for the specification of heterogeneous systems*. L. Benvenistu, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, C. Sofronis. Forum on design languages, 2008

[54] *Moving from Specifications to Contracts in Component-Based Design.* S. Bauer, A. David, R. Hennicker, K. Larsen, A. Legay, U. Nyman, A. Wasowski. Fundamental Aspects in Software Engineering. Springer, 2012

[55] *Refinement Calculus: A Systematic Introduction*. R.J. Back, J. von Wright. Springer, 1998.

In the spirit of our recent work developing an abstract scheduling theory, we want to develop a method of abstract interpretation [56] to reason about the abstraction and refinement of heterogeneous timed specifications in the aim of checking their conformance. A source of inspiration in that prospect is the notion of contract abstraction [57]. To this end, we plan to use SAT-SMT solving techniques to check conformance of abstracted time constraints, in a way which we previously experienced with the automated code generation validation of Polychrony [58] [59] [60].

To check conformance between heterogeneously timed specifications, we will consider variants of the abstract interpretation framework proposed by Bertrane et al. [61] to inject properties from one time domain into another, be it continuous [62] or discrete [63].

This will for instance enable the possibility of verifying cross-domain properties, e.g. cost v.s. power v.s. performance v.s. software mapping. This will allow to formalise intuitions such as that this typical inter-domain constraint: the cost of a system has an impact on the system's controlability; and allow to formally explain why: lower cost means hardware with lower performances, which means longer WCRTs, which means longer end-to-end latency, which may result in a response-time longer than controllability limits. This particular topic (which we could call cross-domain conformance checking) has not been studied in the related literature (on contract-based design, for instance), and could be based on both abstraction techniques, e.g. linear abstractions, or morphisms between domains or even discrete relations, e.g. a simple catalog or "price list" relating price and performance for a data-base of hardware components.

3.6.2. Objective n. 2 – A standard for modelling time in system design

A second objective, to be developed in parallel and synergy to objective n. 1, is the definition of an architecture-specific specification formalism, that would serve as semantic foundation, structure and repository for tooling a component-based design methodology with semantic analysis, to synthesise component interfaces, and formal methods, to verify specified requirements.

In project TEA, it will take form by the definition and tooling of a time annex for the AADL standard, based on the theory developed in objective n. 1. The aim of the AADL time annex is to formalise the logical and physical timing properties of architecture models and represent them as constraints expressed using regular grammars (like in PSL), or using the process calculus of CCSL.

This is an objective reminiscent and in direct application of the principle of time system (objective n.1). We not only want to model time in the heterogeneous logical and physical constituents in an AADL specification, but relate them, and verify the correctness of their composition.

Our aim is to start from the modelling standards AADL and CCSL to define a standard for time in system design. Our contribution will be formalised by a timing annex for the AADL and tools collaboratively developed to support its use. Our first milestone in this prospect is a report [64] of recommendations accepted by the AADL committee. Our next step, the submission of a time annex by team TEA at the SAE consortium, will

[56] *La vérification de programmes par interprétation abstraite*. P. Cousot. Séminaire au Collège de France, 2008.
[57] *Compositional contract abstraction for system design*. A. Benveniste, D. Nickovic, T. Henzinger.
[58] *Efficient deadlock detection for polychronous data-flow specifications*. C. Ngo, J.-P. Talpin, T. Gautier. Electronic System Level Synthesis Conference (ESLSYN'14). IEEE, 2014.
[59] *Formal verification of synchronous data-flow program transformations toward certified compilation*. V.-C. Ngo, J.-P. Talpin, Gautier, P. Le Guernic, L. Besnard. Frontiers of Computer Systems. Springer, 2013.
[60] *Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction*. P. Feautrier, A. Gamatié and L. Gonnord. Journal of Computing, 1(4). Computer Society of India, 2012.
[61] *Temporal Abstract Domains*. J. Bertrane. International Conference on Engineering of Complex Computer Systems. IEEE, 2011
[62] *Abstract Interpretation of the Physical Inputs of Embedded Programs*. O. Bouissou, M. Martel. Verification, Model Checking, and Abstract Interpretation. LNCS 4905, Springer, 2008
[63] *Proving the Properties of Communicating Imperfectly-Clocked Synchronous Systems*. J. Bertrane. Static Analysis Symposium. Springer, 2006
[64] *"Logically timed specifications in the AADL – Recommendations to the SAE committee on AADL*. L. Besnard, E. Borde, P. Dissaux, T. Gautier, P. Le Guernic, J.-P. Talpin, H. Yu. Inria Technical Report n.446, 2014.

employ the principles exposed in objective n.1 in order to formally define a modular and scalable specification formalism to specify heterogeneous timing constraints in the AADL.

Then, the specification of timing relations between AADL objects will be made explicit by contracts. Together with these contracts, we will then formally define abstraction and refinement relation in order to inject properties assumed by one component into the time model guaranteed by another, and vice versa. Lastly, conformance-checking abstracted contracts will be supported by state-of-the-art verification tools. This all will define a design methodology for time in the AADL, and our very last step will be to tool this methodology and provide a reference implementation.

3.6.3. Objective n. 3 – Applications to real-time scheduling

As a prime application of formal methods for interacting time models, scheduling thousands of program blocks or modules found on modern embedded architecture poses a challenging problem. It simply defies known bounds of complexity theory in the field. It is an issue that requires a particular address, because it would find direct industrial impact in present collaborative projects in which we are involved.

One recent milestone in the prospect of large-scale scheduling is the development of abstract affine scheduling [65]. It consists, first, of approximating threads communication patterns in Safety-Critical Java using cyclo-static data-flow graphs and affine functions. Then, it uses state of the art ILP techniques to find optimal schedules and concretise them as real-time schedules for Safety Critical Java programs [66] [67]

To develop the underlying theory of this promising research topic, we first need to deepen the theoretical foundation to establish links between scheduling analysis and abstraction interpretation [68].

The theory of time system developed in objective n.1 offers the ideal framework to pursue this development. It amounts to representing scheduling constraints, inferred from programs, as types. It allows to formalise the target time model of the scheduler (the architecture, its middle-ware, its real-time system) and defines the basic concepts to verify assumptions made in one with promises offered by the other: contract verification or, in this case, synthesis. Objective n.3 is hence defined as a direct application of objective n.1.

3.6.4. Objective n. 4 – Applications to virtual prototyping

A solution usually adopted to handle time in virtual prototyping is to manage hierarchical time scales, use component abstractions where possible to gain performance, use refinement to gain accuracy where needed. Localised time abstraction may not only yield faster simulation, but facilitate also verification and synthesis (e.g. synchronous abstractions of physically distributed systems). Such an approach requires computations and communications to be harmoniously discretised and abstracted from originally heterogeneous viewpoints onto a structuring, articulating, pivot model, for concerted reasoning about time and scheduling of events in a way that ensures global system specification correctness.

Just as model checking usually employs goal-directed abstraction techniques, in order to approximate parts of the model that are not in the path of the property to check, we plan to equivalently define, possibly semi-automate, abstraction techniques to approximate the time model of system components that do not directly influence timing properties to evaluate.

In the short term these component models could be based on libraries of predefined models of different levels of abstractions. Such abstractions are common in large programming workbench for hardware modelling, such as SystemC, but less so, because of the engineering required, for virtual prototyping platforms. Additionally, the level of abstraction required to simulate components could simply (and best) be specified manually by annotating the architecture specification.

[65] *Buffer minimization in earliest-deadline first scheduling of dataflow graphs*. A. Bouakaz and J.-P. Talpin. Conference on Languages, Compilers and Tools for Embedded Systems. ACM, June 2013.

[66] *Affine data-flow graphs for the synthesis of hard real-time applications*. A. Bouakaz, J.-P. Talpin, and J. Vitek. Application of Concurrency to System Design. IEEE Press, June 2012.

[67] *Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks*. A. Bouakaz and J.-P. Talpin. International Workshop on Software and Compilers for Embedded Systems. ACM, June 2013.

[68] *Abstraction-Refinement for Priority-Driven Scheduling of Static Dataflow Graphs*. Submitted for publication, 2014.

The approach of team TEA provides an additional ingredient in the form of rich component interfaces. It therefore dictates to further investigate the combined use of conventional virtual prototyping libraries, defined as executable abstractions of real hardware, with executable component simulators synthesised from rich interface specifications (using, e.g., conventional compiling techniques used for synchronous programs).

Just as virtual integration consists of synthesising the verification model of an architecture specification, virtual prototyping can be seen as synthesising an executable simulator from a model in, e.g., the spirit of the A-350 DMS case study that was realised by team ESPRESSO in the frame of Artemisia project CESAR [69].

4. Application Domains

4.1. Application Domains

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, as demonstrated from this section to section 8. Yet, the research focus on time in team TEA is central in any aspect of, cyber-physical, embedded system design in automotive, music synthesis, signal processing, software radio, circuit and system on a chip design; many application domains which, should more collaborators join the team, would definitely be worth investigating.

Nonetheless, the application domains of our two direct collaborations with industry, avionics with Thales and automotive Toyota, are perfectly in line with the research objectives of team TEA and will allow us to quickly stream our theoretical results onto software and standards, which we will continue to distribute in open-source.

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

Together with the importance of open-source software, one of these project, the FUI Project P, demonstrated that a centralised 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 centralised model for importing discrete-continuous models onto a simplified implementation of SIMULINK: P models. Certified code generators would then be developed 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 Modeller on top of SCADE, which uses the more model-engineering flavoured 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 synthesised, naturally appears better suited for the purpose of virtual prototyping. RT-Builder, developed by TNI, was industrially proven and deployed for that purpose at Peugeot. It served to develop the virtual platform simulating all onboard electronics of PSA cars. This ‘hardware in the loop” simulator was used to test equipments supplied by other manufacturers with respect to virtual cars. In the avent of the related automotive standard, RT-Builder then became AUTOSAR-Builder.

[69] *System-level co-simulation of integrated avionics using polychrony*. Yu, H., Ma, Y., Glouche, Y., Talpin, J.-P., Besnard, L., Gautier, T., Le Guernic, P., Toom, A., and Laurent, O. ACM Symposium on Applied Computing. ACM, 2011.

RT-Builder is the commercial implementation of Signal, whose industrial transfer with TNI was realised in the 90s by Paul Le Guernic and Albert Benveniste. As its actual industry usage has demonstrated, it is clear that the synchronous multi-clocked, or polychronous MoCC of Signal is an appropriate semantic core for the design of embedded software architectures.

5. New Software and Platforms

5.1. The Eclipse project POP

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

The distribution of project POP [70] is a major achievement of the ESPRESSO project. 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.

Technically, project POP is composed of the Polychrony toolset, under GPL license, and its Eclipse framework, under EPL license.

The Polychrony toolset. The Polychrony toolset is an Open Source development environment for critical/embedded systems. It is based on Signal, a real-time polychronous dataflow 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 toolset provides a formal framework:

- to validate a design at different levels, by the way of formal verification and/or simulation,
- to refine descriptions in a top-down approach,
- to abstract properties needed for black-box composition,
- to assemble heterogeneous predefined components (bottom-up with COTS),
- to generate executable code for various architectures.

The Polychrony toolset 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. The Signal toolbox can be installed without other components. The Signal toolbox is distributed under GPL V2 license.
- The Signal GUI, a Graphical User Interface to the Signal toolbox (editor + interactive access to compiling functionalities). The Signal GUI is distributed under GPL V2 license.
- The SME/SSME platform, a front-end to the Signal toolbox in the Eclipse environment. The SME/SSME platform is distributed under EPL license.
- GCCst, a back-end to GCC that generates Signal programs (not yet available for download).

[70] *Polychrony on POLARSYS (POP)*, an Eclipse project in the POLARSYS Industry Working Group, 2013. https://www.POLARSYS.org/projects/POLARSYS.pop

In 2013, to be able to use the Signal GUI both as a specific tool and as a graphical view under Eclipse, the code of the Signal GUI has been restructured in three parts: a common part used by both tools (28 classes), a specific part for the Signal GUI (2 classes), a specific part for Eclipse (2 classes). Such a structuration facilitates the maintenance of the products.

The Polychrony toolset also provides:

- libraries of Signal programs,
- a set of Signal program examples,
- user oriented and implementation documentations,
- facilities to generate new versions.

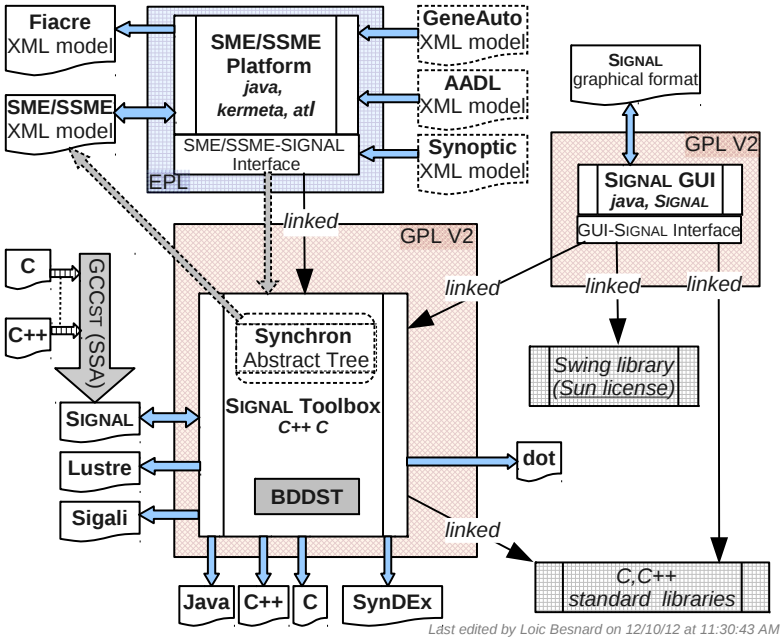

Figure 1. The Polychrony toolset high-level architecture

Dassault Systèmes, supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects.

As part of its open-source release, the Polychrony toolset not only comprises source code libraries but also an important corpus of structured documentation, whose aim is not only to document each functionality and service, but also to help a potential developer to package a subset of these functionalities and services, and adapt them to developing a new application-specific tool: a new language front-end, a new back-end compiler. This multi-scale, multi-purpose documentation aims to provide different views of the software, from a high-level structural view to low-level descriptions of basic modules. It supports a distribution of the software "by apartment" (a functionality or a set of functionalities) intended for developers who would only be interested by part of the services of the toolset.

The Eclipse POP Framework. We have developed a meta-model and interactive editor of Polychrony in Eclipse. Signal-Meta is the meta-model of the Signal language implemented with Eclipse/Ecore. It describes all syntactic elements specified in [71]: 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) within an Eclipse-based development toolchain. 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.

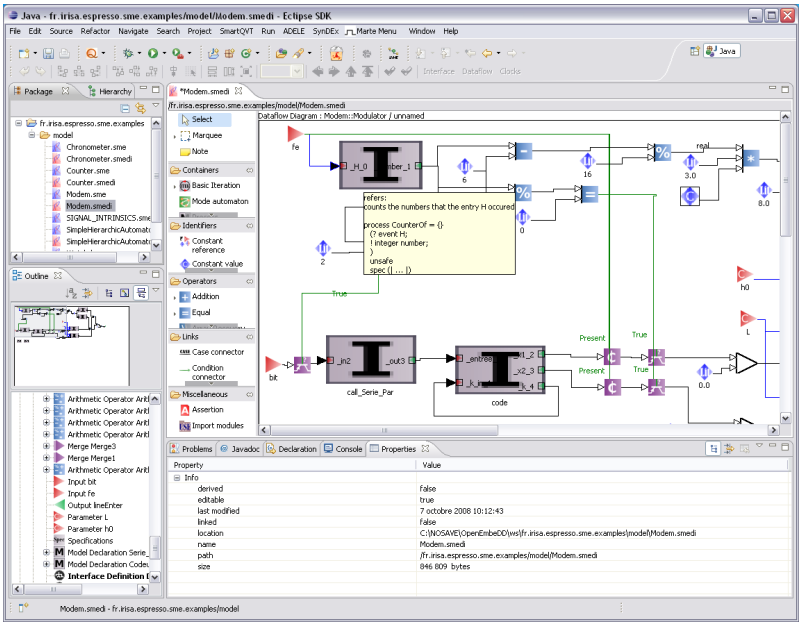

Figure 2. The Eclipse POP Environment

It also provides a graphical modelling framework allowing to design applications using a component-based approach. Application architectures can be easily described by just selecting components via drag and drop, creating some connections between them and specifying their parameters as component attributes. Using the modelling facilities provided with the Topcased framework, we have created a graphical environment for Polychrony called SME (Signal-Meta under Eclipse). To highlight the different parts of the modelling in Signal, we split the modelling of a Signal process in three diagrams: one to model the interface of the process, one to model the computation (or dataflow) part, and one to model all explicit clock relations and dependences. The SME environment is available through the ESPRESSO update site [72]. A new meta-model of Signal, called SSME (Syntactic Signal-Meta under Eclipse), closer to the Signal abstract syntax, has been defined and integrated in the Polychrony toolset.

It should be noted that the Eclipse Foundation does not host code under GPL license. So, the Signal toolbox useful to compile Signal code from Eclipse is hosted on our web server. For this reason, the building of the Signal toolbox, previously managed under Eclipse, has now been exported. The interface of the Signal toolbox for Eclipse is now managed using the CMake tool like the Signal toolbox and the Signal GUI.

5.2. Integrated Modular Avionics design using Polychrony

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

[71] *SIGNAL V4-Inria version: Reference Manual*. Besnard, L., Gautier, T. and Le Guernic, P. http://www.irisa.fr/espresso/Polychrony, 2009

[72] *Polychrony Update Site for Eclipse plug-ins*. http://www.irisa.fr/espresso/Polychrony/update, 2009.

The Apex interface, defined in the ARINC standard [73], provides an avionics application software with the set of basic services to access the operating-system and other system-specific resources. Its definition relies on the Integrated Modular Avionics approach (IMA). A main feature in an IMA architecture is that several avionics applications (possibly with different critical levels) can be hosted on a single, shared computer system. Of course, a critical issue is to ensure safe allocation of shared computer resources in order to prevent fault propagations from one hosted application to another. This is addressed through a functional partitioning of the applications with respect to available time and memory resources. The allocation unit that results from this decomposition is the *partition*.

A partition is composed of *processes* which represent the executive units (an ARINC partition/process is akin to a Unix process/task). When a partition is activated, its owned processes run concurrently to perform the functions associated with the partition. The process scheduling policy is priority preemptive. Each partition is allocated to a processor for a fixed time window within a major time frame maintained by the operating system. Suitable mechanisms and devices are provided for communication and synchronization between processes (e.g. *buffer*, *event*, *semaphore*) and partitions (e.g. *ports* and *channels*). The specification of the ARINC 651-653 services in Signal [4] is now part of the Polychrony distribution and offers a complete implementation of the Apex communication, synchronization, process management and partitioning services. Its Signal implementation consists of a library of generic, parameterizable Signal modules.

5.3. Safety-Critical Java Level 1 Code generation from Dataflow Graph Specifications

Participants: Adnan Bouakaz, Thierry Gautier, Jean-Pierre Talpin.

We have proposed a dataflow design model [2] of SCJ/L1 applications [74] in which handlers (periodic and aperiodic actors) communicate only through lock-free channels. Hence, each mission is modeled as a dataflow graph. The presented dataflow design model comes with a development tool integrated in the Eclipse IDE for easing the development of SCJ/L1 applications and enforcing the restrictions imposed by the design model. It consists of a GMF editor where applications are designed graphically and timing and buffering parameters can be synthesized. Indeed, abstract affine scheduling is first applied on the dataflow subgraph, 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 fixed-priority schedulability analysis (i.e., synthesis of timing and scheduling parameters of actors) considers both periodic and aperiodic actors.

Through a model-to-text transformation, using Acceleo, the SCJ code for missions, interfaces of handlers, and the mission sequencer is automatically generated in addition to the annotations needed by the memory checker. Channels are implemented as cyclic arrays or cyclical asynchronous buffers; and a fixed amount of memory is hence reused to store the infinite streams of tokens. The user must provide the SCJ code of all the `handleAsyncEvent()` methods. We have integrated the SCJ memory checker [75] in our tool so that potential dangling pointers can be highlighted at compile-time. To enhance functional determinism, we would like to develop an ownership type system to ensure that actors are strongly isolated and communicate only through buffers.

6. New Results

6.1. Highlights of the Year

This year's effort has been mainly devoted to the successful creation of project-team TEA and the definition of its new research perspective on Time, Events and Architectures in CPS design.

[73] *ARINC Report 651-1: Design Guidance for Integrated Modular Avionics*. Airlines Electronic Engineering Committee, 1997

[74] *Safety critical Java technology specification*. JSR-302, Year = 2010

[75] *Static checking of safety critical Java annotations*. Tang, D. Plsek, A. and Vitek, J. International Workshop on Java Technologies for Real-Time and Embedded Systems, 2010

The SAE committee on the AADL adopted our recommendations to implement a timed and synchronous behavioural annex [13], [11] for standardisation [20]. The specification and reference implementation of this revised behavioral annex will be the focus of most our attention next year.

Adnan Bouakaz published and implemented more of the original results from his PhD. work on abstract affine scheduling [14], [15].

6.2. Priority-Driven Scheduling of Static Dataflow Graphs through Time Abstraction

Participants: Adnan Bouakaz, Thierry Gautier, Jean-Pierre Talpin.

Static dataflow graph models, such as SDF [76] and CSDF [77], are widely used to design concurrent real-time streaming applications due to their inherent functional determinism and predictable performances. The state of the art usually advocates static-periodic scheduling of dataflow graphs over dynamic scheduling. Through the past decades, a considerable effort has been made to solve this problem [78]. Ensuring boundedness and liveness is the essence of the proposed algorithms in addition to optimizing some nonfunctional performance metrics (e.g. buffer minimization, throughput maximization, etc.).

Nowadays real-time streaming applications on MPSoCs are increasingly complex; and runtime systems are more needed to handle resource sharing, task priorities, etc. Therefore, recent works [79] [80] [81] are considering dynamic scheduling policies (e.g. earliest-deadline first scheduling, deadline monotonic scheduling, etc.) for dataflow graphs. The main motivations of these works are: (1) most existing real-time operating systems support such scheduling policies; (2) applicability of the existing schedulability theory [82] [83]; and (3) with such dynamic approach, multiple and independent applications, each designed as a dataflow graph, can run concurrently on the same platform.

Our work [84] [85] [14], [15] proposes a sequence-based framework in which a large class of priority-driven schedules can be uniformly expressed and analyzed. Infinite sequences are used to describe the dataflow graphs (e.g. rate sequences, execution time sequences) and both concrete and abstract schedules (e.g. activation clocks, priority sequences, activation relations, etc.). The framework can be then easily adapted for specific needs (e.g. affine scheduling). Our schedule construction approach is based on two steps. The first step consists in computing an abstract schedule which consists of a set of priority sequences, processor allocation sequences, and activation relations. An activation relation between two actors describes the relative order of their activations, and hence allows us to compute safe sizes of channels between them using worst-case overflow/underflow scenarios. This step must satisfy some correctness constraints such as consistency and exclusion of overflow and underflow exceptions. Once the best abstract schedule (w.r.t. to a performance metric) is computed, the schedule is refined by computing the actual periods and phases that ensure schedulability on the target architecture.

[76] *Synchronous data-flow*. E. A. Lee and D. G. Messerschmitt. Proceedings of the IEEE, 1987.
[77] *Cycle-static data-flow*. Blisen, G. and Engels, M. and Lauwereins, R. and Peperstraete, Transactions on Signal Processing, v.2. 1996.
[78] *Software synthesis from dataflow graphs*. Battacharyya, S. and Lee, E. and Murthy, P. Kluwer Academic Publishers, 1996.
[79] *Affine Data-Flow Graphs for the Synthesis of Hard Real-Time Applications*. International Conference on Application of Concurrency to System Design. IEEE Press, 2012
[80] *Temporal analysis flow based on an enabling rate characterization for multi-rate applications executed on MPSoCs with non-starvation-free schedulers*. Hausmans, J., et al. International Workshop on Software and Compilers for Embedded Systems, 2014.
[81] *Hard-real-time scheduling of data-dependent tasks in embedded streaming applications.* Bamakhrama, M. and Stefanov, T. Embedded Systems Conference. ACM, 2011
[82] *Real time scheduling theory: a historical perspective*. Sha, L. et al. Real-Time Systems Conference. IEEE, 2004
[83] *A survey of hard real-time scheduling for multiprocessor systems.* Davis, R. and Burns, A. ACM Computing Surveys, v. 4, 2011
[84] *Buffer Minimization in Earliest-First Scheduling of Dataflow Graphs*. A. Bouakaz, J-P. Talpin. ACM conference on languages, compilers and tools for embedded systems. ACM Press, 2013.
[85] *Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks*. A. Bouakaz, J-P. Talpin. International Workshop on Software and Compilers for Embedded Systems, 2013.

6.3. Formal Verification of a Synchronous Data-flow Compiler: from Signal to C

Participants: Van-Chan Ngo, Jean-Pierre Talpin, Thierry Gautier, Paul Le Guernic, Loïc Besnard.

Translation validation [86] [87] is a technique that attempts to verify that program transformations preserve the program semantics. It is obvious to prove globally that the source program and its final compiled program have the same semantics. However, we believe that a better approach is to separate concerns and prove each analysis and transformation stage separately with respect to ad-hoc data-structures to carry the semantic information relevant to that phase.

In the case of the Signal compiler [1], [7][12], the preservation of the semantics can be decomposed into the preservation of clock semantics at the *clock calculation* phase and that of data dependencies at the *static scheduling* phase, and, finally, value-equivalence of variables at the *code generation* phase.

Translation Validation for Clock Transformations in a Synchronous Compiler. In this work, the clock semantics of the source and transformed programs are formally represented as *clock models*. A clock model is a first-order logic formula that characterizes the presence/absence status of all signals in a Signal program at a given instant. Given two clock models, a *clock refinement* between them is defined which expresses the semantic preservation of clock semantics. A method to check the existence of clock refinement is defined as a satisfiability problem which can be automatically and efficiently proved by a SMT solver.

Let Cp^{sig} and Val_{clk} be the functions which define the Signal compiler and a validator, respectively. The following function defines a formally verified compiler for the *clock calculation and Boolean abstraction* phase. We write $C \sqsubseteq_{clk} A$ to denote that there exists a refinement between A and C.

$$Cp^{sig}_{Val_{clk}}(A) = \begin{cases} C & \text{if } Cp^{sig}(A) = C \text{ and } Val_{clk}(A, C) = true \\ Error & \text{if } Cp^{sig}(A) = C \text{ and } Val_{clk}(A, C) = false \\ Error & \text{if } Cp^{sig}(A) = Error \end{cases}$$

where $Val_{clk}(A, C) =$ true if and only if $C \sqsubseteq_{clk} A$.

Precise Deadlock Detection for Polychronous Data-flow Specifications. Dependency graphs are a commonly used data structure to encode the streams of values in data-flow programs and play a central role in scheduling instructions during auto-mated code generation from such specifications. In this work [17], we propose a precise and effective method that combines a structure of dependency graph and first order logic formulas to check whether multi-clocked data-flow specifications are deadlock free before generating code from them. We represent the flow of values in the source programs by means of a dependency graph and attach first-order logic formulas to condition these dependencies. We use an SMT solver [88] to effectively reason about the implied formulas and check deadlock freedom.

Evaluating SDVG translation validation: from Signal to C. This work focuses on proving that every output signal in the source program and the corresponding variable in the compiled program, the generated C program, have the same values. The computations of all signals and their compiled counterparts are represented by a shared value-graph, called *Synchronous Data-flow Value-Graph* (SDVG).

Given a SDVG, assume that we want to show that two variables have the same value. We simply need to check that they are represented by the same sub-graph, meaning that they point to the same graph node. If all output signals in the source program A and the corresponding variables in the generated C program have the same value, then we say that C refines A, denoted by $C \sqsubseteq_{val} A$.

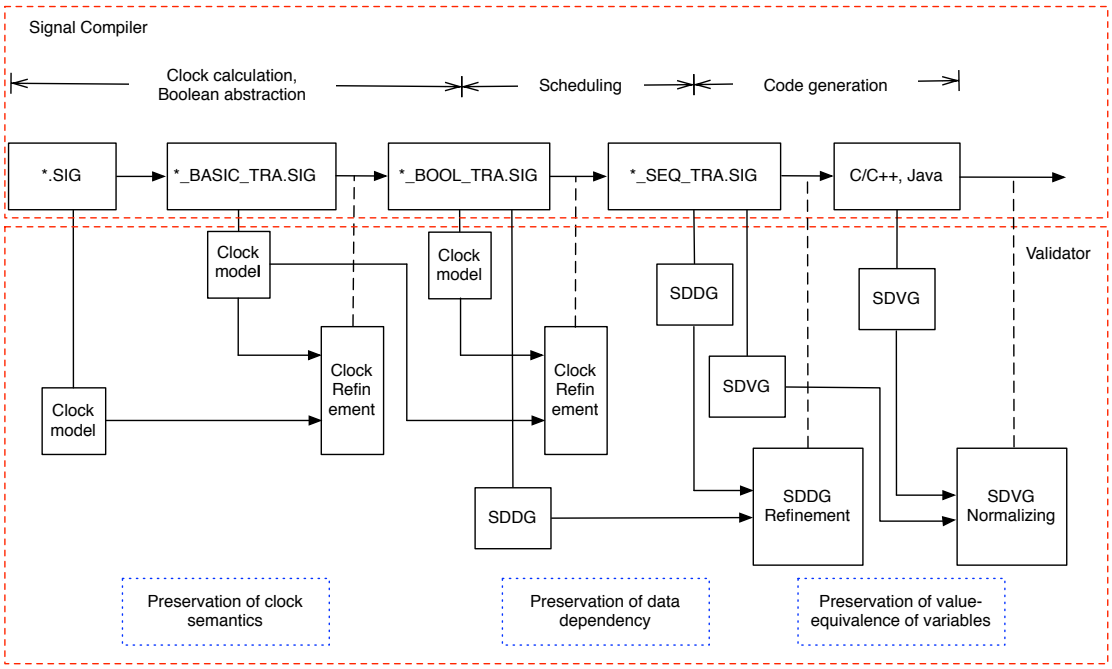

Figure 3. Our Integration within Polychrony Toolset

Implementation and Experiments. At a high level, our tool *SigCert* (https://scm.gforge.inria.fr/svn/sigcert) developed in OCaml checks the correctness of the compilation of Signal compiler w.r.t clock semantics, data dependence, and value-equivalence as given in Figure 3.

6.4. Ongoing integration of Polychrony with the P toolset

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

Current state of P. The FUI project P has been extended until September 2015. Partners in the project now focus on code generation aspects, leaving software architecture aspects aside. The qualifiable model-based code generator, previously known as P toolset, is now named QGen (QGen is developped mostly in Ada 2012 and Python).

Model transformation (P2S). We developped a transformation tool hereafter named `P2S` for expressing P system models as Signal processes. Our work is based on EMF (Eclipse Modelling Framework), taking advantage of the existing Ecore metamodels available for both P and SSME.

The `P2S` tool is written in Clojure, which is a dialect of Lisp running on the Java Virtual Machine. This approach allows to benefits from a terse and expressive language while remaining fully interoperable with existing Java libraries (including Eclipse plugins and especially Polychrony ones).

SSME abstraction layer.`P2S` uses an abstraction layer to simplify the creation of SSME elements, while taking into account EMF idioms. For example, the following expression creates a `ProcessModel` instance using the currently registered EMF factory:

```
(process "TestProcess"
         :in '[boolean h integer x]
         :out '[integer y]
         :body (sigdef
                (id 'y)
                (when* (id 'x) (id 'h))))
```

The newly created object can be saved as an XMI file using EMF utilities (the XMI file is 40 lines long and not shown here). This object and its children represent the following Signal process expression [89]:

```
process TestProcess =
  (? boolean h; integer x;
   ! integer y; )
  (| y := (x when h) |);
```

Transformation to P. Conversion from P to Signal relies on Clojure's multimethods. We defined a `convert` multimethod which dispatches on the type of its argument and possibly on additional modifiers. This mechanism allows to convert expressions differently depending on whether we want to produce a Signal declaration or an expression. For example, the following method specializer converts a P port as a signal declaration:

```
(defmethod convert [Port :declaration] [port & _]
  (ssme/signal-declarations
   (convert (.getDataType port))
     (ssme/with-comment
       [(readable-name port :declaration) :post]
       (ssme/id (p-name port)))))
```

[86] *Translation validation.* Pnueli A., Siegel M., and Singerman E. In Proceedings of TACAS'98, 1998.

[87] *Translation validation: From signal to c.* M. Siegel A. Pnueli and E. Singeman. In Correct Sytem Design Recent Insights and Advances, 2000.

[88] *Satisfiability modulo theories: An appetizer.* L. de Moura and N. Bjorner. In Brazilian Symposium on Formal Methods, 2009.

[89] Even using the dedicated `signalTreeAPI` utility class, the same example would require many more lines of Java code.

Since the specializer contains the `:declaration` keyword, the previous conversion is applied only when called with that keyword given as an extra argument, as follows:

```
(convert some-port :declaration)
```

The more general specializer, which is defined below, is meant to be used inside Signal expressions and, as such, only returns a Signal identifier:

```
(defmethod convert Port [port]
   (ssme/id (p-name port)))
```

Note also that thanks to class inheritance, the above methods are sufficient to convert all kind of P ports (input/output, data/control).

The naming scheme for the resulting SSME elements is handled by the `p-name` multi-method and relies on XMI identifiers of the original P elements: XMI identifiers generated by QGen are string representations of positive integers. Moreover, those identifiers are guaranteed to be unique in a model. These two properties allows to generate valid Signal identifiers while ensuring traceability (e.g. signal `P101` links to the unique port of the original model having 101 as a unique identifier).

Datatypes are currently converted as Signal predefineds types, which do not always match exactly the original types. Another partially implemented option consists in translating them as `external` types in Signal. Some types, like arrays, are converted the same way with both approaches:

```
(defmethod convert TArray [a]
  (reduce (fn [base dim]
            (ssme/array-type base (convert dim :signal)))
          (convert (.getBaseType a))
          (.getDimensions a)))
```

Conversion of arithmetic operations may also lead to predefined Signal operators (by default) or externally defined functions (incomplete). The current approach has been tested on QGen's test models and successfully translates 208 of the 227 models.

Partial block sequencing. The conversion from P models to Signal takes into account block dependencies as computed by QGen. Unfortunately, QGen's block sequencer produces a total order between blocks, with leads to over-constrained Signal models. We contributed to the model compiler by writing an alternative (Ada) package which provides: (i) a way to parameterize block sequencing, and (ii) partial ordering options.

Our implementation is not part of the qualified compiler, but available as a standalone (non-qualifiable) executable. However, during the development of this block sequencer, we were able to find and correct existing bugs in QGen's sequencer.

Perspectives. From a software development point of view, our current work needs to be packaged and better integrated with the build system of Polychrony. By the way, that existing build process itself could be slightly improved by using Maven configuration files instead of Eclipse manual plug-in management.

The use of a functional language on top of the Java Virtual Machine is an interesting aspect of our work. By allowing the abstraction layer, which currently works at the SSME level, to also access the existing Signal library, we could provide an API for writing and compiling Signal code using a domain-specific language expressed in Clojure (there already exist JNI bindings with the native library). This feature could help developpers hook into, or interact with, the existing Signal compiler in order to customize parts of the code generation strategies.

Regarding the P project, we still need to test code distribution strategies on industrial use-cases and determine how it can be exploited at the system-model level.

6.5. A synchronous annex for the AADL

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

The SAE committee on the AADL adopted our recommendations to implement a timed and synchronous behavioural annex for the standard [20]. The specification and reference implementation of this revised behavioral annex will be the focus of most our attention next year.

We propose a synchronous timing annex for the SAE standard AADL. Our approach consists of building a synchronous model of computation and communication that best fits the semantics and expressive capability of the AADL and its behavioral annex and yet requires little to know (syntactic) extension to it, i.e. to identify a synchronous core of the AADL (which prerequisites a formal definition of synchrony at hand) and define a formal design methodology to use the AADL in a way that supports formal analysis, verification and synthesis.

Our approach first identifies the core AADL concepts from which time events can be described. Then, is considers the behavior annex (BA) as the mean to model synchronous signals and traces through automata. Finally, we consider elements of the constraint annex to reason about abstractions of these signals and traces by clocks and relations among them. To support the formal presentation of these elements, we define a model of automata that comprises a transition system to express explicit transitions and constraints, in the form of a boolean formula on time, to implicitly constraint its behavior. The implementation of such an automaton amounts to composing its explicit transition system with that of the controller synthesised from its specified constraints.

6.6. New features of Polychrony

Participants: Loïc Besnard, Thierry Gautier, Paul Le Guernic.

Reduction of communications. We have developed, as a general functionality of the Signal toolbox, a means to reduce communications between two graphs, using assignment clocks and utility clocks.

For a given signal x, its assignment clock represents the instants at which it may be modified (otherwise than keeping its previous value $x\$$) while its utility clock in a given graph represents the instants at which it is effectively used in this graph.

Considering two graphs G_i and G_j with a signal x sent from G_i to G_j, containers are built above G_i and G_j in order to minimize the clock at which x must be communicated. On the sender side, the signal which has to be sent can be reduced to x_j with $x_j := x \texttt{ when } h$, where h is the lower bound of the assignment clock of x and the utility clock of x in G_j. On the receiver side, x is replaced in G_j by x_r with $x\_r := x_j \texttt{ default } x\_r\$$.

Note that this reduction is not always possible because it may introduce cycles between signals and clocks.

Experiments have been made on programs intended to the distribution of Quartz applications, with a gain of up to 40on some of them [18].

Polychronous automata. We have defined a new model of polychronous constrained automata that has been provided as semantic model for our proposal of an extension of the AADL behavioural annex [20]. An algebra of regular expressions is also defined to represent abstractions of constrained automata or, more specifically, their time constraints.

An experimental implementation of the semantic features of this “timing annex” will be provided through the Polychrony framework. For that purpose, representations of automata are introduced in the Signal toolbox of Polychrony. In a first step, we have decided to provide only a minimal extension of the Signal language itself. A new syntactic category of process model, which is an automaton model, has been introduced. States are described by the association of labels with subprocesses, as it is available in Signal, and transitions between states, at a given clock, are written as calls to *intrinsic* (predefined) processes. Constraints described as regular expressions on events should also be introduced using intrinsic processes.

Automata will be used in different ways related to stategies of compilation. In particular, they will serve as an alternative model for the code generation. For that purpose, polychronous programs are rewritten thanks to valuations of memorized boolean signals. The resulting partially valuated programs are the states of a control automaton.

Such techniques can be applied to implement endo-isochronous programs. Currently, code may be generated only for endochronous programs, for which clock hierarchy is a tree. Endo-isochronous programs are compositions of endochronous programs the "intersection" of which is also endochronous. For example, an automaton can be built to generate code when two signals are known to alternate.

6.7. Optimized Distribution of Synchronous Programs via a Polychronous Model

Participants: Ke Sun, Jean-Pierre Talpin, Thierry Gautier, Loïc Besnard.

We propose a distribution methodology for synchronous programs [18], applied in particular on programs written in the Quartz language [90]. The given program is first transformed into an intermediate model of guarded actions. After user-specified partitioning, the generated sub-models are transformed into equivalent Signal processes [7]. Then, the unnecessary constraints are eliminated from the processes to avoid unnecessary synchronization. Finally, within the Signal framework, the minimal frequencies of communication and computation are computed via multi-clock calculation. This operation can efficiently reduce the communication quantity and the computation load, with no change to the interface behaviors. Along this way, an optimized data-flow network over desynchronized processing locations can be constructed.

The presented methodology has been implemented within the integrated framework Quartz/Averest + Signal/Polychrony. To illustrate and validate this methodology, a series of examples served as case studies. Each of them has been written in the Quartz language and distributed over different processing locations using the presented optimization methodology. These case studies confirm that the optimization can bring in significant communication reduction. In the sequel, the efficient utilization of distributed systems is substantially updated.

6.8. Component-based Design of Multi-rate Systems

Participants: Ke Sun, Jean-Pierre Talpin, Thierry Gautier, Loïc Besnard.

The Synchronous language Quartz is well suited for modeling mono-clocked systems. However, as based on the model of computation (MoC) synchrony, its parallelism feature excessively strengthens the synchronization. Such synchronous parallelism in particular restricts independent component design. That is, the modeling of connected components should constantly refer to each other to guarantee the achievement of desired system behavior. Hence, Quartz cannot support well the component-based system design, in particular for the distributed systems that are generally deployed over desynchronized processing locations with multi-rate clocks.

In contrast to Quartz, the polychronous language Signal is based on the MoC polychrony. As its name suggests, a polychronous program makes use of multi-rate clocks to drive its execution. One can consider that each component in the program holds its own master clock, and there is no longer a master clock for the whole program. The resulted architecture is named globally asynchronous locally synchronous (GALS) architecture.

Through integrating Quartz with Signal, a component-based methodology is proposed for designing multi-rate systems: at first, components are modeled independently to achieve local behaviors; secondly, inter-component communications are adjusted using Signal to realize intermittent synchronization. In this way, the modeling approach for mono-clocked systems evolves into a component-based modeling methodology. Such significant progress not only facilitates the component coordination, but also enhances the component reusability, in particular for modeling large scale systems.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

7.1.1. Toyota Info-Technology Centre (2014-2016)

[90] *The Synchronous Programming Language Quartz*. K. Schneider, Technical Report n. 375. University of Kaiserslautern, 2009

Title: Co-Modeling of Safety-Critical Multi-threaded Embedded Software for Multi-Core Embedded Platforms

Inria principal investigator: Jean-Pierre Talpin

International Partner (Institution - Laboratory - Researcher):

Virginia Tech Research Laboratories, Arlington (United States)

Embedded Systems Group, Teschnische Universität Kaiserslautern (Germany)

Duration: 2014 - 2016

Abstract: We started a new project in April 2014 funded by Toyota ITC, California, to work with Huafeng Yu (a former post-doctorate of team ESPRESSO) and with VTRL as US partner. The main topic of our project is the semantic-based model integration of automotive architectures, virtual integration, toward formal verification and automated code synthesis. This year, Toyota ITC is sponsoring our submission for the standardisation of a time annex in the SAE standard AADL.

In a second work-package, we aim at elaborating a standardised solution to virtually integrate and simulate a car based on heterogeneous models of its components. This year, it will be exemplified by the elaboration of a case study in collaboration with Virginia Tech. The second phase of the project will consist of delivering an open-source, reference implementation, of the proposed AADL standard and validate it with a real-scale model of the initial case-study.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR

Program: ANR

Project acronym: **VeriSync**

Project title: Vérification formelle d'un générateur de code pour un langage synchrone

Duration: Nov. 2010 - Oct. 2013

Coordinator: IRIT

Other partners: IRIT

URL: http://www.irit.fr/Verisync/

Abstract:

The VeriSync project aims at improving the safety and reliability assessment of code produced for embedded software using synchronous programming environments developed under the paradigm of Model Driven Engineering. This is achieved by formally proving the correctness of essential transformations that a source model undergoes during its compilation into executable code.

Our contribution to VeriSync consists of revisiting the seminal work of Pnueli et al. on translation validation and equip the Polychrony environment with updated verification techniques to scale it to possibly large, sequential or distributed, C programs generated from the Signal compiler. Our study covers the definition of simulation and bisimulation equivalence relations capable of assessing the correspondence between a source Signal specification and the sequential or concurrent code generated from it, as well as both specific abstract model-checking techniques allowing to accelerate verification and counter-example search techniques, to filter spurious verification failures obtained from excessive abstracted exploration.

——

Program: ANR

Project acronym: **Feever**

Project title: Faust Environment Everyware

Duration: 2014-2016

Coordinator:

Other partners:

URL: http://www.feever.fr

Abstract:

The aim of project FEEVER is to ready the Faust music synthesis language for the Web. In this context, we collaborate with Mines ParisTech to define a type system suitable to model music signals timed at multiple rates and to formally support playing music synthesised from different physical locations.

8.1.2. Competitivity Clusters

Program: FUI

Project acronym: P

Project title: Project P

Duration: March 2011 - Sept. 2015

Coordinator: Continental Automotive France

Other partners: 19 partners (Airbus, Astrium, Rockwell Collins, Safran, Thales Alenia Space, Thales Avionics...)

URL: http://www.open-do.org/projects/p/

Abstract:

The aim of project P is 1/ to aid industrials to deploy model-driven engineering technology for the development of safety-critical embedded applications, 2/ to contribute on initiatives such as ITEA2 OPEES and Artemisia CESAR to develop support for tools inter-operability, and 3/ to provide state-of-the-art automated code generation techniques from multiple, heterogeneous, system-levels models. The focus of project P is the development of a code generation toolchain starting from domain-specific modeling languages for embedded software design and to deliver the outcome of this development as an open-source distribution, in the aim of gaining an impact similar to GCC for general-purpose programming, as well as a kit to aid with the qualification of that code generation toolchain.

The contribution of project-team TEA in project P is to bring the necessary open-source technology of the Polychrony environment to allow for the synthesis of symbolic schedulers for software architectures modeled with P in a manner ensuring global asynchronous deterministic execution..

8.1.3. PAI CORAC

Program: CORAC

Project acronym: CORAIL

Project title: Composants pour l'Avionique Modulaire Étendue

Duration: July 2013 - May 2017

Coordinator: Thales Avionics

Other partners: Airbus, Dassault Aviation, Eurocopter, Sagem...

URL: http://www.corac-ame.com/

Abstract:

The CORAIL project aims at defining components for Extended Modular Avionics. The contribution of project-team TEA is to define a specification method and to provide a generator of multi-task applications.

8.2. International Initiatives

8.2.1. International Project Grants

8.2.1.1. USAF Office for Scientific Research – Grant FA8655-13-1-3049

Title: Co-Modeling of Safety-Critical Multi-threaded Embedded Software for Multi-Core Embedded Platforms

Inria principal investigator: Jean-Pierre Talpin

International Partner (Institution - Laboratory - Researcher):

Virginia Tech Research Laboratories, Arlington (United States)

Embedded Systems Group, Teschnische Universität Kaiserslautern (Germany)

Duration: 2013 - 2016

See also: http://www.irisa.fr/espresso/Polycore

Abstract: The aim of the USAF OSR Grant FA8655-13-1-3049 is to support collaborative research entitled "Co-Modeling of safety-critical multi-threaded embedded software for multi-core embedded platforms" between Inria project-team ESPRESSO, the VTRL Fermat Laboratory and the TUKL embedded system research group, under the program of the Polycore associate-project.

8.2.2. Inria International Partners

8.2.2.1. Declared Inria International Partners

8.2.2.1.1. The University of Hong Kong

Title: Virtual Prototyping of embedded software architectures

International Partner (Institution - Laboratory - Researcher):

The University of Hong Kong (Hong ,Kong)

Duration: 2012 - now

We collaborate with John Koo at the University of Hong Kong (HKU) and the LIAMA since two years through visiting grants of the Chinese Academy of Science and of the University of Rennes on the topics of heterogeneous time modelling and virtual prototyping. We submitted an ANR project proposal on this topic.

An engineer of SIAT, Riu Li, has developed a pilot project to evaluate Polychrony in the context of virtual prototyping and real-time simulation of automotive systems (the controller of a V6 turbo-charged engine model in LMS [91]). Our collaboration started in 2011 at the occasion of a joint Summer School on Embedded Systems organised by SIAT and LIAMA at SIAT. John Koo was invited scientist at Inria-Rennes in Summer 2012 and Jean-Pierre Talpin invited at SIAT by the Chinese Academy of Science from December 2012 to August 2013.

The partners submitted a PHC proposal and intend to resubmit a joint project proposal for the ANR-HK international program. A longer term goal of our collaboration is to setup, within the IET, a joint laboratory with Inria, in order to both disseminate formal methods for embedded system design on a specific Master program, and jointly contribute to an open-source system design platform with European and Asian industrial partners which are sponsoring the IET.

[91] LMS by Siemens http://www.plm.automation.siemens.com/en_us/products/lms

8.2.2.1.2. Virginia Tech Research Laboratories

Title: Models of computation for embedded software design

International Partner (Institution - Laboratory - Researcher):

Virginia Tech Research Laboratories (USA)

Duration: 2003 - now

Team TEA collaborates with Sandeep Shukla, Virginia Tech, since 2002. First, in the frame of the NSF-Inria program with Rajesh Gupta, UCSD, until 2004; Inria's associated project BALBOA, until 2007; with the sabbatical of Sandeep Shukla at IRISA in 2008-2009 (funded by Inria-Rennes, the University of Rennes 1, Inria's scientific board); and, from 2011 to 2013, in the context of the associate-project POLYCORE, together with the ESG group at TU Kaiserslautern.

Following up Sandeep's sabbatical, the Fermat Laboratory was awarded a series of research grant by the US Air Force Research Laboratory (AFRL) to develop a modelling environment based on Polychrony. In this context, Virginia Tech hired a former PhD. of team ESPRESSO, Julien Ouy, to contribute and coordinate this project's work. Since 2013, the scope of our collaboration has extended with the three years grant awarded to team TEA by the USAF Office for Scientific Research (AFOSR).

To date, our fruitful and sustained collaboration has yield the creation of the ACM-IEEE MEMOCODE conference series [92] in 2003, of the ACM-SIGDA FMGALS workshop series, and of a full-day tutorial at ACM-IEEE DATE'09 on formal methods in system design. We have jointly edited two books with Springer [93] [94], two special issues of the IEEE Transactions on Computers and one of the IEEE Transactions on Industrial Informatics, and published more than 30 joint papers in international scientific journals and conferences.

8.2.2.2. Informal International Partners

8.2.2.2.1. Technische Universitaet Kaiserslautern (DE)

We collaborate with Klaus Schneider, leader of the ESG group at Uni. Kaiserslautern, since 2011 in the frame of the POLYCORE associate project. Our aim is to develop a joint, open-source, toolchain based on the Averest (ESG) and POP (TEA) environments. Our collaboration has been quite fruitful with several recent journal publications [95] [96]. Numerous visits and exchanges of personnel between team TEA and the ESG have allowed us to develop ONYX, a cross-compiler between the Averest and POP environments.

Onyx mixes imperative Quartz modules and declarative Signal networks to specify multi-clocked systems. We intend to further its development by the submission of a joint ANR or European project. Our objective is to develop an environment capable of synthesising distributed, loosely synchronised executives from imperative Quartz modules whose schedules are specified by multi-clocked data-flow specifications. A new version of this front-end, developed by Sun Ke, will be integrated in the POP environment.

8.3. International Research Visitors

8.3.1. Visits to International Teams

8.3.1.1. Research stays abroad

Jean-Pierre Talpin was awarded a visiting researcher grant by the US Air Force Research Laboratories for collaborative research with the Virginia Tech Research Laboratories. In this context, he visited the Arlington and Falls Church VT campuses in Spring, Summer and Fall 2014 for a duration of two and a half months.

[92] ACM-IEEE MEMOCODE conference series. http://memocode-conference.com

[93] *Formal methods and models for system design*, R. Gupta, S. Shukla, J.-P. Talpin, Eds. ISBN 1-4020-8051-4. Springer, 2004.

[94] *Synthesis of embedded systems*. S. Shukla, J.-P. Talpin, Eds. ISBN 978-1-4419-6399-4. Springer, 2010

[95] *Embedding polychrony into synchrony*. J. Brandt, M. Gemünde, K. Schneider, S. Shukla, and J.-P. Talpin. In Transactions on Software Engineering (TSE). IEEE, 2012.

[96] *Representation of synchronous, asynchronous, and polychronous components by clocked guarded Actions*. J. Brandt, M. Gemünde, K. Schneider, S. Shukla, and J.-P. Talpin. In Design Automation for Embedded Systems (DAES), Special Issue on Languages, Models and Model Based Design for Embedded Systems. Springer, 2012.

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organisation

9.1.1.1. member of the organizing committee

Jean-Pierre Talpin is a member of the steering committee of the ACM-IEEE Conference on Methods and Models for System Design (MEMOCODE).

9.1.1.2. responsable of the conference program committee

Jean-Pierre Talpin co-chaired the program committee of the 12th. ACM-IEEE MEMOCODE conference and the 5th. IEEE RTSS workshop AVICPS.

9.1.1.3. member of the conference program committee

Jean-Pierre Talpin served the program committee of the 12th. ACM EMSOFT'14, ACM LCTES'14, and SCOPES'14 conferences, as well as the MODELS workshop ACVI'14.

Thierry Gautier served as program commitee member for the 2014 Electronic System Level Synthesis Conference, ESLsyn 2014 (http://www.ecsi.org/eslsyn) and for the 12th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE'14 (http://memocode.irisa.fr).

9.1.2. Journal

9.1.2.1. member of the editorial board

Jean-Pierre Talpin is Associate Editor with the ACM Transactions for Embedded Computing Systems (TECS); he is also Associate Editor with the Springer journal on Frontiers of Computer Science (FCS) and Associate Editor with EURASIP journal of embedded systems.

9.1.2.2. reviewer

Jean-Pierre Talpin reviewed for Acta Informatica and Science of Computer Programming.

9.2. Teaching - Supervision - Juries

9.2.1. Supervision

- *PhD advisory of Chan Ngo* by Jean-Pierre Talpin, Paul Le Guernic and Thierry Gautier, entitled "*Vérification formelle d'un générateur de code pour les spécifications polychrones*", defended in December 2014.
- *PhD advisory of Sun Ke* by Jean-Pierre Talpin, Paul Le Guernic and Thierry Gautier, entitled "*Conception conjointe de modules synchrones et de réseaux flot-de-données polychrones*", since 2011.
- *PhD co-advisory of Imré Frotier de la Messelière* by Jean-Pierre Talpin and Pierre Jouvelot (Mines ParisTech), entitled "*Conception et implémentation d'applications musicales innovantes pour les nouvelles technologies multimédia*", since October 2013.

9.2.2. Juries

Jean-Pierre Talpin served as Referee at the Thesis Defence of Vincent Gaudel (UBO), entitled "*Des patrons de conception pour assurer l'analyse d'architectures : un exemple avec l'analyse d'ordonnancement*", December 2014.

10. Bibliography

Major publications by the team in recent years

[1] L. BESNARD, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Compilation of Polychronous Data Flow Equations*, in "Synthesis of Embedded Software", S. K. SHUKLA, J.-P. TALPIN (editors), Springer, 2010, pp. 1-40 [*DOI :* 10.1007/978-1-4419-6400-7_1], http://hal.inria.fr/inria-00540493

[2] A. BOUAKAZ, 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], https://hal.inria.fr/hal-00916487

[3] C. BRUNETTE, J.-P. TALPIN, A. GAMATIÉ, T. GAUTIER. *A metamodel for the design of polychronous systems*, in "The Journal of Logic and Algebraic Programming", 2009, vol. 78, n^{o} 4, pp. 233 - 259, IFIP WG1.8 Workshop on Applying Concurrency Research in Industry [*DOI :* 10.1016/J.JLAP.2008.11.005], http://www.sciencedirect.com/science/article/pii/S1567832608000957

[4] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Polychronous Design of Embedded Real-Time Applications*, in "ACM Transactions on Software Engineering and Methodology (TOSEM)", April 2007, vol. 16, n^{o} 2, http://doi.acm.org/10.1145/1217295.1217298

[5] A. GAMATIÉ, T. GAUTIER. *The Signal Synchronous Multiclock Approach to the Design of Distributed Embedded Systems*, in "IEEE Transactions on Parallel and Distributed Systems", 2010, vol. 21, n^{o} 5, pp. 641-657 [*DOI :* 10.1109/TPDS.2009.125], http://hal.inria.fr/inria-00522794

[6] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Synchronous design of avionic applications based on model refinements*, in "Journal of Embedded Computing (IOS Press)", 2006, vol. 2, n^{o} 3-4, pp. 273-289, http://hal.archives-ouvertes.fr/hal-00541523

[7] P. LE GUERNIC, J.-P. TALPIN, J.-C. LE LANN. *Polychrony for system design*, in "Journal of Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design", June 2003, vol. 12, n^{o} 03, http://hal.inria.fr/docs/00/07/18/71/PDF/RR-4715.pdf

[8] D. POTOP-BUTUCARU, Y. SOREL, R. DE SIMONE, J.-P. TALPIN. *From Concurrent Multi-clock Programs to Deterministic Asynchronous Implementations*, in "Fundamenta Informaticae", January 2011, vol. 108, n^{o} 1-2, pp. 91–118, http://dl.acm.org/citation.cfm?id=2362088.2362094

[9] J.-P. TALPIN, J. OUY, T. GAUTIER, L. BESNARD, P. LE GUERNIC. *Compositional design of isochronous systems*, in "Science of Computer Programming", February 2012, vol. 77, n^{o} 2, pp. 113-128 [*DOI :* 10.1016/J.SCICO.2010.06.006], http://hal.archives-ouvertes.fr/hal-00768341

Publications of the year

Doctoral Dissertations and Habilitation Theses

[10] V. C. NGÔ. *Formal verification of a synchronous data-flow compiler : from Signal to C*, Université Rennes 1, July 2014, https://tel.archives-ouvertes.fr/tel-01067477

Articles in International Peer-Reviewed Journals

[11] L. BESNARD, A. BOUAKAZ, T. GAUTIER, P. L. GUERNIC, Y. MA, J.-P. TALPIN, H. YU. *Timed behavioural modelling and affine scheduling of embedded software architectures in the AADL using Polychrony*, in "Science of Computer Programming", July 2014, 20 p. , https://hal.inria.fr/hal-01095010

[12] J.-P. TALPIN, J. BRANDT, M. GEMÜNDE, K. SCHNEIDER, S. SHUKLA. *Constructive Polychronous Systems*, in "Science of Computer Programming", September 2014, 20 p. , https://hal.inria.fr/hal-01095004

[13] Z. YANG, K. HU, D. MA, J.-P. BODEVEIX, L. PI, J.-P. TALPIN. *From AADL to timed abstract state machine: a certified model transformation*, in "Journal of Systems and Software", July 2014, 20 p. , https://hal.inria.fr/hal-01095002

International Conferences with Proceedings

[14] A. BOUAKAZ, T. GAUTIER. *An abstraction-refinement framework for priority-driven scheduling of static dataflow graphs*, in "International Conference on Formal Methods and Models for System Design", Lausanne, Switzerland, 2014 Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), IEEE Press, October 2014 [*DOI :* 10.1109/MEMCOD.2014.6961838], https://hal.inria.fr/hal-01088205

[15] A. BOUAKAZ, T. GAUTIER, J.-P. TALPIN. *Earliest-Deadline First Scheduling of Multiple Independent Dataflow Graphs*, in "2014 IEEE Workshop on Signal Processing Systems (SiPS)", Belfast, United Kingdom, October 2014, https://hal.inria.fr/hal-01092606

[16] I. F. D. L. MESSELIÈRE, P. JOUVELOT, J.-P. TALPIN. *A constraint-solving approach to Faust program type checking*, in "Constraint Programming meets Verification", Lyon, France, Constraint Programming meets Verification, The 20th International Conference on Principles and Practice of Constraint Programming, September 2014, https://hal.inria.fr/hal-01094998

[17] V. C. NGO, J.-P. TALPIN, T. GAUTIER. *Precise deadlock detection for polychronous data-flow specifications*, in "ESLsyn - DAC 2014", San Francisco, United States, May 2014 [*DOI :* 10.1109/ESLSYN.2014.6850379], https://hal.inria.fr/hal-01086843

[18] K. SUN, L. BESNARD, T. GAUTIER. *Optimized Distribution of Synchronous Programs via a Polychronous Model*, in "Formal Methods and Models for System Design (MEMOCODE'14)", Lausanne, Switzerland, October 2014, pp. 42 - 51 [*DOI :* 10.1109/MEMCOD.2014.6961842], https://hal.inria.fr/hal-01088953

[19] H. YU, J.-P. TALPIN, S. SHUKLA, P. JOSHI, S. SHIRAISHI. *Towards an architecture-centric approach dedicated to model-based virtual integration for embedded software systems*, in "Workshop on Architecture Centric Virtual Integration", Valencia, Spain, Workshop on Architecture Centric Virtual Integration, CEUR Workshop Proceedings, September 2014, https://hal.inria.fr/hal-01094995

Research Reports

[20] L. BESNARD, E. BORDE, P. DISSAUX, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Logically timed specifications in the AADL : a synchronous model of computation and communication (recommendations to the SAE committee on AADL)*, April 2014, n° RT-0446, 27 p. , https://hal.inria.fr/hal-00970244

Other Publications

[21] V. C. NGO, J.-P. TALPIN, T. GAUTIER. *Translation Validation for Clock Transformations in a Synchronous Compiler*, December 2014, https://hal.inria.fr/hal-01087795

[22] V. C. NGO, J.-P. TALPIN, T. GAUTIER. *Translation Validation for Synchronous Data-flow Specification in the SIGNAL Compiler*, November 2014, https://hal.inria.fr/hal-01087801