INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team aoste

Models and Methods for the Analysis and Optimization of Systems with Real-time and Embedded Constraints

Sophia Antipolis - Rocquencourt

THEME COM

Activity Report

2004

Table of contents

1. Team

Aoste is a joint project with the university of Nice (UNSA) and the CNRS, through the UMR I3S. It is also spread between Sophia-Antipois and Rocquencourt, as a follow-up of the former INRIA Tick and Ostre teams, and the I3S Sports team.

Head of project-team
Robert de Simone [Senior Researcher, Inria]

Vice-head of project team
Yves Sorel [Senior Researcher, Inria]

Administrative assistant
Sophie Honnorat [Sophia-Antipolis, part-time]
Nelly Maloisel [Rocquencourt, part-time]

Staff member UNSA
Charles André [professor]
Frédéric Mallet [associate professor]
Marie-Agnès Peraldi-Frati [associate professor]

Ph. D. student
Julien Boucaron [ST contract funding, since october 2004]
Liliana Cucu [INRIA scolarship, until may 2004]
Hamoudi Kalla [joint INRIA scolarship with BIP team, until december 2004]
Linda Kaouane [joint scolarship with ESIEE, until december 2004]
Fabrice Peix [MENESR scholarship, until september 2004]
Nicolas Pernet [INRIA scolarship]
Mickaël Raulet [INSA/MITSUBISHI ELECTRIC scholarship]
Olivier Tardieu [Corps des Mines, until november 2004]
Eric Vecchié [PACA regional scholarship, until october 2004, then ATER]

Technical Staff
Cyril Faure [Project technical staff, Rocquencourt]
Julien Forget [Project technical staff, Rocquencourt, until november 2004]
Arnaud Rouanet [Software development staff, Rocquencourt]

Student interns
Belouaer Hadj [University Nice/Sophia-Antipolis]
Christophe Bourcier [University Versailles Saint-Quentin-en-Yvelines]
Xavier Cousin [ESIEE Noisy-Le-Grand]
Omar El Ganaoui [INPG Grenoble]
Alain Mvié [University Littoral Côte d'Opale]

External collaborators
Thierry Grandpierre [Assistant Professor ESIEE Noisy-Le-Grand]
Christophe Lavarenne [Engineer UBIC]

2. Overall Objectives

Keywords: *UML, adequation, architectural models, automatic verification, code distribution, code generation, codesign, compilation techniques, formal semantics, hardware synthesis, high-level modeling, mapping, multiprocessors, optimization, program analysis, real-time embedded systems, scheduling, synchronous hypothesis, synchronous reactive formalisms, systems-on-Chip.*

The main goal of Aoste is to provide innovative approaches for the design of real-time and embedded systems, based on powerful algorithmic methods applied to well-defined models with sound mathematical semantics (in short: formal model-based design). Here “design” means altogether:

High-Level Modeling
Model transformation and analysis
Implementation onto Embedded platforms

We shall try to promote our semantically sound, model-based approach at each of these three levels. In the case of *high-level modeling*, this translates into the need for offering appealing and familiar syntactic constructs to help shape up modular semantic model construction; for the *transformation and analysis part*, the goal is to provide "correct-by-construction" synthesis techniques to transform models progressively from higher-level to lower-level, and check otherwise their match against correctness properties; in the *implementation onto embedded platforms* part, we introduce an explicit architectural platform, and consider the optimized mapping in time and space of models-as-programs onto this platform.

To cover this vast spectrum of topics we need to specialize the type of formalisms we shall consider. We focus on *synchronous reactive languages*, such as Esterel/SyncCharts, and on the *Application/Architecture/Adequation* (AAA) methodology, as implemented in the SynDEx design environment. Typical application domains for our techniques are found in nowadays major embedded electronic fields, such as mobile telephones and similar appliances branded as "secure communicating objects", in mobile robotics, in automotive and aircraft transportation, and in digital system-on-chip (SoC) design.

3. Scientific Foundations

3.1. High-Level Modeling

Keywords: *Esterel*, *SyncCharts*, *UML*, *synchronous formalisms*.

Participants: Charles André, Julien Boucaron, Robert de Simone, Frédéric Mallet, Marie-Agnès Péraldi.

3.1.1. Synchronous formalisms

Historically the so-called *synchronous reactive formalisms* [27][10] were developed, mostly inside French research groups in the 1980's (Esterel, Lustre, Signal was the trilogy), as foundational study of semantically well-founded description languages for real-time embedded software systems. Meanwhile a number of modeling languages were also introduced in this field, aiming more specifically at system simulation with discrete time steps: HDLs for hardware circuits, Statecharts for embedded software modeling, Simulink for signal and image processing and control theory. It should be recognized in our view that synchronous languages brought exactly what these simulation formalisms lack: a clear sense of correct construction properties, under which the instantaneous behavior (the reaction) can always be provably scheduled safely in an intelligible way, and which makes executions deterministic and complete (because all valid scheduling are essentially causally equivalent). With such an assumption there is a guaranteed match between the simulation model and the executable code obtained through the implementation ("*what you simulate is what you execute*"), and this opens the way to many important design activities, such as synthesis, verification, and test generation activities, which are largely banned out of a setting where the simulation model may differ from the actual implementation. Also, the precise scheduling in the case of synchronous formalisms is not required from the designer, it is synthesized from the high-level correctness principles. Examples of such benefits are the *clock calculus* in Signal, and in our case the *constructive* semantics of Esterel, and the optimized mapping of application specified with these languages, onto architectural models of SynDEx.

Esterel [12][15] was developed jointly at INRIA and École des Mines de Paris, in the Meije research team then headed by Gérard Berry. The language is of imperative nature, with syntactic features for precise description of reactive instants, conceptual parallelism (potential parallelism), signal broadcast and preemption. Its scope is the representation of control-dominated reactive systems as hierarchical automata. Under a strict correctness condition of *constructive causality* [13] (signal presence values should be determined at any instant before it is tested), it can be given formal interpretation, either in the form of synchronous circuits, or as Mealy finite state machines. These, being classical mathematical models, allow design transformation activities such as optimization and automatic verification based on model-checking, and can produce target C code directly simulating the behavior (of the circuit or the Mealy machine respectively).

SyncCharts [1][7] are syntactically and semantically different from Statecharts (and UML State Machines). SyncCharts respect a strict state containment hierarchy, and transitions cannot cross a macrostate boundary. This should be imposed by a constraint on the structure. Where SyncCharts and UML State Machines greatly diverge is about their semantics. The execution semantics of UML State Machines is described in terms of the operations of a hypothetical machine that implements a state machine specification. Events are dispatched and processed by the state machine, *one at a time*. The UML specification (UML 1.4) mentions that "the order of dequeuing in not defined, leaving open the possibility of modeling different priority-based scheme". This does not open the possibility to handle simultaneous occurrences of events, which are the rule in synchronous models. Another cause of incompatibility is the *signal* concept. Signals in SyncCharts, like in Esterel, are the unique abstraction of communication and synchronization. Each signal has a presence status, and a possible value of a given type. When a signal can be emitted several times at the same instant, a special combination function must be associated with the signal. The broadcast/combine operation on signals is typical of the synchronous approach. In UML a signal is a special event, which cannot be taken as a super class for our signals.

SyncCharts were developed inside the SPORTS project-team at I3S (UNSA-CNRS). The synchronous semantics of SyncCharts can be contrasted with the numerous variants of Statecharts semantics, proposed by Harel, Pnueli and many others, in which microsteps and delayed reactions are introduced to avoid the constructive causality issue. Constructiveness is in fact a notion first studied by Sharad Malik (Princeton U.). Variants of Esterel borrowing syntax from general-purpose languages such as ECL (Esterel-C language) and Jester (Java-Esterel) were designed inside Cadence Berkely Labs in the context of the POLIS codesign/cosimulation project that ultimately led to the VCC product by Cadence. In Germany the SYNERGY project was conducted at GMD in Axel Poigné's group to build an environment merging features of Esterel, Lustre and Argos (a synchronous variant of StateCharts less expressive as SyncCharts, developed at VERIMAG). Work on foundational semantics of synchronous formalisms were also conducted in Germany (Quartz project of Klaus Schneider at Karlsruhe U.) and in the UK (Michael Mendler and Gerard Lüttgen, Sheffield U.). Recently, work on compilation schemes for Esterel (under additional conditions syntactically enforcing uniform causality) were developed in parallel at France Telecom Grenoble (Etienne Closse, Daniel Weil *et al.*), at Synopsys and then Columbia U. (Stephen Edwards), and the INRIA Tick project (PhD thesis of Dumitru Potop); all these works rely on intermediate models where control flow is made explicit from structural operators, but potential concurrency arising from parallelism is preserved (even though signal communication might restrict it to respect causal order); in our case the intermediate model is called a *GRC* graph (for "GRaph Code"). At this level important optimizations can be attempted based on static analysis techniques, thereby saving tremendous algorithmic work in later stages on flat circuit descriptions. We started work in that direction, which should be systematically pursued in AOSTE as a basis for distributed implementation satisfying real-time and embedding contraints.

Esterel/SyncCharts and Lustre/SCADE are now developed and commercialized in an industrial context by Esterel Technologies, an INRIA spin-off founded partially by former members of the Tick research group (and scientifically headed by Gérard Berry).

3.1.2. UML modeling diagrams for Real-Time Embedded applications

The UML consists of a variety of models (or "diagrams"), aiming at covering modeling concerns during the whole lifespan of software engineering. Of particular interest to us are some models of *structural* or *behavioral* nature, and in the later case *state* and *sequence* diagrams. State diagrams may represent components behaviors, in a way inspired from StateCharts. Sequence diagrams represent possible interaction scenarios between components, in a way inspired from Message Sequence Charts.

The only "semi-formal" semantics of models is usually given in natural language, with high risks of ambiguity and, even worse, inconsistencies; there is a uniform lack of clear relationship *between* the various models; currently, sequence diagrams have poor expressiveness. A number of research efforts address this demand for rigourous semantics of the UML models, such as the *Precise UML* group (http://www.cs.york.ac.uk/puml) or the Neptune (Nice Environment with a Process and Tools Using Norms and Example) project (http://neptune.irit.fr). *Umlaut* (J-M. Jezequel, INRIA Triskell team) is a UML transformation framework allowing complex manipulations to be applied to a UML model, where manipulations are expressed as algebraic compositions of reified elementary transformations.

The weak expressivity of sequence diagrams as models of interactions is currently tackled by researchers proposing extensions of *Message Sequence Charts* (MSC) to this end (thus outside the UML standardization community). Work on "High-Level MSCs" (for instance in INRIA Triskell and S4 teams) or on "Live Sequence Charts" (LSC, by D. Harel and W. Damm) fall into this category. With *LSCs* one can express possible, but also mandatory or even forbidden interaction scenarios. Prototypical tools at the University of Oldenburg/OFFIS provide semantic interpretation into timed automaton.

While the standard UML is aimed at general-purpose object-oriented software engineering, specific extensions (or "profiles") have been proposed to deal specifically with real-time aspects. Just to mention a few such: UML-RT (B. Selic) used in the Rational Rose-RT development environment, RT UML (B. Douglass) used in Rhapsody (I-Logix), and ACCORD/UML (F. Terrier, CEA). UML-RT is based on the first success story of ROOM (Real-time Object-Oriented Modeling), introduced in 1994 by Selic, Gullekson, and Ward, and put on the market by ObjecTime. The RTAD (Real-Time Analysis and Design) working group of the OMG has been especially created to promote real-time issues within the OMG, and to specialize the UML to be suitable for different real-time domains. The newly adopted UML-SPT profile is a first visible result, enabling models that support Scheduling, Performance, and Time evaluation. However, it fulfils different needs than ours (quantitative performance analysis rather than executable specifications). We are taking part in the elaboration of a Request-For-Proposal (RFP) for a new profile in this field, named MARTE (Modeling and Analysis of Real-Time Embedded systems). In this context we shall try if possible to promote models relevant to our goals.

There is a growing interest in integrating synchronous concepts into UML (or UML profiles). The former PAMPA and Ep-Atr teams (IRISA) once proposed the BDL formalism, to study a mixed synchronous-asynchronous semantics. The UML here plays the role of a federator notation. The I3S SPORTS project has adopted another point of view [3]: the direct use of synchronous (imperative) models. The question is whether these enrichments are "lightweight" (stereotypes or tag values) or "heavyweight" changes to the UML. Heavyweight here means that synchronous hypotheses are at places incompatible with some basic current assumptions made in UML. A UML state machine, which is a variant of Statecharts, has a queue for incoming events, an event dispatcher that selects and de-queues event instances one at a time, and an event processor which processes dispatched event instances under a run-to-completion scheduling policy. This definitely excludes simultaneous occurrences. Interaction models raise similar difficulties. Introducing our synchronous models to UML (SyncCharts as state-based model, and SIB as interaction model) would need changes at the meta model level.

The new **UML2.0** standard should improve the ability and utility of the UML with respect to architecture and scalability (through its "Superstructure" RFP). In the new version *classes* can be structured and reuse other classes playing specific "*parts*" roles. *Ports* are introduced for architectural modeling, as (instantiable) connection points through which *part* instance export specific services or operations accross the class boundary. *Interfaces* should be also expanded to allow specification of a *required interface* (from the distant other end) in addition to the usual notion of (local) *offered interface*. Moreover, the specification of allowable sets of sequences of service invocations might be specified with "protocol state machine". Almost all these new possibilities were present in the ROOM's *capsule* notion, a major influence. Such a model can play the role of an ADL (Architecture Description Language). Other improvements are related to behavioral models: *sequence diagrams* might now be broken up into "interaction fragments", with nesting capabilities and extended control constructs, making them closer to MSC and LSC. Last but not least, a form of *Data Flow Diagrams* should be introduced (since *activity diagrams* address only partially this issue).

To summarize, new UML trends meet our concerns about system architecture, components, and behavior. UML offers rich and standard notations, but lacks semantic rigor at places. This should not hinder our objectives of rigorous system design. Whenever an official model semantics will appear as not defined well enough, we shall feel free to adapt it: *strict UML compliance is not our goal !*

3.2. Transformations and analysis

Keywords: *Compilation, formal verification, synthesis*.

Participants: Robert de Simone, Fabrice Peix, Olivier Tardieu, Eric Vecchié.

The fact that syntactic constructs in synchronous languages are fully defined in terms of corresponding semantic operations immediately pays off in that all kinds of compilation/synthesis, analysis and verification, or optimization methods can readily be caracterized as formal transformations of mathematical models [47]. Therefore we realized, far in advance, the announced programme of Model-driven Architecture, in our case with true meaningful algorithmic transformations. To this end we extensively use a number of well-defined mathematical models, such as (hierarchical) finite-state Mealy machines, synchronous circuits as Boolean gate netlists, or data-control flowgraphs. The various steps of compilation (or static analysis, or dynamic analysis seen as model-checking, or optimization at each model level) are always formally defined, and thus *correct-by-construction*, discarding altogether issues of "synthetizability subsets", or discrepancy between simulation, that limit the application of syntactic formalisms without clear unambiguous semantics in the field.

3.2.1. Compilation/synthesis

Synchronous languages raise specific issues regarding efficient compilation of programs onto various targets (when producing hardware descriptions, one usually talks of "synthesis" rather than compilation). This is due to the strong demand on semantic preservation across models. The case of compilation onto distributed embedded architectures, where compilation requires an architecture model description and fancy mapping/scheduling techniques shall be further described as part of the AAA methodology.

Concerning Esterel/SyncCharts, compilation was first realized in the 1980's as an expansing into flat global Mealy FSMs; this produces efficient, but often unduly large code size. Then in the 1990's a translation was defined into Boolean equation systems (BES), with Boolean register memories encoding active control points. While such models are known in the hardware design community as Boolean gate *netlists*, they can be used in our context for software code production. Here the code produced in quasi-linear in size (worst-case quadratic in rare cases), but the execution consists in a linear evaluation of the whole equation system (thus each reaction requires an exceution time proportional to the whole program, even when only a small fragment is truly active). Thus in the early 2000's new implementation frameworks were thought, relying on high-level control-data flowgraphs selecting the active parts before execution at each instant. This scheme is both fast and memory-efficient, but cannot cope with all programs (as the full constructive causality analysis underlying the synchronous assumption cannot then be realized at "compile time", and this check is of utterly importance for program correctness). Correctnes is in this context ensured by a stronger, more restrictive *acyclicity* criterion, which provides a static evaluation order for signal propagation.

Some of the specific issues of Esterel/SyncCharts compilation are:
- the potential existence of *instantaneous loops*, diverging so that the instant never reaches completion ("non-zeno" behaviors);
- *shizophrenic* problems, where signals and other variables may assume several distinct values, and be instanciated multiply in the same reaction. This calls for program fragment *reincarnation* for efficient implementation (with single-static assignment techniques);
- *constructive causality* issues, when a static order to signal propagation cannot be defined, as this order may vary through time. This may call for the symbolic computation of the reachable state space (RSS) covering all potential control configurations, to establish computability of behaviors.

In fact these issues are sometimes *less* specific than can be thought on first glance, and the interplay between our specialized techniques, and more general compilation methods, is a subject of current studies. This si useful to help synchronous languages get rid of this false "niche topic" image. In particular, *static analysis* techniques are getting momentum in the justification of several compilation steps for Esterel, allowing to smoothly derive compiler specification from (extended) formal semantics, thus paving the way to true mathematical compiler certification.

One advantage of compilation schemes transforming programs from formal models to formal models is that one can benefit (and sometimes contribute) to a rich body of optimization and analysis techniques developed for these models. Circuit and FSM optimization/minimization, equivalence-checking and temporal logic model-checking are instances of this.

3.2.2. Dynamic analysis, automatic verification and model-checking

Formal finite state semantics paves the way to model-checking, which has reached its largest success in synchronous hardware verification. Here the most famous tool is SMV, a symbolic BDD-based model-checker developed at CMU (E. Clarke, K. McMillan), although a number of other similar tools exists, both in public domain or industrial proprietary versions. Recently a standardization of temporal property syntax named SUGAR was achieved at international level. In the past dedicated model-checkers were developed for all synchronous languages: XEVE for Esterel, LESAR/NBAC for Lustre, SIGALI for Signal. In all cases they are symbolic BDD-based model-checkers, avoiding temporal logic formalisms by stating generic properties, and adding synchronous parallel processes to test and monitor observed systems so that large classes of properties can be reduced to generic ones on combined systems. In the last decade model-checking techniques, mostly based on symbolic state space constructions, were extended to cover other problems than mere satisfaction of properties. In Esterel they were used to enforce “state-conscious” constructive causality, and in designing more efficient (but costly) optimization techniques on circuit descriptions. In Lustre, and then Esterel, they were used for automatic test pattern generation with the aim of covering specified portions of the state space (actually those techniques using explicit state spaces originated in the context of asynchronous languages, in the work of INRIA Pampa team for instance). In Signal they were extended to the problem of *controller synthesis*, in which a synchronous parallel supervisor is algorithmically built to keep the observed system from reaching pathological or forbidden states.

Recently, new techniques for “bounded” model-checking using efficient SAT-solvers were introduced (SAT being the well-known SATisfiability problem for propositional logic). They sacrify completeness (one must know “up to what depth” the property is meant to hold, or provide non-automatically some sort of recursion invariant assumption to be proved) to the sake of efficiency; in fact the economic pressure of hardware verification led to new advances in the algorithmic heuristics involved in SAT-solving methods, with the so-called Stalmark method in Sweden (by Prover Technologies), and iterative learning methods as implemented in Schaff (Malik, Princeton U.), BerkMin and predecessors.

3.3. Mapping onto Embedded platforms: The AAA methodology

Keywords: *RTL*, *Rapid prototyping*, *distributed*, *embedded*, *executive*, *fault tolerance*, *graph*, *hardware/software co-design*, *multiprocessor*, *off-line*, *on-line*, *optimization*, *parallel*, *partial order*, *partitioning*, *real-time*, *real-time operating system*, *real-time scheduling*, *specific integrated circuit*, *synchronous languages*, *system level CAD*.

Participants: Liliana Cucu, Thierry Grandpierre, Rémy Kocik, Christophe Lavarenne, Yves Sorel.

The AAA methodology (Algorithm-Architecture Adequation) allows to specify “application algorithms” (functionalities) and redundant “multicomponent architectures” (composed of processors and specific integrated circuits all together interconnected) with graph models. Consequently, all the possible implementations of a given algorithm onto a given architecture is described in terms of graphs transformations. An implementation consists in distributing and scheduling a given algorithm onto a given architecture. Adequation amounts to chose one implementation among all the possible ones, such that the real-time and embedding constraints are satisfied and the hardware redundancy is fully used. Furthermore, from the adequation results our graph models allow to generate automatically, as an ultimate graphs transformation, two types of codes: dedicated distributed real-time executives or configuration of standard distributed real-time executives (RTlinux, OSEK, etc) for processors, and net-lists (structural VHDL) for specific integrated circuits. Finally fault tolerance is of great concern because the applications we are dealing with are often critical, that is to say, may lead to catastrophic consequences when they fail. The AAA methodology provides a mathematical framework for rapid prototyping and hardware/software co-design taking into account fault tolerance.

From the optimization point of view, real-time systems are, first of all, “reactive systems” which mandatorily must react to each input event of the infinite sequence of events it consumes, such that “cadence” and “latency” constraints are satisfied. The latency corresponds to the delay between an input event consumed by the system and an output event produced by the system in reaction to this input event. The cadence corresponds to the delay between two successive input events, i.e. a period. The term event is used in a broad sense, it may refers to a periodic or to an aperiodic discrete (sampled) signal. When hard (critical) real-time is considered, off-line approaches are preferred due to their predictability and best performances, and when on-line approaches are unavoidable, mainly to take into account aperiodic events, we intend to minimize the decisions taken during the real-time execution. When soft real-time is considered off-line and on-line approaches are mixed. The application domains we are involved in, e.g. automobile, avionic, lead to consider scheduling problems for systems of tasks with precedence, latency and periodicity constraints. We seek optimal results in the mono-processor case where distribution is not considered, and sub-optimal results through heuristics in the multiprocessor case, because the problems are NP-hard due to distribution consideration. Also, in addition to these timing constraints, embedded systems must satisfy technological constraints, such as power consumption, weight, volume, memory, etc, leading in general to minimize hardware resources. In the most general case architectures are distributed, and composed of several programmable components (processors) and several specific integrated circuits (ASIC[1] or FPGA[2]) all together interconnected with possibly different types of communication media. We call such heterogeneous architectures “multicomponent” [36].

The complexity, not only of the algorithms that must be implemented, but also of the hardware architectures, and also the multiple constraints, imply to use methodologies when development cycle time must be minimized from the high level specification until the successive prototypes which ultimately will become a commercial product. In order to avoid gaps between the different steps of the development cycle our AAA methodology is based on a global mathematical framework which allows to specify the application algorithms as well as the hardware architecture with graph models, and the implementation of algorithms onto architectures in terms of graphs transformations. This approach has the benefit on the one hand to insure traceability and consistency between the different steps of the development cycle, and on the other hand to perform formal verifications and optimizations which decrease real-time tests, and also to perform automatic code generation (real-time executives for processors and net-list for specific integrated circuits). All these benefits contribute to minimize the development cycle. Actually, the AAA methodology provides a framework for hardware/software co-design where safe design is achieved by construction, and automatic fault-tolerance is possible only by specifying the components that the user accepts to fail.

To summarize, we are interested in the optimization of distributed real-time embedded systems according to four research topics:

1. models for specifying, with graphs and partial orders, application algorithm, hardware architecture, and optimized implementation,
2. implementation optimization:

   – real-time scheduling algorithms in the case of mono-processor,
   – real-time distribution and scheduling heuristics in the case of multiprocessor,
   – heuristics for resources minimization in the case of multiprocessor and specific integrated circuit,

[1]ASIC : Application Specific Integrated Circuit
[2]FPGA : Field Programmable Gate Array

3. automatic code generation for processor (dedicated or standard RTOS configuration) and for specific integrated circuit (net-list),
4. fault tolerance.

Beside these researches, we propose a tool implementing the AAA methodology. It is a system level CAD software called SynDEx (http://www.syndex.org). This software, coupled with a high level specification language, like one of the Synchronous Languages or Scicos, leads to a seamless environment allowing to perform rapid prototyping and hardware/software co-design while reducing drastically the development cycle duration and providing safe design.

3.3.1. Algorithm/Architecture/Implementation models

3.3.1.1. Algorithm

Our algorithm model is an extension of the well known data-flow model from Dennis [58]. It is a directed acyclic hyper-graph (DAG) [57] that we call "conditioned factorized data dependence graph" [32], whose vertices are "operations" and hyper-edges are directed "data or control dependences" between operations. Hyper-edges are necessary in order to model data diffusion since a standard edge only relates a pair of operations. The data dependences defines a partial order on the operations execution [67], called "potential operation-parallelism". Each operation may be in turn described as a graph allowing a hierarchical specification of an algorithm. Therefore, a graph of operations is also an operation. Operations which are the leaves of the hierarchy are said "atomic" in the sense that it is not possible to distribute each of them on more than one computation resource. The basic data-flow model was extended in three directions, firstly infinite (resp. finite) repetitions in order to take into account the reactive aspect of real-time systems (resp. "potential data-parallelism" similar to loop or iteration in imperative languages), secondly "state" when data dependence are necessary between repetitions introducing cycles which must be avoided by specific vertices called "delays" (similar to z^{-n} in automatic control), thirdly "conditioning" of an operation by a control dependence similar to conditional control structure in imperative languages. Delays combined with conditionings allow to specify FSM (Finite State Machine) necessary for specifying "mode changes", e.g. some control law is performed when the motor is the state "idle" whereas another one is performed when it is in the state "permanent". Repetition and conditioning are both based on hierarchy. Indeed, a repeated or "factorized graph of operations" is a hierarchical vertex specified with a "repetition factor" (factorization allows to display only one repetition). Similarly, a "conditioned graph of operations" is a hierarchical vertex containing several alternative operations, such that for each infinite repetition, only one of them is executed, depending on the value carried by the "conditioning input" of this hierarchical vertex. Moreover, the proposed model has the synchronous language semantics [60], i.e. physical time is not taken into account. This means that it is assumed an operation produces its output events and consumes its inputs events simultaneously, and all the input events are simultaneously present. Thus, by transitivity of the execution partial order associated to the algorithm graph, outputs of the algorithm are obtained simultaneously with its inputs. Each input or output carries an infinite sequence of events taking values, which is called a "signal". Here, the notion of event is general, i.e. signals may be periodic as well as aperiodic. The union of all the signals defines a "logical time", where physical time elapsing between events are not considered.

3.3.1.2. Architecture

The typical coarse-grain architecture models such as the PRAM (Parallel Random Access Machines) and the DRAM (Distributed Random Access Machines) [68] are not enough detailed for the optimizations we intend to perform. On the other hand the very fine grain RTL-like (Register Transfer Level) [66] models are too detailed. Thus, our model of multicomponent architecture is also a directed graph [23], whose vertices are of four types: "operator" (computation resource or sequencer of operations), "communicator" (communication resource or sequencer of communications, e.g. DMA), memory resource of type RAM (random access) or SAM (sequential access), "bus/mux/demux/(arbiter)" (choice resource or selection of data from or to a memory) possibly with arbiter (arbitration of memory accesses when the memory is shared by several operators), and whose edges are directed connections. For example, a typical processor is a graph composed of an operator, interconnected with memories (program and data) and communicators, through bus/mux/demux/(arbiter). A "communication medium" is a linear graph composed of memories, communicators, bus/mux/demux/arbiters corresponding to a "route", i.e. a path in the architecture graph. Like for the algorithm model, the architecture model is hierarchical but specific rules must be carefully observed, e.g. a hierarchical memory vertex may be specified with bus/mux/demux and memories (e.g. several banks), but not with operator. Although this model seems very basic, it is the result of several studies in order to find the appropriate granularity allowing, on the one hand to provide accurate optimization results, and on the other hand to quickly obtain these results during the rapid prototyping phase. Data communications can be precisely modeled through shared memory or through message passing possibly using routes. Furthermore, complex interactions between operators and communicators can be taken into account through bus/mux/demux/arbiter, e.g. when communications with DMA require the sequencer of a processor.

Our model of integrated circuit architecture is the typical RTL model. It is a directed graph whose vertices are of two types: combinatorial circuit executing an instruction, and register storing data used by instructions, and whose edges are data transfers between a combinatorial circuit and a register, and reciprocally.

In order to unify both multicomponent and integrated circuit models we extend the RTL model in a new one called "macro-RTL". Thus, an operator executes "macro-instructions", i.e. operations, which consume and produce data in "macro-registers". This model allows to encapsulate specific details related to the instructions set such as cache, pipe-line and other non deterministic features of processors that are difficult to take into account.

3.3.1.3. Implementation

An implementation of a given algorithm onto a given multicomponent architecture corresponds to a distribution and a scheduling of, not only the algorithm operations onto the architecture operators, but also a distribution and a scheduling of the data transfers between operations [25].

The distribution consists in distributing each operation of the algorithm graph onto an operator of the architecture graph. This leads to a partition of the operations set, in as much as sub-graphs that there are of operators. Then, for each operation two vertices called "alloc" for allocating program (resp. data) memory must be added, and each of them is allocated to a program (resp. data) RAM connected to the corresponding operator. Moreover, each "inter-operator" data transfer between two operations distributed onto two different operators, is distributed onto a route connecting these two operators. In order to actually perform this data transfer distribution, according to the element composing the route we create and insert as much as "communication operations" that there are of communicators, as much as "identity" vertices that there are of bus/mux/demux, and as much as "alloc" vertices for allocating data to communicate that there are of RAM and SAM. Finally, communication operations, identity and alloc vertices are distributed onto the corresponding vertices of the architecture graph. All the alloc vertices, those for allocating data and program memories as well as those for allocating data to communicate, allow to determine the amount of memory necessary for each processor of the architecture.

The scheduling consists in transforming the partial order of the corresponding subgraph of operations distributed onto an operator, in a total order. This "linearization of the partial order" is necessary because an operator is a sequential machine which executes sequentially the operations. Similarly, it also consists in transforming the partial order of the corresponding subgraph of communications operations distributed onto a communicator, in a total order. Actually, both schedulings amount to add edges, called "precedence dependences" rather than data dependences, to the initial algorithm graph. To summarize, an implementation corresponds to the transformation of the algorithm graph (addition of new vertices and edges to the initial ones) according to the architecture graph.

Finally, the set of all the possible implementations of a given algorithm onto a given architecture may be modeled, in intention, as the composition of three binary relations: namely the "routing", the "distribution", and the "scheduling" [38]. Each relation is a mapping between two pairs of graphs (algorithm graph, architecture graph). It also may be seen as a external compositional law, where an architecture graph operates on an algorithm graph in order to give, as a result, a new algorithm graph, which is the initial algorithm graph distributed and scheduled according to the architecture graph. Then the implementation graph is of type algorithm which may in turn be composed with another architecture graph, allowing complex combinations.

The set of all the possible implementations of a given algorithm onto a specific integrated circuit is different because we need a transformation of the algorithm graph into an architecture graph which is directly the implementation graph. This graph is composed of two parts: the data-path obtained by translating each operation in a corresponding logic function, and the control path obtained by translating each control structure in a “control unit”, which is a finite state machine made of counters, multiplexers, demultiplexers and memories, managing repetitions and conditionings [18].

3.3.2. Optimization

We must choose among all the possible implementations a particular one for which the constraints are satisfied and possibly some criteria are optimized.

In the case of multiprocessor architecture the problem consisting in distributing and scheduling the algorithm onto the architecture such that the execution time of the algorithm is minimum, is known to be of NP-hard complexity [63]. This amounts to consider, in addition to precedences constraints specified through the algorithm graph model, one latency constraint between the first operation(s) (without predecessor) and the last operation(s) (without successor), equal to a unique periodicity constraint (cadence) for all the operations. We propose several heuristics based on the characterization of the operations (resp. communication operations) relatively to the operators (resp. communicators), e.g. execution durations of operations and data transfers, amount of memory, etc, in order to minimize the execution duration of the algorithm graph on the multiprocessor architecture, taking into account communications, possibly concurrent [25]. The characterization amounts to relate the logical time described by the interleaving of events with the physical time. We prefer “greedy heuristics” because they are very fast [64] giving results in a time well suited to rapid prototyping of realistic industrial applications. In this type of applications the algorithm graph may have five thousand vertices and the architecture graph may have some tens of vertices. We also extend these greedy heuristics to iterative versions [37] which are much slower, due to back-tracking, but give better results for the final commercial product.

New applications in the automobile, avionic, or telecommunication domains, lead us to consider more complex constraints. In such applications it is not sufficient to consider the execution duration of the algorithm graph. We need also to consider periodicity constraints for the operations, possibly different, and several latency constraints imposed possibly on whatever pair of operations. Presently there are only partial and simple results for such situations in the multiprocessor case, and only few results in the mono-processor case. Then, we began few years ago to investigate this research area, by interpreting, in our algorithm graph model, the typical scheduling model given by Liu and Leyland [65] for the mono-processor case. This leads us to redefine the notion of periodicity through infinite and finite repetitions of an operations graph (i.e. the algorithm), thus generalizing the SDF (Synchronous Data-Flow) model [61] proposed in the software environment Ptolemy. For simplicity reason and because this is consistent with the application domains we are interested in, we presently only consider that our real-time systems are non-preemptive, and that “strict periodicity” constraints are imposed on operations. In this case we give a schedulability condition for graph of operations with precedence and periodicity constraints in the non-preemptive case [17]. We also formally define the notion of latency which is more powerful [16], for the applications we are interested in, than the usual notion of “deadline” that does not allow to impose directly a timing constraint on a pair of operations, connected by at least one path, like it is necessary for “end-to-end constraints”. In order to study schedulability conditions for multiple latency constraints we defined three relations between pair of paths, such that for each pair a latency constraint is imposed on its extremities. Using these relations called II, Z and X, we give a schedulability condition for graph of operations with precedence and latency constraints in the non-preemptive case. Then by combining both previous results we give a schedulability condition for graph of operations with precedence, periodicity and latency constraints in the non-preemptive case, using an important result which gives a relation between periodicity and latency. We also give an optimal scheduling algorithm in the sense that, if there is a schedule the algorithm will find it.

On the other hand, thanks to these results obtained in the mono-processor case, we study extensions of our heuristics for one latency constraint equal to a unique periodicity constraint, in order to solve the distribution and scheduling problem for graph of operations with precedence, periodicity and latency constraints in the multiprocessor case. However, the aforementioned scheduling problems do not take into account aperiodic operations for which there is no off-line solution, at least to our best knowledge, but there are on-line solutions. These aperiodic operations come from aperiodic events, usually related to control. Presently we take them into account off-line by integrating the control-flow in our data-flow model, well suited to distribution, and by maximizing the control effects. We study relations between control-flow and data-flow in order to better exploit their respective advantages. Finally, for soft real-time applications, we study the possibilities in order to mix off-line and on-line approaches in order to take benefit of a better cooperation of control-flow and data-flow.

In the case of integrated circuit the potential parallelism of the algorithm corresponds exactly to the actual parallelism of the circuit. However, this may lead to exceed the required surface of an ASIC or the number of CLB (Combinatorial Logic Block) of a FPGA, and then some operations must be sequentially repeated several times in order to reuse them, reducing in this way the potential parallelism to an actual parallelism with less logic functions. But reducing the surface has a price in terms of time, and also in terms of surface but of a lesser importance, due to the sequentialization itself (instead of parallelism) performed by the finite state machines (control units) necessary to implement the repetitions and the conditionings. Then, we are seeking a compromise taking into account surface and performances. Because these problems are again of NP-hard complexity, we propose greedy and iterative heuristics in order to solve them [18].

Finally, we plan to work on the unification of multiprocessor heuristics and integrated circuit heuristics in order to propose "automatic hardware/software partitioning" for co-design, instead of the usual manual one. The most difficult issue concerns the integration in the cost functions of the notion of "flexibility" which is crucial for the choice of software versus hardware.

3.3.3. Automatic code generation

As soon as an implementation is chosen among all the possible ones, it is straightforward to automatically generate executable code through an ultimate graphs transformation leading to a distributed real-time executive for the processors, and to a structural hardware description, e.g. synthetizable VHDL, for the specific integrated circuits.

For a multicomponent each operator (resp. each communicator) has to execute the sequence of operations (resp. communication operations) described in the implementation graph. Thus, this graph is translated in an "executive graph" [26] where new vertices and edges are added in order to manage the infinite and finite loops, the conditionings, the inter-operator data dependences corresponding to "read" and "write" when the communication medium is a RAM, or to "send" and "receive" when the communication medium is a SAM. Specific vertices, called "pre" and "suc", which manage semaphores, are added to each read, write, send and receive vertices in order to synchronize the execution of operations and of communication operations when they must share, in mutual exclusion, the same sequencer as well as the same data. These synchronizations insure that the real-time execution will satisfy the partial order specified in the algorithm. Executives generation is proved to be dead-lock free [23] maintaining the properties, in terms of events ordering, shown thanks to the synchronous language semantics. This executive graph is directly transformed in a macro-code [24] which is independent of the processor. This macro-code is macro-processed with "executive kernels" libraries which are dependent of the processors and of the communication media, in order to produce as much as source codes that there are of processors. Each library is written in the best adapted language regarding the processors and the media, e.g. assembler or high level language like C. Finally, each produced source code is compiled in order to obtain distributed executable code satisfying the real-time constraints.

For an integrated circuit, because we associate to each operation and to each control unit an element of a synthetizable VHDL library, the executable code generation relies on the typical synthesis tools of integrated circuit CAD vendors like Synopsis or Cadence.

3.3.4. Fault tolerance

For the applications we are dealing with, if real-time constraints are not satisfied, this may have catastrophic consequences in terms of human beings lost or pollution, for example. When a fault occurs despite formal verifications which allow safe design by construction, we propose to specify the level of fault the user accepts by adding redundant processors and communication media. Then, we extended our optimization heuristics in order to generate automatically the redundant operations and data dependences necessary to make transparent these faults [21]. As soon as the redundant hardware is fully exploited, “degraded modes” are necessary. They are specified at the level of the algorithm graph using conditionings. Presently, we only take into account “fail silent” faults. They are detected using “watchdogs”, the duration of which depends on the operations and data transfers durations. We obtained solid results in the case of processor failures only, i.e. in this case the communication media are assumed error free. We propose two approaches in order to achieve this goal. The first approach is based on spatial redundancy, of operations and data transfers, for point-to-point communication media [22]. The second approach is based on spatial redundancy of operations and temporal redundancy of data transfers for multi-point communication media.

4. Application Domains

4.1. Embedded systems

Our generic field of applications is termed as “Embedded Systems”, meaning all kind of equipments including software and electronical parts, apart from regular mainstream computers. This includes transportation vehicles (cars, aircrafts,...), mobile robotics, communicating appliances such as mobile telephones, or System-on-Chip design. These fields are further described in their particular aspects below.

Common to all embedded systems are: the *reactivity* aspect (supervising or simply interacting with an outside environment); the demand for *safety* (often critical); the *heterogeneity of models*, incumbing a multiplicity of engineering techniques from different scientific disciplines. Large development projects usually involve many providers for components, and the design flow is far less linear as in traditional software: it includes customarily various stages of modeling, prototyping, simulation/evaluation/dimensioning, manual reencoding, and iterative (re)design space exploration. Component reuse is also usually a great concern (as components might be physical preexisting parts). Here formal models and notations can greatly help keep the traceability of the design process, and justify some of its steps for soundness and accurary across the flow and its various actors.

4.2. Mobile robotics, automotive and transportation

Keywords: .., *embedded automotive electronics*.

With increasing functionality demands in powertrain, body comfort or telematics applications, modern cars are becoming complex systems including Real-Time OS (OSEK), complex data buses (CAN or TTP/FlexRay), with distributed intelligent sensors and powerful computing power. Software and electronic are now becoming a prominent part of both the price and added value. Still, no “ideal” hardware/software architecture is yet standardized, and the development methodologies are still at infancy. Proposals for high-level modeling and infrastructure platform organization have been proposed, as in the EAST-EEA and AutoSar consortium. We are taking part in the first initiative, mostly through the AAA methodology which proposes computer-aided mapping of synchronous applications onto heterogeneous platforms. This methodology was amply demonstrated in the framework of CyCab mobile robotic applications.

4.3. Mobile phones and other communicating objects

Such systems usually combine intensive (multimedia) data processing with mode control switching, and wireless or on-chip communications. The issue is often here to integrate design techniques for the three domains (data, control, communications), while preserving modular independence of concern as much as possible. At high-level modeling this translate into combining models of computations that are state-oriented (imperative) or datapath-oriented (declarative), with appropriate communication models, while preserving the sound semantics of the systems built from all such kinds of components. Dedicated architecture platforms here usually associate general-purpose processor (ARM) with specific DSP coprocessors. In the future the level of integration should become even higher, with the corresponding challenges for design methodologies.

4.4. System-on-Chip design

While design of digital circuits is already a fairly complex development process, involving many modeling and programming stages, together with intensive testing and involved low-leval synthesis and place-and-route techniques, SoC desig adds yet new complexity dimensions to this process. Fully synchronous designs are not feasible anymore, and custon IP reuse becomes mandatory to integrate full processor cores into a new designs. New aproaches are being proposed, which try to depart only as little as possible form the synchronous/cycle-accurate prevailing design techniques, while allowing more timing flexibility at interfaces between blocks. These aaproaches are generally flagged as GALS (Globally-Asynchronous/Locally-Synchronous). They usually put a stress on proper mathematical modeling at every stage, thereby revisiting and associating known models with new intent. Synthesis seen as model-transformation seems here a nice way to bring some of the OMG MDA schemes into true existence.

5. Software

5.1. SyncCharts/Esterel

Keywords: *Esterel*, *SyncCharts*, *compilation*, *static analysis*, *verification*.

Participants: Charles André, Robert de Simone, Olivier Tardieu, Eric Vecchié.

The main software development activities concerning synchronous formalisms went to the ESTEREL TECHNOLOGIES company as it was spun-off from the former Meije team. We still carrry some experimental development on the former academic versions of Esterel and SynCharts, mostly to validate new algorithmic model transformations or analysis.

5.2. SynDEx

Participants: Julien Forget, Arnaud Rouanet, Yves Sorel.

SynDEx is a system level CAD software implementing the AAA methodology for rapid prototyping and for optimizing distributed real-time embedded applications. It can be downloaded free of charge, under INRIA copyright, at the url: http://www.syndex.org. It provides the following functionalities:

- specification and verification of an application algorithm as a directed acyclic graph (DAG) of operations, or interface with specification languages such as the synchronous languages providing formal verifications, AIL a language for automobile architectures, Scicos a Simulink-like language, AVS for image processing, CamlFlow a functional data-flow language, etc,
- specification and verification of a "multicomponent" architecture as a graph composed of programmable components (processors) and/or specific non programmable components (ASIC, FPGA), all interconnected through communication media (shared memory, message passing),
- specification of the algorithm characteristics, relative to the hardware components (execution and transfer time, period, memory, etc), and specification of the real-time constraints to satisfy (latencies, periodicities),
- exploration of possible implementations (distribution and scheduling) of the algorithm onto the multicomponent, performed manually or automatically with optimization heuristics, and visualization of a timing diagram simulating the distributed real-time implementation,

- generation of dedicated distributed real-time executives, or configuration of general purpose real-time operating systems: RTlinux, Osek, etc. These executives are deadlock free and based on off-line policies. Dedicated executives which induce minimal over-head are built from processor-dependent executive kernels. Presently executives kernels are provided for: ADSP21060, TMS320C40, TMS320C60, i80386, MC68332, MPC555, i80C196 and Unix/Linux workstations. Executive kernels for other processors can be easily ported from the existing ones.

The distribution and scheduling heuristics, as well as the timing diagram, help the user to parallelize his algorithm and to explore architectural solutions while satisfying real-time constraints. Since SynDEx provides a seamless framework from the specification to the distributed real-time execution, formal verifications obtained during the early stage of the specification, are maintained along the whole development cycle. Moreover, since the executives are automatically generated, part of tests and low level hand coding are eliminated, decreasing the development cycle duration.

SynDEx was evaluated by the main companies involved in distributed real-time embedded systems, and is presently used to carry out new applications at Robosoft, MBDA and Mitsubishi Electric ITE.

5.3. SynDEx-IC

Participants: Mohamed Akil [Professor ESIEE Noisy-Le-Grand], Julien Forget, Thierry Grandpierre, Linda Kaouane, Pierre Niang [Ph. D. Student ESIEE Noisy-Le-Grand], Yves Sorel.

SynDEx-IC is a CAD software for the design of non programmable components such as ASIC or FPGA for which the application algorithm to implement is specified with the graph model of the AAA methodology. It is developed in collaboration with the team A2SI of ESIEE. It allows to specify the application algorithm like in SynDEx and automatically synthesizes the data path and the control path of the specific integrated circuit as a synthetizable VHDL program while real-time and surface constraints are satisfied. Because these problems are again of NP-hard complexity, we propose greedy and iterative heuristics based on "loop-unrolling" of the algorithm graph, in order to solve them. Non programmable components designed with SynDEx-IC may be in turn used in SynDEx in order to specify complex multicomponent architectures composed of non programmable and programmable components all together interconnected. Presently, both softwares SynDEx and SynDEx-IC are separated, consequently the hardware/software partitioning of co-design must be done manually. We plan in the future to integrate them in an unique software environment, and also to provide heuristics to automatically carry out hardware/software partitioning.

5.4. Sep

Keywords: *Architecture*, *component*, *simulation*, *validation*.

Participant: Frederic Mallet.

SEP is an object-oriented modeling and simulation environnement that allows for an incremental modeling of hardware architectures at different levels of abstraction. Validation of a given software execution on this hardware is mainly performed using simulation. But, more sensitive parts that require more formal approaches are validated using algebraic simulations and when possible interoperability with synchronous models (Esterel, SyncCharts or S-Grafcet). Most of the Sep methodology has been transferred into the JavaHase environment which is currently developed at the University of Edinburgh. A local copy is maintained to validate new UML-based model transformations.

6. New Results

6.1. Syntax-driven model checking

Participants: Robert de Simone, Eric Vecchié.

We considered the issue of exploiting the structural form of ESTEREL programs to partition the algorithmic Reachable State Space fix-point construction used in model-checking techniques. The basic idea sounds utterly simple, as seen on the case of sequential composition: in $P;Q$, first compute entirely the states reached in P, and then only carry on to Q, each time using only the relevant transition relation part. Difficulties appear in our decomposition approach when scheduling the different parts of the transition relation in presence of parallelism and local signal exchanges. Program blocks (or “Macro-states”) put in parallel can be synchronized in various ways, due to dynamic behaviors, and considering all possibilities may lead to an excessive division complexity. The goal was here to find a satisfactory trade-off between compositional and global approaches.

We proposed an efficient algorithm for this type of partitionned model-checking, based on identified frontiers, computed by static analysis of the structural program description. Frontiers are withhold progressively to compute the full RSS in a highly constrained way, each time using only local transitional parts.

This research was the main topic of Eric Vecchié PhD thesis [42], defended in July 2004. The results were presented by Eric at the SLAP workshop [56], and a longer version is now accepted for journal publication [46]. A BDD-based model-checker prototype has been implemented, showing true promising experimental gain on large examples.

6.2. Esterel: from formal semantics to provably correct compilers

Participants: Robert de Simone, Olivier Tardieu.

Based on the synchronous paradigm, Esterel semantics relies on a clear distinction of instants of computation. All primitives of the language, safe the `pause` unit-cycle delay instruction, execute in zero time. Execution is thus a sequence of instantaneous computations separated by explicit pauses. Arbitrary loops in this context are troublesome, potentially leading to a non-termination problem or a schizophrenia issue: first, instantaneous loops may prevent the instant to end; second, program blocks may be traversed several times within the same instant, thus having a "schizophrenic" behavior. Instantaneous loops are forbidden by the semantics. Such errors have to be anticipated, and programs rejected by compilers on this behalf. Moreover, efficient code generation for schizophrenic program patterns is complex. While many existing compilers already generate correct code for loops, the efficient implementations available today are neither generic (i.e. target-independent) nor formally specified or verified.

We thoroughly considered loop handling in Esterel, starting from the operational semantics of the language, all the way down to a provably correct implementation. We formally characterized the related issues and define efficient static analysis techniques to detect them in Esterel code. In order to get rid of schizophrenic behaviors by source-to-source rewriting - "cure schizophrenia" - we introduced in the Esterel language a new primitive, which we call "`gotopause`". It behaves as a non-instantaneous jump instruction compatible with concurrency. We described a first program transformation that systematically replaces loops by the mean of gotopause statements, providing a loop-free equivalent program for any correct Esterel program. By combining static analysis and rewriting techniques, we obtained a preprocessor for Esterel that rejects incorrect loops and cure schizophrenia, which we have implemented. Due to our source-to-source transformation methodology, our preprocessor is highly generic; because of static analysis, it is very efficient; thanks to our fully formalized approach, we could formally establish its correctness. We view this as an important example of deriving actual compiler specifications from the formal language semantics, by extracting static analysis techniques which focus on specific steps on the compilation process.

These studies were reported in Olivier Tardieu’s PhD thesis [41], as well as in conference or journal publications [55][53][54][48]

6.3. Reaction to Absence in a distributed context

Participants: Robert de Simone, Fabrice Peix.

The translation of Esterel programs to SynDEx poses a specific problem, coming from the fact the the same signal event can be emitted from several distinct locations, some active and some not.Here the issue of efficient mapping involves the problem of determining in a distributed framework which emissions are still feasible. We studied this problem, and proposed extensions to SynDEx in order to cope with the issue in a clean fashion while fully exploiting SynDEx conditioning operators. The results were reported in part as Fabrice Peix PhD thesis [40].

6.4. SyncCharts improvements

Participants: Charles André, Julien Boucaron, Daniel Gaffé (I3S Lab.).

An XML format of SyncCharts, defined by an XML Schema, is now available, and thus facilitates model reuse in other applications.

SynDEx has been the first software to make use of this capability in interfacing. The experimental compiler of SyncCharts developed by Julien Boucaron and Daniel Gaffé also takes this format as input and generates a circuit without intermediate translation into the Esterel language. This direct translation has several advantages: a better use of the structure of the syncChart, a more efficient translation, and is independent of the commercial Esterel compiler. However, this compiler suffers from some limitations and has been applied only to toy examples. This study should be pursued.

6.5. UML Patterns for hardware/software architectures

Participants: Frédéric Mallet, Charles André.

In the field of Real-Time Embedded systems, the research community is looking for a methodology to model at different levels of abstraction (from Transaction Level down to RTL) both hardware and software parts of architectures. This approach needs to provide mechanisms allowing for reusability of existing models, interoperability with existing tools, formal verification of composite behaviours, deployment on actual systems.

Several UML-based approaches (SysML, UML4Soc, AADL) have been proposed to address these issues, either providing UML-profiles or meta-models. These approaches always provide mechanisms to describe the structural part of systems and either rely on languages like SystemC for the behaviour part or gives some description of the behaviour but without any strong links to connect with the structural description.

As a first step toward a synchronous UML-profile for Real-Time Embedded Systems, we proposed a set of analysis patterns that allow for annotation of existing hardware and software models. These patterns contain both a structural and a behavioural description of different identified actors, these two descriptions being very closely related. Additionnally, as much as possible, UML activity diagrams used to describe the behaviour are stereotyped to give an equivalent execution semantics with synchronous models.

In a short future, we intend to use these patterns to facilitate interoperability between existing models and tools used in the synchronous community.

6.6. AAA models

Participants: Liliana Cucu, Julien Forget, Nicolas Pernet, Mickaël Raulet, Yves Sorel.

We studied the differences between our algorithm model and the typical model used by the real-time community in the mono-processor case. We justified that “release time” and “deadline” are not necessary and that “strict periodicity” is sufficient, for the applications of signal, image and control processing we are interested in. Also, we justified that “multiple latency constraints” are more powerful than “deadlines” for the same applications. We studied how to introduce the “preemption” in our algorithm model to take into account its cost in off-line scheduling.

We proposed a new version of the architecture model which takes into account program and data memories. This model is simpler than the hierarchical one described in [23] but is sufficient to be exploited by the code generator in order to minimize the number of buffers, and then the amount of memory when an embedded processor is targetted.

We studied the possibility to modify the implementation model in order to take benefit, during the adequation and the code generation, of the capability to specify repetitive sub-graphs in the algorithm model. Presently the user will have to explicitely specify whether he wants to minimize code generation, and in this case each repetitive sub-graph will be coded as a loop. If it is not the case the adequation will exploit parallelism as much as possible and the code generator operates as usual.

6.7. Scheduling and Optimization

Participants: Liliana Cucu, Omar El Ganaoui, Julien Forget, Nicolas Pernet, Yves Sorel.

6.7.1. Off-line scheduling

We continued the studies on non-preemptive real-time systems with precedence, periodicity and latency constraints in the case of several operators, i.e. several processors [39].

After proving the NP-hardness of our scheduling and distribution problem for systems with precedence and periodicity constraints, we proposed a heuristic which takes into account the communication times. We proved that operations with periods which are not co-prime can not be scheduled on the same operator. Therefore, we defined classes of operations which may be scheduled on the same operator relatively to the periodicity constraints. These classes are not mutually exclusive. In order to schedule operations which may be scheduled on the same operator, the heuristic uses the main idea of the algorithm we proposed in the case of one operator (monoprocessor case) [17]. This way the first repetition of an operation is scheduled before the first repetitions of all operations which have a larger period [49].

After proving the NP-hardness of our scheduling and distribution problem for systems with precedence and latency constraints, we proposed a heuristic which takes into account the communication times. This heuristic uses the schedulability results obtained in the case of one operator concerning the three relations *II*, *Z* and *X* between pairs of operations, on which latency constraints are defined. These latter results prove that the best way of scheduling operations is to avoid scheduling, between the first and the last operation of a latency constraint, operations which do not belong to this latency constraint. Therefore, we defined classes of operations which may be scheduled on the same operator relatively to the latency constraints. These classes are not mutually exclusive. In order to schedule operations which may be scheduled on the same operator, the heuristic uses the main idea of the algorithm we proposed in the case of one operator [50]. This way we used a marking algorithm which allows us to "predict" the operations which are important for the latency constraints. The available operations are scheduled on the same operator in the increasing order of their marks.

After proving the NP-hardness of our scheduling and distribution problem of systems with precedence, periodicity and latency constraints, we proposed a heuristic which takes into account the communication times. We proved that operations belonging to the same latency constraint must have the same period. A direct consequence is that the operations belonging to the same pair or to pairs which are in relation *II*, *Z* or *X* must have the same period. So, the heuristic may use the main ideas of the heuristic for the case of precedence and latency constraints and of the heuristic for the case of precedence and periodicity constraints. Therefore, we defined classes of operations which may be scheduled on the same operator relatively to the periodicity constraints and the latency constraints. These classes are not mutually exclusive. The operations which belong to the same latency constraint are scheduled on the same operator as the operations with periods not co-prime with the period of operations belonging to the latency constraint. This way the first repetition of an operation is scheduled before the first repetitions of all operations which have a larger period and a smaller mark.

The performances of these three heuristics were compared to those of exact algorithms. The numerical results show that the heuristics are definitely faster then the exact algorithms for all cases when the heuristics find a solution.

We continued also our studies on preemtive real-time systems with precedence, periodicity and latency constraints in the monoprocessor case. We proved that it is not possible to find a schedule which verifies all the constraints for operations with co-prime periods. Then, in this case we proved that the scheduling policy which schedules operations according to the increasing order of their periods is sufficient. With these restrictions we proposed a scheduling algorithm which gives the number of preemtions to introduce.

6.7.2. Mixing off-line and on-line scheduling

The AAA methodology is presently based on off-line scheduling because it allows to be deterministic, and thus consistent with the synchronous semantics which provides formal verifications. This approach is perfectly suited for hard real-time systems. Moreover, it induces a very low overhead which is crucial in embedded systems. The main drawback of off-line scheduling is that it does not allow on the one hand to take into account aperiodic events which occur totally randomly, and on the other hand the dynamic creation of new operations during the execution of the algorithm. Because we intend more and more to take into account control [33], and because control is usually aperiodic, e.g. it is impossible to know when the car driver will turn on the starting key, and consequently when the corresponding event will trigger the transition from the state “stop” to “start”, we need to mix aperiodic on-line scheduling with periodic off-line scheduling.

Most works on scheduling which mix periodic and aperiodic events concern on-line approaches. For example, in the typical “aperiodic server” approach a new periodic operation is added which will be devoted to manage aperiodics operations. In the monoprocessor case, with RM (Rate Monotonic) or EDF (Earliest Deadline First) scheduling algorithms, it is simple to determine exactly the period and the duration of this new operation such that the new set of operations is schedulable, because there are well known schedulability bounds. However, in distributed systems we are interested in, such bounds do not exist. Lehoczky and Ramos-Thuel propose an alternative to aperiodic server called “slack stealing” [62] which relies on the knowledge of the slack of each operation, that is to say how many times an operation may be delayed without causing a missed deadline. The drawback of this approach is its overhead. Nevertheless off-line scheduling, we are interested in, has some advantages for aperiodic scheduling: every execution date is known, every CPU inactivity too. Fohler proposes “slot shifting” [59] which is close to slack stealing but uses off-line scheduling for periodic operations and in-line scheduling for aperiodic operations. The knowledge of the periodic scheduling eases to process, in-line, the slack of periodic operations. This strongly reduces the overhead. Slot shifting consists in splitting the scheduling in “execution intervals”. These intervals are delimited by the deadlines of operations, and of interprocessor communications. Then, the slack of each interval, called “spare capacity”, is processed. This approach is independent of the scheduling algorithm used to perform the off-line scheduling.

We aim at extending the slot shifting approach in order to deals with the specific features of our model, that is to say multiple latency constraints, stict periodicity and conditionning. A first slot shifting extension was proposed in the monoprocessor case for multiple latency constraints but with a unique period for every operation. This period must be equal or greater than the sum of all the operations durations. A latency constraint on a pair of operations imposes a maximum duration between the beginning of the first operation of the pair and the end of the second operation. Our extension consists in deducing deadlines from the latency constraints. But, such a deadline may move if the first operation of the latency constraint is delayed due to the execution of an aperiodic operation. This is the reason why the proposed extension allows to keep the spare capacities consistent when the deadlines move.

We also performed new experiments on the Cycab in order to tune the PID parameters of the “adaptive scheduling” algorithm that we proposed last year in order to take into account aperiodic operations mainly in the context of soft real-time only.

Finally, because mixing hard and soft real-time relates to support GALS (Globally Asynchronous Locally Synchronous) systems where synchronous (periodic) sub-systems are scheduled off-line, but these sub-systems asynchronously communicate through aperiodic events and are globally scheduled on-line, we started to establish links with the community of GALS in order to better understand these relationships.

6.7.3. Memory reuse

Because the amount of memory is usually limited in embedded processors we aim at minimizing memory utilization. Presently the RAMs necessary for storing local data memory (inside the processor) and for storing data to transfer (between processors) are specified in the implementation model by as many `alloc` vertices as there are of data dependences. A buffer is associated to each `alloc` vertex during the code generation. We studied the possibility to reuse a buffer as soon as it is not used latter on. We state this problem with colored graphs and proposed an algorithm which allows to reuse the buffers.

6.8. Automatic code generation

Participants: Julien Forget, Thierry Grandpierre, Linda Kaouane, Yves Sorel.

Again because the amount of memory is limited in embedded processors we started to study how to calculate the amount of program memory necessary when code is generated in order to compare it to the actual program memory of the processors.

Concerning the executive kernels, we worked on two executive kernels for processors: one for the PowerPC G4 processor, used in a multiprocessor architecture from Mercury, and communicating through the Raceway crossbar (collaboration with MBDA), and one for the TMS320C55 digital signal processor, the Arm general purpose processor, and the Xilinx FPGA, all together used in the Omap multiprocessor architecture from Texas Instrument, communicating through a shared memory (collaboration with Thales in the P2I european project).

We worked also on an executive kernel for synthetizable VHDL which is compiled for Xilinx FPGA circuit [44]. This code generation was tested on image processing applications.

6.9. Fault tolerance

Participants: Hamoudi Kalla, Yves Sorel.

We proposed several heuristics which are extensions of the one used in AAA/SynDEx.

The first one tolerates a fixed number of arbitrary processors and links (point-to-point communication medium) failures. Because of the resource limitation in the embedded systems this heuristic implements a software solution. It is based on the redundancy of the operations (resp. data dependences) of the algorithm onto the processors (rep. links) of the architecture [52][43].

The second one tolerates a fixed number of arbitrary processors and buses (multipoint communication medium) failures. This heuristic implements also a software solution but is based on the active redundancy of the operations and the passive redundancy of the communications with the fragmentation in several packets of the transfered data on the buses.

The third one tolerates a fixed number of arbitrary processors and communication media (point-to-point or multipoint) failures. It is based on a quite different approach. This heuristic generates as much distributions and schedulings as there are of different architecture configuration corresponding to the possible failures. Then, all the distributions and schedulings are merged together to finally obtain a resulting distribution and a scheduling which tolerates all the faults [51]. This technique asks for carefully canceling duplicated informations.

Finally, we proposed a heuristic for generating reliable distributions and schedulings. The software redundancy is used to maximize the reliability of the distribution and scheduling taking into account two criteria: the minimization of the latency (execution duration of the distributed and scheduled algorithm onto the architecture) and the maximization of the reliability of the processors and the communication media.

6.10. Improvements in SynDEx

Participants: Julien Forget, Arnaud Rouanet, Yves Sorel.

Version 6 of SynDEx was the latest major release. It was completely redesigned, in OCaml instead of C++ which was used previously. It provides new features such as hierarchical modularity (essential for top-dow design of huge application), repetition constructs (equivalent to `For...Do...`) and conditionals (equivalent to `If...Then...Else...`). SynDEx-6.0 is available since April 2002. In 2004 we had a lively of bug-fixing activity, mainly due to the success met by this software environment, and the many reports by industrial users developing practical applications for innovative products [45]. We provide further details on the kind of applications conducted at MBDA and Mitsubishi Electric ITE in the contractual section below.

Besides bug fixes, the complexity of the applications carried through led to improvements both in the hierarchy flattening techniques and in the optimization heuristics, in order to provide results in reasonable time. For the most complex application at MBDA we succeeded in dividing the adequation matching time by an order of magnitude, providing the results in about half an hour, which is considered acceptable in an approach of “rapid prototyping”.

We had to improve the code generator part of SynDEx, mainly because the real-time distributed executives that were automatically generated did increase dramatically with the size of the application. In the context of ITEA P2I (se below), Thales telecommunication develops a JPEG2000 application onto a TI Omap architecture consisting of an ARM RISC processor and a TMS320C50 DSP coprocessor, with very little memory on chip. To decrease the memory required for generated code, we introduced a new type of *input/output* port; this allows to reuse the same buffer for an input and an output port of the same operation. While this type of port proved useful for image applications such as JPEG2000, it associates more complex synchronizations to this type of operation, but decreases the amount of memory needed. We are currently working on more general formulation of the reuse of buffers, which remains a hard problem due to the intricate synchronization modifications involved.

The software architecture of the OCaml SynDEx program was completely revised, to help with its maintenance and its further evolutions. Modules were modified to offer better separation and communication between them, through cleaner and saner APIs, using .mli files. Also, genericity was added whenever possible, e.g. on the graph module used for displaying the three types of graphs we have in SynDEx: algorithm, architecture, adequation. A maintenance guide was written for the next developers of SynDEx.

Some users wanted to use SynDEx without its GUI, for example to run the adequation heuristic inside a meta-heuristic. We propose now a “SynDEx toplevel” which allows to execute only the adequation or the code generator.

We installed the Bugzilla tool http://bugzilla.irisa.fr to manage more easily the bugs of SynDEx.

7. Contracts and Grants with Industry

7.1. ST MIcroelectronics

Participants: Julien Boucaron, Robert de Simone.

This collaboration takes place in a much larger setting of a focused support program agreement between this company and the PACA regional public authorities. We collaborate with the team of Marc Benveniste, from ST Smart Card division, to study abstraction/refinement of SoC specification in a way inspired from the B Method, but using synchronous languages concepts instead. Julien Boucaron PhD thesis is funded largely on this contract.

7.2. Texas Instruments

Participants: Julien Boucaron, Dumitru Potop, Robert de Simone.

This contract was initiated by Texas Instruments because of a need they felt to better understand the theory of *Latency-insensitive* systems, in its connexions with synchronous semantics and more generally GALS description models. So we are conducted an extensive bibliographic survey, checking all along the potential (good or bad) implications for efficient SoC modeling, according to our previous experience gathered on the field.

7.3. MBDA

Participants: Julien Forget, Arnaud Rouanet, Yves Sorel.

MBDA develops with AAA/SynDEx a new automatic guidance application involving an algorithm with more than 6000 operations executed at different periods, whereas the architecture is made of several PowerPC and ASICs all interconnected through a crossbar.

7.4. Mitsubishi Electric ITE

Participants: Julien Forget, Arnaud Rouanet, Yves Sorel.

Mitsubishi Electric ITE develops with AAA/SynDEx software radio applications which involve a lot of modes supported by the conditioning feature and in each mode a lot of signal processing algorithms, each algorithm uses processing repetition feature, whereas the architecture is composed of several TMS320C60 DSP (Digital Signal Processor).

7.5. Robosoft

Participants: Julien Forget, Arnaud Rouanet, Yves Sorel.

This company is the maker of the CyCab automatic vehicle. SynDEx is used as high-level CAD software for new CyCab applications, and also as code generator for CyCab multi MPC555 platforms.

8. Other Grants and Activities

8.1. Regional collaborations

8.1.1. CIM PACA

Participant: Robert de Simone.

This initiative is intended to favor the development of collaborations between local PACA industry and academia partners on the topics of microelectronic design. In this context we are planning to gather a pool of PhD students, with coordinated research programs built in conjunction between INRIA, I3S, and local industry partners Texas Instruments, Philips, ST Microelectronics, and Esterel Technologies, on a project named Spec2RTL. This group of students should be funded on a par basis between industry and local authorities.

8.1.2. Numatec Automotive (temporary name)

Participants: Robert de Simone, Yves Sorel.

We have been attending the first meetings of this newborn project, trying to set up collaborations with automotive manufacturers and suppliers. Prospects are still at a preliminary stage.

8.1.3. The JavaHASE project

Participant: Frederic Mallet.

HASE and JavaHASE [28] are systems developed at the University of Edinburgh to support, through simulation, the visualisation of activities taking place inside computer architectures as they execute programs. Both systems are based on the same ADL called EDL, the behavourial part are described in different languages but automatic translations are provided when required. The French part of this project consists in defining modeling abstractions that allow for giving precise semantics to component collaboration and generalization mechanisms. The proposed abstractions are UML-based and environment independent. JavaHASE will be used to implement these mechanisms. Some automatic model transformation tools will be provided to interoperate with other environment like SystemC.

8.2. Nation-wide collaborations

8.2.1. Relations with other INRIA teams

We have strong ties with INRIA teams ESPRESSO and DaRT through the PROTES initiative on synchronous and more generally RTE (Real-Time Embedded) modeling in UML. We conduct joint work with POP-ART on fault tolerance and adaptive scheduling for robotic applications. Together with the S4 team we regularly attend the same events gathering teh “Synchronous languages” community.

We also collaborate with IMARA on applications of SynDEx into automatic vehicles such as the CyCab, and with METALAU on coupling of Scilab/Scicos with AAA/SynDEx. Historical links are preserved with the team SOSSO, on adaptive scheduling for applications mixing soft and hard real-time.

8.2.2. CARROLL project PROTES

Participants: Charles André, Robert de Simone, Yves Sorel.

CARROLL is a joint initiative between Thales, CEA, and INRIA to launch collaborative projects, mostly on UML and/or MDA based topics. In this context we are trying in the PROTES project to promote a specific UML profile for Real-Time Embedded systems. This profile should borrow extensively from results of the ITEA P2I project (see below), and promote rigorous synchronous behaviors semantics, as well as high-level architecture modeling of computer-aided application mapping. We benefit here from the experience gathered by Thales in the governing policies of the OMG, and the procedural steps required for acceptance of our “RFP” (Request-For-Proposal) document. This should be done by january 2005.

8.2.3. RNTL project ECLIPSE

Participant: Yves Sorel.

The goal of Eclipse is to providea seamless environment from specification/modeling/simulation with Scilab/Scicos to optimized implementation with AAA/SynDEx. It was started in 2003 for a duration of two years and a half. The partners are: PSA, CS-SI, CRIL, ESIEE, INRIA.

8.3. European collaborations

8.3.1. ITEA project EAST-EEA

Participants: Charles André, Yves Sorel.

This consortium aims at improving the management of complex electronic functions in the automotive domain, with an open architecture based on interoperable software and hardware components. It started beginning of 2002, with a three years duration. Partners are: PSA, RENAULT, AUDI, BMW, Daimler-Chrysler, FIAT, OPEL, VOLVO, BOSCH, Magneti-Marelli, SIEMENS, ZF, ETAS, VECTOR, Paderborn University, Linkoping University, Malardalen University, Technical University of Darmstadt, IrCCyn Nantes, LORIA Nancy and INRIA.labla

AOSTE actively contributes to the definition of the EAST-EEA language for automotive embedded electronic architecture description, its metamodel and the code generation model.

8.3.2. ITEA project PROMPT2IMPLEMENTATION (P2I)

Participants: Julien Forget, Fabrice Peix, Robert de Simone, Yves Sorel.

This projet gathered in its consortium the following partners: Thales Telecommunication, Nokia, Esterel Technology, Tampere University, Turku University, LIFL Lille, INRIA. Its goal is to establish a specific UML profile for Real-Time Embedded systems, designed as to: 1) allow the modeling of synchronous applications in a way borrowed from SCADE and SyncCharts formalisms; 2) allow the abstract modeling of architectural platforms and non-functional performance features, in a way fit to apply AAA optimized mapping methods with the SynDEx tools. An extensive case study advocating the design flow was realized, including JPEG2k coding/decoding functions, and wireless transmission. The results were presented (and apparently appraised) at the ITEA global symposium in Sevilla in october 2004.

8.3.3. IST Network of Excellence ARTIST

Participants: Robert de Simone, Yves Sorel.

Our participation here consists essentially (as for many other partners) in attending working group presentation meetings (without real collaborative work so far). We follow particularly the work of Working Group 1 on Hard Real-Time, with focus on Synchronous languages, Time-Triggered architectures and fixed priority scheduling.

8.4. Research exchange visits

We obtained in the end of 2004 a grant under the INRIA Associated Team exchange program, together with the research group of Stephen Edwards at Columbia University (New York).

9. Dissemination

9.1. Leadership within scientific community

Robert de Simone was editor of the TSI French journal, and is now on the International Advisory Board for the CRC Press on embedded systems. He was Program Member for Memocode'04 and MSR'04. He is also member of the *Commission de Spécialiste UNSA 27^{e} section*, and INRIA representative to the CIM PACA regional initiative on Microelectrnics design. In his role as INRIA leader of the CARROLL PROTES project he attended several OMG Technical Meetings at various locations in the US.

Yves Sorel leads the Theme C Working Group (Adequation Algorithme Architecture) of the PRC-GDR ISIS (Information Signal Images et viSion). He was Program Member for the following conferences and workshops: JFAAA, ERTS, EUSIPCO, GRESTSI, JEAI, SYMPA, RTS.

Frederic Mallet was responsible for a session on Java and object-oriented technologies for Electrical Engineering at the annual Summer School of GEII in Montlucon. He was selected as an expert on UML technologies for producing result assessments on two TEMPUS-TACIS projects involving the University of Nice, the Glasgow Caledonian University and three Ukrainian Universities.

9.2. Teaching

Charles André is a Professor at the University of Nice-Sophia Antipolis, department of Electrical Engineering. He teaches sequential circuits, discrete event systems, computer architecture and real-time programming. He also teaches "synchronous programming" and "UML for systems" in the university's engineer schools (ESINSA: electrical engineering and ESSI: computer science) and in the STIC research master.

Robert de Simone taught courses on Formal Methods and Models for embedded systems, at UNSA in the STIC research master and the Math-info DEA, as well as at ISIA (an engineering school located in Sophia-Antipolis).

Yves Sorel teaches at ESIEE (an engineering school located in Noisy-le-Grand), in the Research Master cursus at the University of Orsay Paris 11, and at ENSTA (an engineering school located in Paris), on topics comprising the AAA methodology, formal modeling and specification of distributed embedded systems.

Both Eric Vecchié and Fabrice Peix hoed part-time ATER positions at UNSA during the final duration of their PhD period. Eric Vecchié taught lab classes in programming languages (Java/C/Scheme) and Unix to computer science major students, for a global duration of 92 hours. Fabrice Peix taught classes at IUT for the same time amount.

Frederic Mallet teaches object-orientation and UML at the master Level of ESINSA (45h), javacard technology (24h) in the Master of Telecommunications in Sophia Antipolis. For undergraduate students, he teaches hardware architecture foundations (42h) in the Informatics Department in Nice and gives an introduction to Embedded Java (9h).

10. Bibliography

Major publications by the team in recent years

[1] C. ANDRÉ. *Semantics of SSM (Safe State Machine)*, Esterel Technologies, April 2003, http://www.esterel-technologies.com.

[2] C. ANDRÉ, F. BOULANGER, A. GIRAULT. *Software Implementation of Synchronous Programs*, in "Proceedings of the Second International Conference on Application of Concurrency to System Design, Newcastle upon Tyne, UK, June 25–29, 2001", IEEE Computer Society Press Order Number PR01071 Library of Congress Number 2001090878 ISBN 0-7695-1071-X, IEEE Computer Society, 2001, p. 133-142.

[3] C. ANDRÉ, M.-A. PERALDI-FRATI, J.-P. RIGAULT. *Integrating the Synchronous Paradigm into UML: Application to Control-Dominated Systems*, in "UML ≪2002≫, Dresden (D)", Springer-Verlag, October 2002, p. 163–178.

[4] C. ANDRÉ, J.-P. RIGAULT. *Variations on the Semantics of Graphical Models for Reactive Systems*, in "SMC'02, ISBN: 2-9512309-4-x, Hammamet (TN)", On CD-ROM, index: TA2L2, IEEE Press, October 2002.

[5] C. ANDRÉ, R. DE SIMONE. *Synchronous Programming : Properties within a Reaction*, in "JESA", vol. 36, n° 7, 2002, p. 891–903.

[6] C. ANDRÉ, R. DE SIMONE. *Towards a Synchronous UML Profile ?*, in "Proceedings of the first Workshop on Specification and Validation of UML models for Real-Time and Embedded Systems (SVERTS'03)", Electronically available, 2003.

[7] C. ANDRÉ. *Representation and Analysis of Reactive Behavior: a Synchronous Approach*, in "Computational Engineering in Systems Applications (CESA)", IEEE-SMC, 1996, p. 19–29.

[8] C. ANDRÉ, F. BOULANGER, M.-A. PERALDI, J.-P. RIGAULT, G. VIDAL-NAQUET. *Objects and Synchronous Programming*, in "RAIRO-APII-JESA", vol. 31, n° 3, 1997.

[9] C. ANDRÉ, M.-A. PERALDI-FRATI, J.-P. RIGAULT. *Scenario and Property Checking of Real-Time Systems Using a Synchronous Approach*, in "4th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001), Magdeburg, Germany", IEEE, May 2001.

[10] A. BENVENISTE, G. BERRY. *The Synchronous Approach to Reactive and Real-Time Systems*, in "Proceedings of the IEEE", vol. 79, n° 9, September 1991, p. 1270-1282.

[11] A. BENVENISTE, P. CASPI, S. EDWARDS, N. HALBWACHS, P. L. GUERNIC, R. DE SIMONE. *Synchronous Languages Twelve Years Later*, in "Proceedings of the IEEE", January 2003.

[12] G. BERRY. *The Foundations of Esterel*, Foundations of Computing Series, MIT Press, 2000, http://www-sop.inria.fr/esterel.org/Html/Downloads/Doc/StartDocument.htm.

[13] G. BERRY. *The Constructive Semantics of Pure Esterel*, electronic version only, 1999, http://www-sop.inria.fr/esterel.org/Html/Downloads/Doc/StartDocument.htm.

[14] G. BERRY. *The Esterel Language Primer*, version electronique, 1999, http://www-sop.inria.fr/esterel.org/Html/Downloads/Doc/Sta

[15] F. BOUSSINOT, R. DE SIMONE. *The Esterel Language*, in "Proceedings of the IEEE", September 1991.

[16] L. CUCU, R. KOCIK, Y. SOREL. *Real-time scheduling for systems with precedence, periodicity and latency constraints*, in "Proceedings of 10th Real-Time Systems Conference, RTS'02, Paris, France", March 2002.

[17] L. CUCU, Y. SOREL. *Schedulability condition for systems with precedence and periodicity constrainsts without preemption*, in "Proceedings of 11th Real-Time Systems Conference, RTS'03, Paris", March 2003.

[18] A. DIAS, C. LAVARENNE, M. AKIL, Y. SOREL. *Optimized Implementation of Real-Time Image Processing Algorithms on Field Programmable Gate Arrays*, in "Proceedings of Fourth International Conference on Signal Processing, ICSP'98, Beijing, China", October 1998.

[19] S. F. MALLET R.N. IBBETT. *An Extensible Clock mechanism for Computer Architecture Simulations*, in "Proceedings of the 13th International Conference on Modeling and Simulation, los angeles", may 2002.

[20] A. GIRAULT, H. KALLA, M. SIGHIREANU, Y. SOREL. *An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules*, in "Proceedings of International Conference on Dependable Systems and Networks, DSN'03, San Francisco, California, USA", June 2003.

[21] A. GIRAULT, C. LAVARENNE, M. SIGHIREANU, Y. SOREL. *Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems*, Rapport de recherche, n^o 4006, INRIA, septembre 2000, http://www.inria.fr/rrrt/rr-4006.html.

[22] A. GIRAULT, C. LAVARENNE, M. SIGHIREANU, Y. SOREL. *Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems*, in "Proceedings of 21st International Conference on Distributed Computing Systems, ICDCS'01, Phoenix, USA", April 2001.

[23] T. GRANDPIERRE. *Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés*, Ph. D. Thesis, Université de Paris Sud, Spécialité électronique, 30/11/2000.

[24] T. GRANDPIERRE, C. LAVARENNE, Y. SOREL. *Modèle d'exécutif distribué temps réel pour SynDEx*, Rapport de Recherche, n^o 3476, INRIA, August 1998, http://www.inria.fr/rrrt/rr-3476.html.

[25] T. GRANDPIERRE, C. LAVARENNE, Y. SOREL. *Optimized Rapid Prototyping For Real-Time Embedded Heterogeneous multiprocessors*, in "Proceedings of 7th International Workshop on Hardware/Software Co-Design, CODES'99, Rome, Italy", May 1999.

[26] T. GRANDPIERRE, Y. SOREL. *From Algorithm and Architecture Specification to Automatic Generation of Distributed Real-Time Executives: a Seamless Flow of Graphs Transformations*, in "Proceedings of First ACM and IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE'03, Mont Saint-Michel, France", June 2003.

[27] N. HALBWACHS. *Synchronous Programming of Reactive Systems*, in "Computer Aided Verification", 1998, p. 1-16, http://citeseer.ist.psu.edu/10686.html.

[28] F. M. R. IBBETT. *JavaHase: Automatic Generation of Applets from HASE Simulation Models*, in "Proceedings of the 2003 Summer Computer Simulation Conference, montreal", august 2003.

[29] L. KAOUANE, M. AKIL, Y. SOREL, T. GRANDPIERRE. *From algorithm graph specification to automatic synthesis of FPGA circuit: a seamless flow of graph transformations*, in "Proceedings of 13th international conference on Field-Programmable Logic and Applications, FPL'03, Lisbon, Portugal", September 2003.

[30] C. LAVARENNE, O. SEGHROUCHNI, Y. SOREL, M. SORINE. *The SynDEx Software Environment for Real-Time Distributed Systems, Design and Implementation*, in "Proceedings of European Control Conference, ECC'91, Grenoble, France", July 1991.

[31] C. LAVARENNE, Y. SOREL. *Performance Optimization of Multiprocessor Real-Time Applications by Graph Transformations*, in "Proceedings of Parallel Computing Conference, PARCO'93, Grenoble, France", September 1993.

[32] C. LAVARENNE, Y. SOREL. *Modèle unifié pour la conception conjointe logiciel-matériel*, in "Traitement du Signal", vol. 14, n^{o} 6, 1997.

[33] N. PERNET, Y. SOREL. *From Specification to Optimized Implementation of Distributed Real-Time Embedded Systems Mixing Control and Data Processing*, in "Proceedings of ISCA 16th International Conference: Computer Applications in Industry and Engineering, CAINE'03, Las Vegas Nv", November 2003.

[34] F. M. S. ALAM. *Performance Evaluation of Local Communications: A Case-study*, in "Proceedings of the 15th International Conference on Parallel Distributed Computings and Systems, CA,USA", november 2003.

[35] Y. SOREL. *Massively Parallel Systems with Real Time Constraints, the Algorithm Architecture Adequation Methodology*, in "Proceedings of Conference on Massively Parallel Computing Systems, MPCS'94, Ischia, Italy", May 1994.

[36] Y. SOREL. *Real-Time Embedded Image Processing Applications using the AAA Methodology*, in "Proceedings of IEEE International Conference on Image Processing, ICIP'96, Lausanne, Switzerland", September 1996.

[37] A. VICARD, Y. SOREL. *Formalization and Static Optimization for parallel implementations*, in "Proceedings of Workshop on Distributed and Parallel Systems, DAPSYS'98, Budapest, Hungary", September 1998.

[38] A. VICARD. *Formalisation et optimisation des systèmes informatiques distribués temps réel embarqués*, Ph. D. Thesis, Université de Paris Nord, Spécialité informatique, 5/07/1999.

Doctoral dissertations and Habilitation theses

[39] L. CUCU. *Ordonnancement non préemptif et condition d'ordonnançabilité pour système embarqués à contraintes temps réel*, Ph. D. Thesis, Université de Paris Sud, Spécialité électronique, 28/05/2004.

[40] F. PEIX. *Distribution de programmes synchrones : Le cas d'Esterel*, Ph. D. Thesis, Université de Nice-Sophia Antipolis, Juillet 2004.

[41] O. TARDIEU. *De la sémantique opérationnelle à la spécification formelle de compilateurs : l'exemple des boucles en Esterel*, Ph. D. Thesis, Ecole des Mines de Paris, 2004.

[42] E. VECCHIÉ. *Calcul des états atteignables de programmes Esterel partitionné se lon la syntaxe*, Ph. D. Thesis, Université de Nice – Sophia Antipolis, July 2004.

Articles in referred journals and book chapters

[43] A. GIRAULT, H. KALLA, Y. SOREL. *A Scheduling Heuristics for Distributed Real-Time Embedded Systems Tolerant to Processor and Communication Media Failures*, in "International Journal of Production Research", vol. 42, n^{o} 14, July 2004, p. 2877–2898.

[44] L. KAOUANE, M. AKIL, T. GRANDPIERRE, Y. SOREL. *A methodology to implement real-time applications onto reconfigurable circuits*, in "Journal of Supercomputing", vol. 30, n^{o} 3, December 2004, p. 362-376.

[45] Y. SOREL. *SynDEx: System-Level CAD Software for Optimizing Distributed Real-Time Embedded Systems*, in "Journal ERCIM News", vol. 59, October 2004, p. 68-69.

[46] E. VECCHIÉ, R. DE SIMONE. *Syntax-driven optimizations for Reachable State Space construction of E sterel programs*, in "International Journal of Embedded Systems – Special Issue on Design and Verification of Real-Time Embedded Software", April 2005, http://www.cs.ccu.edu.tw/~pahsiung/ijes-rtes/.

[47] R. DE SIMONE, D. POTOP, J.-P. TALPIN. *The Synchronous Hypothesis and Synchronous Languages*, chapter of the Embedded Systems Handbook, CRC Press, 2005.

[48] R. DE SIMONE, O. TARDIEU. *Loops in Esterel*, accepted for publication in Transactions on Embedded Computing Systems, 2005.

Publications in Conferences and Workshops

[49] L. CUCU, Y. SOREL. *Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints*, in "Proceedings of the 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PlanSIG'04", December 2004.

[50] L. CUCU, Y. SOREL. *Ordonnancement non-préemptif pour systèmes temps réel à contraintes de précédences et de latences*, in "Actes de la Conférence internationale en Recherche Opérationnelle, FRANCORO IV, Fribourg, Suisse", august 2004.

[51] C. DIMA, A. GIRAULT, Y. SOREL. *Static fault-tolerant real-time scheduling with "pseudo-topological" orders*, in "Proceedings of Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System, FORMATS-FTRTFT'04, Grenoble, France", LNCS, Springer-Verlag, September 2004.

[52] A. GIRAULT, H. KALLA, Y. SOREL. *An Active Replication Scheme that Tolerates Failures in Distributed Embedded Real-Time Systems*, in "Proceedings of IFIP Working Conference on Distributed and Parallel Embedded Systems, DIPES'04, Toulouse, France", Kluwer Academic, August 2004.

[53] O. TARDIEU, R. DE SIMONE. *Curing Schizophrenia by Program Rewriting in Esterel*, in "Proc. MEM-OCODE'04", IEEE Press, 2004.

[54] O. TARDIEU. *A Deterministic Logical Semantics for Esterel*, in "Proc. SOS Workshop", Electronic Notes in Theoretical Computer Science, Elsevier, 2004.

[55] O. TARDIEU. *Goto and Concurrency: Introducing Safe Jumps in Esterel*, in "Proc. SLAP'04", Electronic Notes in Theoretical Computer Science, Elsevier, 2004.

[56] E. VECCHIÉ, R. DE SIMONE. *Syntax-driven behavior partitioning for model-checking of Esterel progr ams*, in "Proceedings of the Third International Workshop on Synchronous Language s, Applications, and Programs (SLAP'04), Barcelona", March 2004, http://www.inrialpes.fr/pop-art/people/girault/Slap04/.

Bibliography in notes

[57] R. BALAKRISNAN, K. RANGANATHAN. *A Textbook of Graph Theory*, Springer, 2000.

[58] J. DENNIS. *First Version of a Dataflow Procedure Language*, in "Lecture Notes in Computer Sci.", vol. 19, Springer-Verlag, 1975, p. 362-376.

[59] G. FOHLER. *Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems*, in "Procedings of IEEE Real-Time Systems Symposium", 1995, p. 152-161.

[60] N. HALBWACHS. *Synchronous programming of reactive systems*, Kluwer Academic Publishers, Dordrecht Boston, 1993.

[61] E. LEE, M. D.G.. *Synchronous Data Flow*, in "Proceedings of the IEEE", 1987, vol. 75, no. 9.

[62] J. LEHOCZKY, S. RAMOS-THUEL. *Scheduling periodic and aperiodic tasks using the slack stealing algorithm*, Advances in Real-Time Systems, chap. 8, Prentice-Hall, 1995, p. 175–198.

[63] J. LEUNG, W. J.. *On the complexity of fixed-priority scheduling of periodic real-time tasks*, in "Performance Evaluation(4)", 1982.

[64] Z. LIU, C. CORROYER. *Effectiveness of heuristics and simulated annealing for the scheduling of concurrent task. An empirical comparison*, in "PARLE'93, 5th international PARLE conference, June 14-17, Munich, Germany", November 1993, p. 452-463.

[65] C. LIU, J. LAYLAND. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, in "Journal of the ACM", 1973.

[66] C. MEAD, L. CONWAY. *Introduction to VLSI systems*, Addison-Wesley, 1980.

[67] V. PRATT. *Modeling concurrency with partial orders*, in "International Journal of Parallel Programming", vol. 15, n^{o} 1, 1986.

[68] A. ZOMAYA. *Parallel and distributed computing handbook*, McGraw-Hill, 1996.