Activity Report 2011

Project-Team VERTECS

Verification models and techniques applied to testing and control of reactive systems

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Embedded and Real Time Systems

Table of contents

1. Members 1
2. Overall Objectives 1
2.1. Introduction 1
2.2. Highlights 2
3. Scientific Foundations 2
3.1. Underlying models 2
3.2. Verification 3
3.2.1. Abstract interpretation and data handling 4
3.2.2. Model-checking quantitative systems 4
3.2.3. Analysis of infinite state systems defined by graph grammars 5
3.3. Automatic test generation 5
3.4. Control synthesis 6
4. Application Domains 7
4.1. Overview 7
4.2. Telecommunication systems 8
4.3. Software embedded systems 8
4.4. Control-command systems 8
5. Software 8
5.1. TGV 8
5.2. STG 9
5.3. SIGALI 9
6. New Results 9
6.1. Verification 9
6.1.1. Analysis of partially observed recursive discrete-event systems 9
6.1.2. Analysis of timed systems 10
6.1.2.1. Approximate determinization of timed automata 10
6.1.2.2. Frequency analysis for timed automata 10
6.1.3. Petri nets reachability graphs 10
6.2. Active and passive testing 10
6.2.1. Off-line test selection with test purposes for non-deterministic timed automata 10
6.2.2. Test generation using pushdown automata 10
6.2.3. Test case selection in asynchronous testing 11
6.2.4. A tagging protocol for asynchronous testing 11
6.2.5. Abstracting time and data for conformance testing of real-time systems 11
6.2.6. Ensuring security properties 11
6.2.6.1. Runtime enforcement monitors: composition, synthesis, and enforcement abilities 11
6.2.6.2. What can you verify and enforce at runtime? 12
6.3. Control synthesis 12
6.3.1. Controllers for probabilistic systems 12
6.3.2. Supervisory control for synchronous systems 12
6.3.2.1. Controller synthesis and programming language 12
6.3.2.2. Symbolic supervisory control of infinite transition systems under partial observation using abstract interpretation 12
6.3.2.3. Decentralized control of infinite systems 13
6.3.2.4. Polychronous controller synthesis from MARTE CCSL timing specifications 13
6.3.3. Control of distributed systems 13
7. Partnerships and Cooperations 14
7.1. National initiatives 14
7.1.1. ANR TesTec: Test of real-time and critical embedded system 14

7.1.2. ANR VACSIM: Validation of critical control-command systems by coupling simulation and formal analysis 14
7.1.3. Action Incitative VeSPa: Verification of security and privacy properties 14
7.2. European initiatives 14
7.2.1. Artist design network of excellence 14
7.2.2. PHC Tournesol STP : Verification of timed and probabilistic systems 15
7.2.3. Followed collaborations with major European organizations 15
7.3. International Initiatives 15
7.3.1. INRIA associate team 15
7.3.2. INRIA international partners 16
7.3.3. Visits of international scientists 16
8. Dissemination **16**
8.1. Animation of the scientific community 16
8.2. Teaching and supervision 16
8.2.1. Teaching 16
8.2.2. PhD 17
9. Bibliography **17**

Project-Team VERTECS

Keywords: Formal Methods, Verification, Model-Checking, Program Testing, Control, Embedded Systems

1. Members

Research Scientists

Thierry Jéron [Team Leader, Senior Researcher, INRIA, HdR]
Nathalie Bertrand [Junior Researcher, INRIA, on sabbatical at Liverpool University since November 2011]
Hervé Marchand [Junior Researcher, INRIA]

PhD Students

Sébastien Chédor [UNIVERSITÉ DE RENNES 1, since September 2009]
Srinivas Pinisetty [INRIA, since December 2011]
Amélie Stainer [UNIVERSITÉ DE RENNES 1, since October 2010]

Post-Doctoral Fellows

Puneet Bhateja [INRIA, April 2010 to May 2011]
Yliès Falcone [INRIA, December 2009 to July 2011]

Visiting Scientists

Christophe Morvan [Assistant Professor, Univ. de Marne-la-Vallée]
Laurie Ricker [Associate Professor, Mount Allison University, January 2011 to June 2011]

Administrative Assistant

Lydie Mabil [TR INRIA, (80%)]

2. Overall Objectives

2.1. Introduction

The VerTeCs team is focused on the use of formal methods to assess the reliability, safety and security of reactive software systems. By reactive software system we mean a system controlled by software which reacts with its environment (human or other reactive software). Among these, critical systems are of primary importance, as errors occurring during their execution may have dramatic economical or human consequences. Thus, it is essential to establish their correctness before they are deployed in a real environment, or at least detect incorrectness during execution and take appropriate action. For this aim, the VerTeCs team promotes the use of formal methods, i.e. formal specification of software and their required properties and mathematically founded validation methods. Our research covers several validation methods, all oriented towards a better reliability of software systems:

- Verification, which is used during the analysis and design phases, and whose aim is to establish the correctness of specifications with respect to requirements, properties or higher level specifications.
- Control synthesis, which consists in "forcing" (specifications of) systems to stay within desired behaviours by coupling them with a supervisor.
- Conformance testing, which is used to check the correctness of a real system with respect to its specification. In this context, we are interested in model-based testing, and in particular automatic test generation of test cases from specifications.
- Diagnosis and monitoring, which are used during execution to detect erroneous behaviour.
- Combinations of these techniques, both at the methodological level (combining several techniques within formal validation methodologies) and at the technical level (as the same set of formal verification techniques - model checking, theorem proving and abstract interpretation - are required for control synthesis, test generation and diagnosis).

Our research is thus concerned with the development of formal models for the description of software systems, the formalization of relations between software artifacts (e.g. satisfaction, conformance between properties, specifications, implementations), the interaction between these artifacts (modelling of execution, composition, etc). We develop methods and algorithms for verification, controller synthesis, test generation and diagnosis that ensure desirable properties (e.g. correctness, completeness, optimality, etc). We try to be as generic as possible in terms of models and techniques in order to cope with a wide range of application domains and specification languages. Our research has been applied to telecommunication systems, embedded systems, smart-cards application, and control-command systems. We implement prototype tools for distribution in the academic world, or for transfer to the industry.

Our research is based on formal models and our basic tools are **verification** techniques such as model checking, abstract interpretation, the control theory of discrete event systems, and their underlying models and logics. The close connection between testing, control and verification produces a synergy between these research topics and allows us to share theories, models, algorithms and tools.

2.2. Highlights

FoSSaCS paper [18] and TACAS paper [17] seriously improve the state of the art and may have a strong impact. [18] proposes an approximate determinization procedure for timed automata, successfuly adapted in [17] for off-line test generation from timed automata, and is promising for other observability problems (diagnosis, implementability,...).

3. Scientific Foundations

3.1. Underlying models

The formal models we use are mainly automata-like structures such as labelled transition systems (LTS) and some of their extensions: an LTS is a tuple $M = (Q, \Lambda, \rightarrow, q_o)$ where Q is a non-empty set of states; $q_o \in Q$ is the initial state; A is the alphabet of actions, $\rightarrow \subseteq Q \times \Lambda \times Q$ is the transition relation. These models are adapted for testing and controller synthesis.

To model reactive systems in the testing context, we use Input/Output labeled transition systems (IOLTS for short). In this setting, the interactions between the system and its environment (where the tester lies) must be partitioned into inputs (controlled by the environment), outputs (observed by the environment), and internal (non observable) events modeling the internal behavior of the system. The alphabet Λ is then partitioned into $\Lambda_! \cup \Lambda_? \cup \mathcal{T}$ where $\Lambda_!$ is the alphabet of outputs, $\Lambda_?$ the alphabet of inputs, and $\mathcal{T}$ the alphabet of internal actions.

In the controller synthesis theory, we also distinguish between controllable and uncontrollable events ($\Lambda = \Lambda_c \cup \Lambda_{uc}$), observable and unobservable events ($\Lambda = \Lambda_O \cup \mathcal{T}$).

In the context of verification, we also use Timed Automata. A timed automaton is a tuple $A = (L, X, E, \mathcal{I})$ where L is a set of locations, X is a set of clocks whose valuations are positive real numbers, $E \subseteq L \times \mathcal{G}(\mathcal{X}) \times 2^X \times L$ is a finite set of edges composed of a source and a target state, a guard given by a finite conjunction of expressions of the form $x \sim c$ where x is a clock, c is a natural number and $\sim \in \{<, \leq, =, \geq, >\}$, a set of resetting clocks, and $\mathcal{I} : \mathcal{L} \rightarrow \mathcal{G}(\mathcal{X})$ assigns an invariant to each location [27]. The semantics of a timed automaton is given by a (infinite states) labelled transition system whose states are composed of a location and a valuation of clocks.

Also, for verification purposes, we use graph grammars that are a general tool to define families of graphs. Such grammars are formed by a set of rules, left-hand sides being simply hyperedges and right-hand sides hypergraphs. For finite degree, these graph grammars characterise transition graphs of pushdown automata (each graph generated by such a grammar corresponds to the transition graph of a pushdown automaton). They provide a simple yet powerfull setting to define and study infinite state systems.

In order to cope with more realistic models, closer to real specification languages, we also need higher level models that consider both control and data aspects. We defined (input-output) symbolic transition systems ((IO)STS), which are extensions of (IO)LTS that operate on data (i.e., program variables, communication parameters, symbolic constants) through message passing, guards, and assignments. Formally, an IOSTS is a tuple (V, Θ, Σ, T), where V is a set of variables (including a counter variable encoding the control structure), Θ is the initial condition defined by a predicate on V, Σ is the finite alphabet of actions, where each action has a signature (just like in IOLTS, Σ can be partitioned as e.g. $\Sigma_? \cup \Sigma_! \cup \Sigma_\tau$), T is a finite set of symbolic transitions of the form $t = (a, p, G, A)$ where a is an action (possibly with a polarity reflecting its input/output/internal nature), p is a tuple of communication parameters, G is a guard defined by a predicate on p and V, and A is an assignment of variables. The semantics of IOSTS is defined in terms of (IO)LTS where states are vectors of values of variables, and transitions between them are labelled with instantiated actions (action with valued communication parameter). This (IO)LTS semantics allows us to perform syntactical transformations at the (IO)STS level while ensuring semantical properties at the (IO)LTS level. We also consider extensions of these models with added features such as recursion, fifo channels, etc. An alternative to IOSTS to specify systems with data variables is the model of synchronous dataflow equations.
Our research is based on well established theories: conformance testing, supervisory control, abstract interpretation, and theorem proving. Most of the algorithms that we employ take their origins in these theories:

- graph traversal algorithms (breadth first, depth first, strongly connected components, ...). We use these algorithms for verification as well as test generation and control synthesis.
- BDDs (Binary Decision Diagrams) algorithms, for manipulating Boolean formula, and their MTB-DDs (Multi-Terminal Decision Diagrams) extension for manipulating more general functions. We use these algorithms for verification, test generation and control.
- abstract interpretation algorithms, specifically in the abstract domain of convex polyhedra (for example, Chernikova's algorithm for the computation of dual forms). Such algorithms are used in verification and test generation.
- logical decision algorithms, such as satisfiability of formulas in Presburger arithmetics. We use these algorithms during generation and execution of symbolic test cases.

3.2. Verification

Verification in its full generality consists in checking that a system, which is specified by a formal model, satisfies a required property. Verification takes place in our research in two ways: on the one hand, a large part of our work, and in particular controller synthesis and conformance testing, relies on the ability to solve some verification problems. Many of these problems reduce to reachability and coreachability questions on a formal model (a state s is *reachable from an initial state* s_i if an execution starting from s_i can lead to s; s is *coreachable from a final state* s_f if an execution starting from s can lead to s_f). These are important cases of verification problems, as they correspond to the verification of safety properties.

On the other hand we investigate verification on its own in the context of complex systems. For expressivity purposes, it is necessary to be able to describe faithfully and to deal with complex systems. Some particular aspects require the use of infinite state models. For example asynchronous communications with unknown transfer delay (and thus arbitrary large number of messages in transit) are correctly modeled by unbounded FIFO queues, and real time systems require the use of continuous variables which evolve with time. Apart from these aspects requiring infinite state data structure, systems often include uncertain or random behaviours (such as failures, actions from the environment), which it make sense to model through probabilities. To encompass these aspects, we are interested in the verification of systems equipped with infinite data structures and/or probabilistic features.

When the state space of the system is infinite, or when we try to evaluate performances, standard model-checking techniques (essentially graph algorithms) are not sufficient. For large or infinite state spaces, symbolic model-checking or approximation techniques are used. Symbolic verification is based on efficient representations of sets of states and permits exact model-checking of some well-formed infinite-state systems. However, for feasibility reasons, it is often mandatory to use approximate computations, either by computing a finite abstraction and resort to graph algorithms, or preferably by using more sophisticated abstract interpretation techniques. For systems with stochastic aspects, a quantitative analysis has to be performed, in order to evaluate the performances. Here again, either symbolic techniques (e.g. by grouping states with similar behaviour) or approximation techniques should be used.

We detail below verification topics we are interested in: abstract interpretation, quantitative model-checking and analysis of systems defined by graph grammars.

3.2.1. Abstract interpretation and data handling

Most problems in test generation or controller synthesis reduce to state reachability and state coreachability problems which can be solved by fixpoint computations of the form $x = F(x), x \in C$ where C is a lattice. In the case of reachability analysis, if we denote by S the state space of the considered program, C is the lattice $\wp(S)$ of sets of states, ordered by inclusion, and F is roughly the "*successor states*" function defined by the program.

The big change induced by taking into account the data and not only the (finite) control of the systems under study is that the fixpoints become uncomputable. The undecidability is overcome by resorting to approximations, using the theoretical framework of Abstract Interpretation [29]. The fundamental principles of Abstract Interpretation are:

1. to substitute to the *concrete domain* C a simpler *abstract domain* A (static approximation) and to transpose the fixpoint equation into the abstract domain, so that one has to solve an equation $y = G(y), y \in A$;
2. to use a *widening operator* (dynamic approximation) to make the iterative computation of the least fixpoint of G converge after a finite number of steps to some upper-approximation (more precisely, a post-fixpoint).

Approximations are conservative so that the obtained result is an upper-approximation of the exact result. In simple cases the state space that should be abstracted has a simple structure, but this may be more complicated when variables belong to different data types (Booleans, numerics, arrays) and when it is necessary to establish *relations* between the values of different types.

3.2.2. Model-checking quantitative systems

Model-checking techniques for finite-state systems are now quite developed, and a current challenge is to adapt them as much as possible to infinite-state systems. We detail below two types of models we are interested in: timed automata and infinite-state probabilistic systems.

Model-checking timed automata The model of timed automata, introduced by Alur and Dill in the 90's [27] is commonly used to represent real-time systems. Timed automata consist of an extension of finite automata with continuous variables, called clocks, that evolve synchronously with time, and can be tested and reset along an execution. Despite their uncountable state space, checking reachability, and more generally ω-regular properties, is decidable *via* the construction of a finite abstraction, the so-called region automaton. The recent developments in model-checking timed automata have aimed at modelling and verifying quantitative aspects encompassing timing constraints, for example costs, probabilities, frequencies. These quantitative questions demand advanced techniques that go far beyond the classical methods.

Model-checking infinite state probabilistic systems Model-checking techniques for finite state probabilistic systems are now quite developed. Given a finite state Markov chain, for example, one can check whether some property holds almost surely (i.e. the set of executions violating the property is negligible), and one can even compute (or at leat approximate as close as wanted) the probability that some property holds. In general, these techniques cannot be adapted to infinite state probabilistic systems, just as model-checking algorithms for finite state systems do not carry over to infinite state systems. For systems exhibiting complex data structures (such as unbounded queues, continuous clocks) and uncertainty modeled by probabilities, it can thus be hard to design model-checking algorithms. However, in some cases, especially when considering qualitative verification, symbolic methods can lead to exact results. Qualitative questions do not aim at computing neither approximating a probability, but are only concerned with almost-sure or non negligible behaviours (that is events either of probability one, or non zero). In some cases, qualitative model-checking can be derived from a combination of techniques for infinite state systems (such as abstractions) with methods for finite state probabilistic systems. However, when one is interested in computing (or rather approximating) precise probability values (neither 0 nor 1), exact methods are scarce. To deal with these questions, we either try to restrict to classes of systems where exact computations can be made, or look for approximation algorithms.

3.2.3. Analysis of infinite state systems defined by graph grammars

Currently, many techniques (reachability, model checking, ...) from finite state systems have been generalised to pushdown systems, that can be modeled by graph grammars. Several such extensions heavily depend on the actual definition of the pushdown automata, for example, how many top stack symbols may be read, or whether the existence of ε-transitions (silent transitions) is allowed. Many of these restrictions do not affect the actual structure of the graph, and interesting properties like reachability or satisfiability (of a formula) only depend on the structure of a graph.

Deterministic graph grammars enable to focus on structural properties of systems. The connexion with finite graph algorithms is often straightforward: for example reachability is simply the finite graph algorithm iterated on the right hand sides. On the other hand, extending these grammars with time or probabilities is not straightforward: qualitative values associated to each copy (in the graph) of the same vertex (in the grammar) is different, introducing more complex equations. Furthermore, the fact that the left-hand sides are single hyperarcs is a very strong restriction. But removing this restriction leads to non-recursive graphs. Identifying decidable families of graphs defined by contextual graph grammars is also very challenging.

3.3. Automatic test generation

We are mainly interested in conformance testing which consists in checking whether a black box implementation under test (the real system that is only known by its interface) behaves correctly with respect to its specification (the reference which specifies the intended behavior of the system). In the line of model-based testing, we use formal specifications and their underlying models to unambiguously define the intended behavior of the system, to formally define conformance and to design test case generation algorithms. The difficult problems are to generate test cases that correctly identify faults (the oracle problem) and, as exhaustiveness is impossible to reach in practice, to select an adequate subset of test cases that are likely to detect faults. Hereafter we detail some elements of the models, theories and algorithms we use.
We use IOLTS (or IOSTS) as formal models for specifications, implementations, test purposes, and test cases. We adapt a well established theory of conformance testing [32], which formally defines conformance as a relation between formal models of specifications and implementations. This conformance relation, called **ioco** compares the visible behaviors (called *suspension traces*) of the implementation I (denoted by $STraces(I)$) with those of the specification S ($STraces(S)$). Suspension traces are sequence of inputs, outputs or quiescence (absence of action denoted by δ), thus abstract away internal behaviors that cannot be observed by testers. Intuitively, $I\ ioco\ S$ if after a suspension trace of the specification, the implementation I can only show outputs and quiescences of the specification S. We re-formulated ioco as a partial inclusion of visible behaviors as follows:

$$I\ ioco\ S \Leftrightarrow STraces(I) \cap [STraces(S).\Lambda_!^\delta \setminus STraces(S)] = \varnothing.$$

In other words, suspension traces of I which are suspension traces of S prolongated by an output or quiescence, should still be suspension traces of S.

Interestingly, this characterization presents conformance with respect to S as a safety property of suspension traces of I. The negation of this property is charaterized by a *canonical tester* $Can(S)$ which recognizes exactly $[STraces(S).\Lambda_!^\delta \setminus STraces(S)]$, the set of non-conformant suspension traces. This *canonical tester* also serves as a basis for test selection.

Test cases are processes executed against implementations in order to detect non-conformance. They are also formalized by IOLTS (or IOSTS) with special states indicating *verdicts*. The execution of test cases against implementations is formalized by a parallel composition with synchronization on common actions. A *Fail* verdict means that the IUT is rejected and should correspond to non-conformance, a *Pass* verdict means that the IUT exhibited a correct behavior and some specific targeted behaviour has been observed, while an *Inconclusive* verdict is given to a correct behavior that is not targeted.

Test suites (sets of test cases) are required to exhibit some properties relating the verdict they produce to the conformance relation. Soundness means that only non conformant implementations should be rejected by a test suite and exhaustiveness means that every non conformant implementation may be rejected by the test suite. Soundness is not difficult to obtain, but exhaustiveness is not possible in practice and one has to select test cases.
Test selection is often based on the coverage of some criteria (state coverage, transition coverage, etc). But test cases are often associated with *test purposes* describing some abstract behaviors targeted by a test case. In our framework, test purposes are specified as IOLTS (or IOSTS) associated with marked states or dedicated variables, giving them the status of automata or observers accepting runs (or sequences of actions or suspension traces). Selection of test cases amounts to selecting traces of the canonical tester accepted by the test purpose. The resulting test case is then both an observer of the negation of a safety property (non-conformance wrt. S), and an observer of a reachability property (acceptance by the test purpose). Selection can be reduced to a model-checking problem where one wants to identify states (and transitions between them) which are both reachable from the initial state and co-reachable from the accepting states. We have proved that these algorithms ensure soundness. Moreover the (infinite) set of all possibly generated test cases is also exhaustive. Apart from these theoretical results, our algorithms are designed to be as efficient as possible in order to be able to scale up to real applications.

Our first test generation algorithms are based on enumerative techniques, thus adapted to IOLTS models, and optimized to fight the state-space explosion problem. On-the-fly algorithms where designed and implemented in the TGV tool (see 5.1), which consist in computing co-reachable states from a target state during a lazy exploration of the set of reachable states in a product of the specification and the test purpose [4]. However, this enumerative technique suffers from some limitations when specification models contain data.

More recently, we have explored symbolic test generation techniques for IOSTS specifications [31]. The objective is to avoid the state space explosion problem induced by the enumeration of values of variables and communication parameters. The idea consists in computing a test case under the form of an *IOSTS*, i.e., a reactive program in which the operations on data are kept in a symbolic form. Test selection is still based on test purposes (also described as IOSTS) and involves syntactical transformations of IOSTS models that should ensure properties of their IOLTS semantics. However, most of the operations involved in test generation (determinisation, reachability, and coreachability) become undecidable. For determinisation we employ heuristics that allow us to solve the so-called bounded observable non-determinism (i.e., the result of an internal choice can be detected after finitely many observable actions). The product is defined syntactically. Finally test selection is performed as a syntactical transformation of transitions which is based on a semantical reachability and co-reachability analysis. As both problems are undecidable for IOSTS, syntactical transformations are guided by over-approximations using abstract interpretation techniques. Nevertheless, these over-approximations still ensure soundness of test cases [5]. These techniques are implemented in the STG tool (see 5.2), with an interface with NBAC used for abstract interpretation.

3.4. Control synthesis

The supervisory control problem is concerned with ensuring (not only checking) that a computer-operated system works correctly. More precisely, given a specification model and a required property, the problem is to control the specification's behavior, by coupling it to a supervisor, such that the controlled specification satisfies the property [30]. The models used are LTSs and the associated languages, which make a distinction between *controllable* and *non-controllable* actions and between *observable* and *non-observable* actions. Typically, the controlled system is constrained by the supervisor, which acts on the system's controllable actions and forces it to behave as specified by the property. The control synthesis problem can be seen as a constructive verification problem: building a supervisor that prevents the system from violating a property. Several kinds of properties can be ensured such as reachability, invariance (i.e. safety), attractivity, etc. Techniques adapted from model checking are then used to compute the supervisor w.r.t. the objectives. Optimality must be taken into account as one often wants to obtain a supervisor that constrains the system as few as possible.

Supervisory control theory overview. Supervisory control theory deals with control of Discrete Event Systems. In this theory, the behavior of the system S is assumed not to be fully satisfactory. Hence, it has to be reduced by means of a feedback control (named Supervisor or Controller) in order to achieve a given set of requirements [30]. Namely, if S denotes the specification of the system and Φ is a safety property that has to be ensured on S (i.e. $S\neg\models\Phi$), the problem consists in computing a supervisor $\mathcal{C}$, such that

$$S\|\mathcal{C} \models \Phi \tag{1}$$

where $\|$ is the classical parallel composition between two LTSs. Given S, some events of S are said to be uncontrollable (Σ_{uc}), i.e. the occurrence of these events cannot be prevented by a supervisor, while the others are controllable (Σ_c). It means that all the supervisors satisfying (1) are not good candidates. In fact, the behavior of the controlled system must respect an additional condition that happens to be similar to the *ioco* conformance relation that we previously defined in 3.3. This condition is called the *controllability condition* and is defined as follows.

$$\mathcal{L}(S\|\mathcal{C})\Sigma_{uc} \cap \mathcal{L}(S) \subseteq \mathcal{L}(S\|\mathcal{C}) \tag{2}$$

Namely, when acting on S, a supervisor is not allowed to disable uncontrollable events. Given a safety property Φ, that can be modeled by an LTS A_Φ, there actually exist many different supervisors satisyfing both (1) and (2). Among all the valid supervisors, we are interested in computing the supremal one, ie the one that restricts the system as few as possible. It has been shown in [30] that such a supervisor always exists and is unique. It gives access to a behavior of the controlled system that is called the supremal controllable sub-language of A_Φ w.r.t. S and Σ_{uc}. In some situations, it may also be interesting to force the controlled system to be non-blocking (See [30] for details).

The underlying techniques are similar to the ones used for Automatic Test Generation. It consists in computing a product between the specification and A_Φ and to remove the states of the obtained LTS that may lead to states that violate the property by triggering only uncontrollable events.

4. Application Domains

4.1. Overview

The methods and tools developed by the VERTECS project-team for test generation and control synthesis of reactive systems are intended to be as generic as possible. This allows us to apply them in many application domains where the presence of software is predominant and its correctness is essential. In particular, we apply our research in the context of telecommunication systems, for embedded systems, for smart-cards application, and control-command systems.

4.2. Telecommunication systems

Our research on test generation was initially proposed for conformance testing of telecommunication protocols. In this domain, testing is a normalized process [26], and formal specification languages are widely used (SDL in particular). Our test generation techniques have already proved useful in this context, going up to industrial transfer. New standardized component-based design methodologies such as UML and OMG's MDE increase the need for formal techniques in order to ensure the compositionality of components, by verification and testing. Our techniques, by their genericity and adaptativity, have also proved useful at different levels of these methodologies, from component testing to system testing. The telecommunication industry now also tries to provide more and more services to the users. These services must be validated.

4.3. Software embedded systems

In the context of transport, software embedded systems are increasingly predominant. This is particularly important in automotive systems, where software replaces electronics for power train, chassis (e.g. engine control, steering, brakes) and cabin (e.g. wiper, windows, air conditioning) or new services to passengers are increasing (e.g. telematics, entertainment). Car manufacturers have to integrate software components provided by many different suppliers, according to specifications. One of the problems is that testing is done late in the life cycle, when the complete system is available. Faced with these problems, but also to the complexity of systems, compositionality of components, distribution, etc, car manufacturers now try to promote standardized interfaces and component-based design methodologies. They also develop virtual platforms which allow for testing components before the system is complete. It is clear that software quality and trust are one of the problems that have to be tackled in this context. This is why we believe that our techniques (testing and control) can be useful in such a context.

4.4. Control-command systems

The main application domain for our techniques is control-command systems. In general, such systems control costly machines (see e.g. robotic systems, flexible manufacturing systems), that are connected to an environment (e.g. a human operator). Such systems are often critical systems and errors occurring during their execution may have dramatic economical or human consequences. In this field, the controller synthesis methodology (CSM) is useful to ensure by construction the interaction between 1) the different components, and 2) the environment and the system itself. For the first point, the CSM is often used as a safe scheduler, whereas for the second one, the supervisor can be interpreted as a safe discrete tele-operation system. Also in the context of the Vacsim ANR project, we investigate the testing, monitoring and verification of control-command systems.

5. Software

5.1. TGV

Participant: Thierry Jéron.

TGV (Test Generation with Verification technology) is a tool for test generation of conformance test suites from specifications of reactive systems [4]. It is based on the IOLTS model, a well defined theory of testing, and on-the-fly test generation algorithms coming from verification technology. Originally, TGV allows test generation focused on well defined behaviors formalized by test purposes. The main operations of TGV are (1) a synchronous product which identifies sequences of the specification accepted by a test purpose, (2) abstraction and determinisation for the computation of next visible actions, (3) selection of test cases by the computation of reachable states from the initial states and co-reachable states from accepting states. TGV has been developed in collaboration with Vérimag Grenoble and uses libraries of the CADP toolbox (VERIMAG and VASY). TGV can be seen as a library that can be linked to different simulation tools through well defined APIs. An academic version of TGV is distributed in the CADP toolbox and allows test generation from Lotos specifications by a connection to its simulator API. TGV has been registered at APP (*Agence de Protection des Programmes*) under deposit number IDDN.FR.001.310012.00.R.P.1997.000.2090.

5.2. STG

Participant: Thierry Jéron.

STG (Symbolic Test Generation) is a prototype tool for the generation and execution of test cases using symbolic techniques. It takes as input a specification and a test purpose described as IOSTS, and generates a test case program also in the form of IOSTS. Test generation in STG is based on a syntactic product of the specification and test purpose IOSTS, an extraction of the subgraph corresponding to the test purpose, elimination of internal actions, determinisation, and simplification. The simplification phase now relies on NBAC, which approximates reachable and coreachable states using abstract interpretation. It is used to eliminate unreachable states, and to strengthen the guards of system inputs in order to eliminate some *Inconclusive* verdicts. After a translation into C++ or Java, test cases can be executed on an implementation in the corresponding language. Constraints on system input parameters are solved on-the-fly (i.e. during execution) using a constraint solver. The first version of STG was developed in C++, using Omega as constraint solver during execution. This version has been deposited at APP under number IDDN.FR.001.510006.000.S.P.2004.000.10600.

A new version in OCaml has been developed in the last years. This version is more generic and will serve as a library for symbolic operations on IOSTS. Most functionalities of the C++ version have been re-implemented. Also a new translation of abstract test cases into Java executable tests has been developed, in which the constraint solver is LUCKYDRAW (VERIMAG). This version has also been deposit at APP and is available for download on the web as well as its documentation and some examples.

Finally, in collaboration with ULB, we implemented a prototype SMACS, derived from STG, that is devoted to the control of infinite system modeled by STS.

5.3. SIGALI

Participant: Hervé Marchand.

SIGALI is a model-checking tool that operates on ILTS (Implicit Labeled Transition Systems, an equational representation of an automaton), an intermediate model for discrete event systems. It offers functionalities for verification of reactive systems and discrete controller synthesis. It is developed jointly by the ESPRESSO and VERTECS teams. The techniques used consist in manipulating the system of equations instead of the set of solutions, which avoids the enumeration of the state space. Each set of states is uniquely characterized by a predicate and the operations on sets can be equivalently performed on the associated predicates. Therefore, a wide spectrum of properties, such as liveness, invariance, reachability and attractivity, can be checked. Algorithms for the computation of predicates on states are also available [6] [28]. SIGALI is connected with the Polychrony environment (ESPRESSO project-team) as well as the Matou environment (VERIMAG), thus allowing the modeling of reactive systems by means of Signal Specification or Mode Automata and the visualization of the synthesized controller by an interactive simulation of the controlled system. SIGALI is registered at APP.

6. New Results

6.1. Verification

6.1.1. Analysis of partially observed recursive discrete-event systems

Participants: Sébastien Chédor, Thierry Jéron, Hervé Marchand, Christophe Morvan.

Monitoring of recursive discrete-event systems under partial observation is an important issue with major applications such as the diagnosability of faulty behaviors and the detection of information flow. We consider regular discrete-event systems, that is recursive discrete-event systems definable by deterministic graph grammars. This setting is expressive enough to capture classical models of recursive systems such as the pushdown systems. Hence they are infinite-state in general and standard powerset constructions for monitoring do not apply anymore. We exhibit computable conditions on these grammars together with non-trivial transformations of graph grammars that enable us to construct a monitor. This construction is applied to diagnose faulty behaviors, to detect information flow in regular discrete-event systems, and to generate tests.

6.1.2. Analysis of timed systems

6.1.2.1. Approximate determinization of timed automata

Participants: Nathalie Bertrand, Thierry Jéron, Amélie Stainer.

Timed automata are frequently used to model real-time systems. Their determinization is a key issue for several validation problems. However, not all timed automata can be determinized, and determinizability itself is undecidable. In [18], we propose a game-based algorithm which, given a timed automaton, tries to produce a language-equivalent deterministic timed automaton, otherwise a deterministic over-approximation. Our method subsumes two recent contributions: it is at once more general than an existing (non terminating) determinization procedure by Baier *et al.* (2009) and more precise than the approximation algorithm of Krichen and Tripakis (2009). Moreover, an extension of the method allows to deal with invariants and ϵ-transitions, and to consider other useful approximations: under approximation, and combination of under- and over-approximations which are particularly useful in testing (see 6.2.1).

6.1.2.2. Frequency analysis for timed automata

Participants: Nathalie Bertrand, Amélie Stainer.

The languages of infinite timed words accepted by timed automata are traditionally defined using Büchi-like conditions. These acceptance conditions focus on the set of locations visited infinitely often along a run, but completely ignore quantitative timing aspects. In [15] we propose a natural quantitative semantics for timed automata based on the so-called frequency, which measures the proportion of time spent in the accepting states. We study various properties of timed languages accepted with positive frequency, and in particular the emptiness and universality problems.

6.1.3. Petri nets reachability graphs

Participant: Christophe Morvan.

Petri nets are a general model for concurrency, the structure of their reachability graph is mostly unknown. In [19] we have investigated the decidability and complexity status of model-checking problems on unlabelled reachability graphs of Petri nets by considering first-order, modal and pattern-based languages without labels on transitions or atomic propositions on markings. We consider several parameters to separate decidable problems from undecidable ones. These results illustrate the intrinsic complexity of the structure of these graphs.

6.2. Active and passive testing

6.2.1. Off-line test selection with test purposes for non-deterministic timed automata

Participants: Nathalie Bertrand, Thierry Jéron, Amélie Stainer.

In [17], we propose novel off-line test generation techniques for non-deterministic timed automata with inputs and outputs (TAIOs) in the formal framework of the tioco conformance theory. In this context, a first problem is the determinization of TAIOs, which is necessary to foresee next enabled actions, but is in general impossible. The determinization problem is addressed in [18] thanks to an approximate determinization using a game approach (see 6.1.2.1). We adapt this procedure here to over- and under-approximation, in order to preserve tioco and guarantee the soundness of generated test cases. A second problem is test selection for which a precise description of timed behaviors to be tested is carried out by expressive test purposes modeled by a generalization of TAIOs. Finally, using a symbolic co-reachability analysis guided by the test purpose, test cases are generated in the form of TAIOs equipped with verdicts.

6.2.2. Test generation using pushdown automata

Participant: Puneet Bhateja.

IOLTS (*input output labeled transition system*) is a versatile model and is frequently used in model based testing to model the functional behavior of an IUT (implementation under test). However when a system is tested remotely, its observed behavior can be different from its actual functional behavior. In a previous paper, we defined a notion of remotely observed behavior of an IOLTS in terms of its actual behavior. Paper [14] contributes by proposing a methodology to simulate a PDA (*pushdown automaton*) from the given IOLTS such that the simulated PDA precisely expresses the remotely observed behavior of the IOLTS. The simulated PDA can be thought of as an automatic test generator for remote testing.

6.2.3. Test case selection in asynchronous testing

Participants: Puneet Bhateja, Thierry Jéron.

Conformance testing has a rich underlying formal theory called IOLTS-based conformance testing. Depending upon whether the implementation-under-test (IUT) interacts with its environment directly, or indirectly through a medium, IOLTS-based conformance testing can be classified as synchronous testing or asynchronous testing, respectively. So far the problem of test case selection has been addressed mostly in the context of synchronous testing. In this work we contribute by addressing this problem in the context of asynchronous testing. Though an asynchronously communicating process can be simulated by a synchronously communicating process, the fact that the simulating process is infinite state even if the simulated process is finite state made the problem challenging.

6.2.4. A tagging protocol for asynchronous testing

Participant: Puneet Bhateja.

Conformance testing has a rich underlying theory popularly called IOCO-test theory. In the realm of IOCO-test theory, this paper addresses the issue of testing a component of an asynchronously communicating distributed system. Testing a system which communicates asynchronously (i.e., through some medium) with its environment is more difficult than testing a system which communicates synchronously (i.e., directly without any medium). What impedes asynchronous testing is that the actual behavior of the implementation under test (IUT) appears distorted and infinite to the tester. This impediment consequently renders the problem of generating a complete test suite, from the given specification of the IUT, infeasible. To this end, paper [13] proposes a tagging protocol which when implemented by the asynchronously communicating distributed system will enable the generation of a complete test suite, from the specification of any of its component. Further, this paper describes how to generate the test suite from the given specification of the component.

6.2.5. Abstracting time and data for conformance testing of real-time systems

Participants: Thierry Jéron, Hervé Marchand.

Current approaches to model-based conformance testing of real-time systems are mostly based either on finite state machines/transition systems or on timed automata. However, most real-time systems manipulate data while being subjected to time constraints. The usual solution consists in enumerating data values (in finite domains) while treating time symbolically, thus leading to the classical state explosion problem. Paper [12] with W.L. Andrade and P. Machado (Fed. Univ. Campina Grande) proposes a new model of real-time systems as an extension of both symbolic transition systems and timed automata, in order to handle both data and time requirements symbolically. We then adapt the tioco conformance testing theory to deal with this model and describe a test case generation process based on a combination of symbolic execution and constraint solving for the data part and symbolic analysis for timed aspects.

6.2.6. Ensuring security properties

6.2.6.1. Runtime enforcement monitors: composition, synthesis, and enforcement abilities

Participant: Yliès Falcone.

Runtime enforcement is a powerful technique to ensure that a program will respect a given set of properties. In [9] we extend previous work on this topic in several directions. Firstly, we propose a generic notion of enforcement monitors based on a memory device and finite sets of control states and enforcement operations. Moreover, we specify their enforcement abilities w.r.t. the general Safety-Progress classification of properties. Furthermore, we propose a systematic technique to produce a monitor from the automaton recognizing a given safety, guarantee, obligation or response property. Finally, we show that this notion of enforcement monitors is more amenable to implementation and encompasses previous runtime enforcement mechanisms.

6.2.6.2. What can you verify and enforce at runtime?

Participant: Yliès Falcone.

The underlying property, its definition and representation play a major role when monitoring a system. Having a suitable and convenient framework to express properties is thus a concern for runtime analysis. It is desirable to delineate in this framework the sets of properties for which runtime analysis approaches can be applied to. [8] presents a unified view of runtime verification and enforcement of properties in the Safety-Progress classification. Firstly, we extend the Safety-Progress classification of properties in a runtime context. Secondly, we characterize the set of properties which can be verified (monitorable properties) and enforced (enforceable properties) at runtime. We propose in particular an alternative definition of ”property monitoring” to the one classically used in this context. Finally, for the delineated sets of properties, we define specialized verification and enforcement monitors.

6.3. Control synthesis

6.3.1. Controllers for probabilistic systems

Participant: Nathalie Bertrand.

Partially Observable Markov Decision Processes (*POMDP* for short) have been extensively studied in several research communities, among which AI and model-checking. In [16] we address the problem of the *minimal information* a user needs *at runtime* to achieve a simple goal, modeled as reaching an objective with probability one. More precisely, to achieve her goal, the user can either choose at each step to use partial information only, or pay a fixed cost and receive full information. The natural question is then to minimize the cost the user needs to fulfill its objective. This optimization question gives rise to two different problems, whether we consider to minimize the *worst case cost*, or the *average cost*. On the one hand, concerning the worst case cost, we show that efficient techniques from the model checking community can be adapted to compute the optimal worst case cost and give optimal strategies for the users. On the other hand, we show that the optimal average price (a question typically considered in the AI community) cannot be computed in general, nor can it be approximated in polynomial time even up to a large approximation factor.

6.3.2. Supervisory control for synchronous systems

6.3.2.1. Controller synthesis and programming language

Participant: Hervé Marchand.

In [24] we define a mixed imperative/declarative programming language: declarative contracts are enforced upon imperatively described behaviors. We rely on the notion of Discrete Controller Synthesis (DCS), a formal technique stemming from control theory and the supervisory control of discrete event systems. We target the application domain of adaptive and reconfigurable computing systems: our language can serve programming closed-loop adaptation controllers, enabling flexible execution of functionalities w.r.t. changing resource and environment conditions. We give a synthetic presentation of the language, its semantics and compilation, and we illustrate its use with the example of a robot system.

6.3.2.2. Symbolic supervisory control of infinite transition systems under partial observation using abstract interpretation

Participant: Hervé Marchand.

In [11], we propose algorithms for the synthesis of state-feedback controllers with partial observation of infinite state discrete event systems modelled by Symbolic Transition Systems. We provide models of safe memoryless controllers both for potentially deadlocking and deadlock free controlled systems. The termination of the algorithms solving these problems is ensured using abstract interpretation techniques which provide an overapproximation of the transitions to disable. We then extend our algorithms to controllers with memory and to online controllers. We also propose improvements in the synthesis of controllers in the finite case which, to our knowledge, provide more permissive solutions than what was previously proposed in the literature. Our tool SMACS gives an empirical validation of our methods by showing their feasibility, usability and efficiency.

6.3.2.3. Decentralized control of infinite systems

Participant: Hervé Marchand.

In [10] we propose algorithms for the synthesis of decentralized state-feedback controllers with partial observation of infinite state systems, which are modeled by Symbolic Transition Systems. We first consider the computation of safe controllers ensuring the avoidance of a set of forbidden states and then extend this result to the deadlock free case. The termination of the algorithms solving these problems is ensured by the use of abstract interpretation techniques, but at the price of overapproximations, in particular, in the computation of the states which must be avoided. We then extend our algorithms to the case where the system to be controlled is given by a collection of subsystems (modules). This structure is exploited to locally compute a controller for each module. Our tool SMACS gives an empirical evaluation of our methods by showing their feasibility, usability and efficiency.

6.3.2.4. Polychronous controller synthesis from MARTE CCSL timing specifications

Participant: Hervé Marchand.

The UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) defines a mathematically expressive model of time, the Clock Constraint Specification Language (CCSL), to specify timed annotations on UML diagrams and thus provides them with formally defined timed interpretations. Thanks to its expressive capability, the CCSL allows for the specification of static and dynamic properties, of deterministic and non-deterministic behaviors, or of systems with multiple clock domains. Code generation from such multiclocked specifications (for the purpose of synthesizing a simulator, for instance) is known to be a difficult issue. We address it in [23] by using the approach of controller synthesis. In our framework, a timed CCSL specification is regarded as a property whose satisfaction should be enforced for any UML diagram carrying it as annotation. To do so, CCSL statements are first translated into dynamical polynomial systems. Such systems can be manipulated using the model-checker Sigali to synthesize an executable property (a controller) which enforces the satisfaction of the specified timing constraints on the UML diagram with which it is executed.

6.3.3. Control of distributed systems

Participant: Hervé Marchand.

In this work, we consider the control of distributed systems composed of subsystems communicating asynchronously; the aim is to build local controllers that restrict the behavior of a distributed system in order to satisfy a global state avoidance property. We model our distributed systems as communicating finite state machines with reliable unbounded FIFO queues between subsystems. Local controllers can only observe the behavior of their local subsystem and do not see the queue contents. To refine their control policy, the controllers can use the FIFO queues to communicate by piggybacking extra information (some timestamps and their state estimates) to the messages sent by the subsystems [21]. We provide an algorithm that computes, for each local subsystem (and thus for each controller), during the execution of the system, an estimate of the current global state of the distributed system. The local estimate is updated at each message reception. We then define synthesis algorithms allowing to compute the local controllers. Our method relies on the computation of (co)reachable states. Since the reachability problem is undecidable in our model, we use abstract interpretation techniques to obtain regular overapproximations of the possible FIFO queue contents, and hence of the possible current global states. An implementation of our algorithms provides an empirical evaluation of our method [22].

7. Partnerships and Cooperations

7.1. National initiatives

7.1.1. *ANR TesTec: Test of real-time and critical embedded system*

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

The TesTec project is a three years [2008-2010] industrial research project that gathers two companies: an end-user (EDF R&D) and one software editor for embedded real-time systems and automation systems (Geensys), and four laboratories from automation engineering and computer science (I3S, INRIA Rennes, LaBRI, LURPA). This project focuses on automatic generation and execution of tests for the class of embedded real-time systems. They are highly critical. Such systems can be found in many industrial domains, such as energy, transport systems. More precisely the project TesTec will address two crucial technological issues:

- optimisation of test generation techniques for large size systems, in particular by an explicit modelling of time and by simultaneous management of continuous and discrete variables in hybrid applications;
- reduction of the size of the tests derived from specification models by using the results of formal verification of implementation models.

The overall aim of this project is to propose a software tool for generation and execution of tests; this tool will be based on an existing environment for embedded systems design and will implement the scientific results of the project.

This year our contributions to this project were our works on test generation from timed models, as well as approximate determinization of timed automata.

In 2011, the post-doc position of Puneet Bhateja was funded by TestTec.

7.1.2. *ANR VACSIM: Validation of critical control-command systems by coupling simulation and formal analysis*

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

The Vacsim project (2011-2014) is a 3 years project with EDF R&D, Dassault Systèmes, LURPA Cachan, I3S Nice and Labri Bordeaux. The project aims at developping both methodological and formal contributions for the simulation and validation of control-command systems. The rôle of the Vertecs team will be to contribute to the advance of validation techniques for timed systems, including quantitative analysis and its application to testing, monitoring of timed systems, and verification of communicating timed automata.

7.1.3. *Action Incitative VeSPa: Verification of security and privacy properties*

Participant: Nathalie Bertrand.

The VeSPa "Action Incitative" is a one-year [2011] project funded by Rennes 1 University to develop emerging research themes. The goal of the project is to strat and verify security and privacy properties in protocols, using logic and games techniques. The participants are Sophie Pinchinat (leader, S4), Sébastien Gambs (Cidre), Guillaume Aucher (DistribCom), and Nathalie Bertrand (Vertecs). To gather researchers interested in the topic, the second edition of a workshop on Games, Logics and Security has been organized in October 2011.

7.2. European initiatives

7.2.1. *Artist design network of excellence*

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

Program: FP7

Project acronym: Artist Design

Project title: Artist - European Network of Excellence on Embedded System Design

Duration: 01/08 - 12/11

Coordinator: VERIMAG

Abstract: The central objective for ArtistDesign http://www.artist-embedded.org/artist/-ArtistDesign-Participants-.html is to build on existing structures and links forged in Artist2, to become a virtual Center of Excellence in Embedded Systems Design. This will be mainly achieved through tight integration between the central players of the European research community. Also, the consortium is smaller, and integrates several new partners. These teams have already established a long-term vision for embedded systems in Europe, which advances the emergence of Embedded Systems as a mature discipline.
The research effort aims at integrating topics, teams, and competencies, grouped into 4 Thematic Clusters: “Modelling and Validation”, “Software Synthesis, Code Generation, and Timing Analysis”, “Operating Systems and Networks”, “Platforms and MPSoC”. “Transversal Integration” covering both industrial applications and design issues aims for integration between clusters.
The Vertecs EPI is a partner of the “Validation” activity of the “Modeling and Validation” cluster. This year, the Vertecs EPI has contributed to quantitative verification of timed automata [15], approximate determinization of timed automata [18] and its adaptation to test generation [17], and control sysnthesis using abstract interpretation for infinite state systems [11], on decentralized [10] and distributed control [21], [22]. Amélie Stainer spent one month in Aalborg to implement the approximate determinization of timed automata using UPPAAL libraries.

7.2.2. PHC Tournesol STP : Verification of timed and probabilistic systems

Participants: Nathalie Bertrand, Amélie Stainer.

A two-year contract with the group of Thomas Brihaye (Université Mons) started in 2010. Its objective is to study timed and probabilistic systems. This year, Nathalie Bertrand visited Thomas Brihaye in Mons, and Thomas Brihaye came to Rennes to give a seminar and further discuss with Nathalie Bertrand and Amélie Stainer.

7.2.3. Followed collaborations with major European organizations

Université Libre Bruxelles (Belgium), Prof. Thierry Massart

Testing and control of symbolic transitions systems

University of Kaiserslautern (Germany), Roland Meyer

Petri nets

University of Dresden (Germany), Prof. Christel Baier

Probabilistic automata over infinite words

7.3. International Initiatives

7.3.1. INRIA associate team

7.3.1.1. TREATIES

Title: Test of Real-Time Embedded Systems

INRIA principal investigator: Thierry Jéron

International Partner:

Institution: Federal University of Campina Grande (Brazil)

Laboratory: Universidade Federal do Campina Grande

Duration: 2009 - 2011

See also: http://www.irisa.fr/vertecs/Treaties.html

This associated team with the Federal University of Campina Grande (Prof. Patrícia D. L. Machado) and University Pernambuco (Prof. Augusto Sampaio) in Brazil started in 2009 and ended this year.

In 2011 Nathalie Bertrand and Sébastien Chédor visited the Brazilian team in Recife in November where a meeting took place, and we had the visit of Wilkerson Andrade in November.

This year the cooperation addressed problems in test generation for timed input/output symbolic transition systems (see 6.2.5) and compositional conformance verification for these models, on the problems of non-determinism in timed models for test generation (see 6.1.2.1 and 6.2.1), on test vector generation for timed models, and automatic test case generation and execution for regular graphs (see 6.2.2).

7.3.2. INRIA international partners

University of Michigan (Prof. Stéphane Lafortune) on control and diagnosis of discrete event systems.

7.3.3. Visits of international scientists

Laurie Ricker, associate professor at the Mathematics & Computer Science department of Mount Allison University (Canada) has visited Vertecs for 6 months, from January 2011 to June 2011. We collaborate on control of discrete event systems for distributed systems.

8. Dissemination

8.1. Animation of the scientific community

Nathalie Bertrand was PC member of QAPL'11, and co-organiser of the 2nd edition of the GIPSy Workshop in October 2011. She was invited to give a talk at the Seminar of the Centre Fédéré en Vérification in Brussels, in May 2011.

Yliès Falcone was a reviewer for DATE'12, ACM Tissec, Elsevier Cosrev, ICFEM'11, HSCC'11. He was invited to give a talk at LIG (Grenoble), LORIA (Nancy), LRI (Paris), NICTA (Canberra, Asutralia), Valence (in the context of the SEMBA project). He is in the Organization Committee of AFADL'12.

Thierry Jéron was PC member of Scenario'2011, ICTSS'2011, ICST'2012, TAP'2011. He was invited to give a talk in LIFC Besançon. He is member of IFIP WG 10.2. He was member of the PhD commitee of Alexander Heussner (Labri Bordeaux) and reviewer of the PhD thesis of Gilles Benattar (University of Nantes), Julien Provost (LURPA, ENS Cachan) and Pierre-Christophe Bué (LIFC Besançon).

Hervé Marchand is Associate Editor of the IEEE Transactions on Automatic Control journal and member of the IFAC Technical Committee (TC 1.3 on Discrete Event and Hybrid Systems). He was PC member of the ICINCO'11, DCDS,11, MSR'11 Conferences and IFAC World Congress 2011. He was member of the PhD committee of Mingming REN (INSA de Lyon, July 2011).

8.2. Teaching and supervision

8.2.1. Teaching

Nathalie Bertrand

Master : Advanced model-checking, 8h, niveau M2, ISTIC, Université de Rennes 1, France.

Agrégation : Langages formels, 16h, niveau M2, ENS Cachan antenne de Bretagne, France.

Sébastien Chédor

Licence : Java (TD-TP), 40h, niveau L1, ISTIC, Université de Rennes 1, France.
Scheme (TP), 20h, niveau L1, ISTIC, Université de Rennes 1, France.
Java (TP), 20h, niveau L2, ISTIC, Université de Rennes 1, France.
Programmation (TP), 20h, niveau L3, ENS-Cachan-Antenne de Bretagne, France.
Initiation à LaTeX (Cours), 5h, niveau L3, ENS-Cachan-Antenne de Bretagne, France.

Thierry Jéron

Master : Testing of timed systems, 4h, niveau M2, ISTIC, Université de Rennes 1, France.
Testing of timed systems, 4h, niveau M2, ESIR, Université de Rennes 1, France.

Christophe Morvan

Licence : Object Oriented programming with Java, 192h, niveau L2, Université de Marne-la-Vallée, France.

Amélie Stainer

Licence : Java (Cours-TD-TP), 42h, niveau L1, INSA-Rennes, France.
Scheme (TP), 14h, niveau L1, INSA-Rennes, France.
Modélisation de structures de données (TP), 20h, niveau L3, INSA-Rennes, France.

Agrégation : Modélisation (Cours), 13h, niveau M2, ENS-Cachan-Antenne de Bretagne, France.
Oraux blancs de mathématiques et de modélisation, 24h, niveau M2, ENS-Cachan-Antenne de Bretagne, France.
Préparation de leçons d'informatique, 12h, niveau M2, ENS-Cachan-Antenne de Bretagne, France.

8.2.2. PhD

PhD in progress: Sébastien Chédor, Verification and test of systems modeled by regular graphs, started in September 2009, supervised by Christophe Morvan and Thierry Jéron.

PhD in progress: Amélie Stainer, Quantitative verification of timed automata, started in October 2010, supervised by Nathalie Bertrand and Thierry Jéron.

PhD in progress: Srinivas Pinisetty, Runtime validation of critical control-command systems, started in December 2011, supervised by Hervé Marchand and Thierry Jéron.

9. Bibliography

Major publications by the team in recent years

[1] C. BAIER, N. BERTRAND, PH. SCHNOEBELEN. *Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties*, in "ACM Transactions on Computational Logic", 2007, vol. 9, n^{o} 1.

[2] C. CONSTANT, T. JÉRON, H. MARCHAND, V. RUSU. *Integrating formal verification and conformance testing for reactive systems*, in "IEEE Transactions on Software Engineering", August 2007, vol. 33, n^{o} 8, p. 558-574.

[3] B. GAUDIN, H. MARCHAND. *An Efficient Modular Method for the Control of Concurrent Discrete Event Systems: A Language-Based Approach*, in "Discrete Event Dynamic System", 2007, vol. 17, n^{o} 2, p. 179-209.

[4] C. JARD, T. JÉRON. *TGV: theory, principles and algorithms, A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems*, in "Software Tools for Technology Transfer (STTT)", October 2004, vol. 6.

[5] B. JEANNET, T. JÉRON, V. RUSU, E. ZINOVIEVA. *Symbolic Test Selection based on Approximate Analysis*, in "11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), Volume 3440 of LNCS", Edinburgh (Scottland), April 2005, p. 349-364, http://www.irisa.fr/vertecs/Publis/Ps/tacas05.pdf.

[6] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment*, in "Discrete Event Dynamic System : Theory and Applications", Octobre 2000, vol. 10, n^{o} 4, p. 347-368, http://www.irisa.fr/vertecs/Publis/Ps/2000-J-DEDS.pdf.

[7] V. RUSU. *Verifying an ATM Protocol Using a Combination of Formal Techniques*, in "Computer Journal", November 2006, vol. 49, n^{o} 6, p. 710–730.

Publications of the year

Articles in International Peer-Reviewed Journal

[8] Y. FALCONE, J.-C. FERNANDEZ, L. MOUNIER. *What can you verify and Enforce at Runtime?*, in "Sotfware Tools for Technology Transfer", April 2011, To appear, http://hal.inria.fr/hal-00627594/en.

[9] Y. FALCONE, L. MOUNIER, J.-C. FERNANDEZ, J.-L. RICHIER. *Runtime Enforcement Monitors: composition, synthesis, and enforcement abilities*, in "Formal Methods in System Design", March 2011, p. 10.1007/s10703-011-0114-4, http://hal.inria.fr/hal-00576948/en.

[10] G. KALYON, T. LE GALL, H. MARCHAND, T. MASSART. *Decentralized Control of Infinite Systems*, in "Discrete Event Dynamic Systems", 2011, vol. 21, n^{o} 3, p. 359-393 [*DOI :* 10.1007/S10626-011-0106-Y], http://hal.inria.fr/inria-00594665/en.

[11] G. KALYON, T. LE GALL, H. MARCHAND, T. MASSART. *Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation*, in "Discrete Event Dynamic Systems", 2011 [*DOI :* 10.1007/s10626-011-0101-3], http://hal.inria.fr/inria-00586169/en.

International Conferences with Proceedings

[12] W. L. ANDRADE, P. D. L. MACHADO, T. JÉRON, H. MARCHAND. *Abstracting Time and Data for Conformance Testing of Real-Time Systems*, in "7th Workshop on Advances in Model Based Testing A-MOST 2011", Berlin, Germany, 2011, http://hal.inria.fr/hal-00646089/en.

[13] P. BATHEJA. *A Tagging Protocol for Asynchronous Testing*, in "IEEE International Conference on Theoretical Aspects of Software Engineering", Xi'an, China, IEEE, 2011, http://hal.inria.fr/inria-00628769/en.

[14] P. BATHEJA. *Test Case Generation Using PDA*, in "IEEE International Conference on Theoretical Aspects of Software Engineering", Xi'an, China, IEEE, 2011, http://hal.inria.fr/inria-00628763/en.

[15] N. BERTRAND, P. BOUYER, T. BRIHAYE, A. STAINER. *Emptiness and Universality Problems in Timed Automata with Positive Frequency.*, in "38th International Colloquium on Automata, Languages and Program-

ming (ICALP'11)", Zürich, Switzerland, LNCS, Springer, 2011, vol. 6756, p. 246-257, http://hal.inria.fr/inria-00629172/en.

[16] N. BERTRAND, B. GENEST. *Minimal Disclosure in Partially Observable Markov Decision Processes*, in "31st Annual Conference Foundations on Software Technology and Theoretical Computer Science (FSTTCS'11)", Mumbai, India, 2011, http://hal.inria.fr/inria-00629101/en.

[17] N. BERTRAND, T. JÉRON, A. STAINER, M. KRICHEN. *Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata.*, in "17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)", Saarbrücken, Germany, LNCS, Springer, 2011, vol. 6605, p. 96-111, http://hal.inria.fr/inria-00629177/en.

[18] N. BERTRAND, A. STAINER, T. JÉRON, M. KRICHEN. *A Game Approach to Determinize Timed Automata.*, in "14th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'11)", Saarbrücken, Germany, LNCS, Springer, 2011, vol. 6604, p. 245-259, http://hal.inria.fr/inria-00629179/en.

[19] P. DARONDEAU, S. DEMRI, R. MEYER, C. MORVAN. *Petri Net Reachability Graphs: Decidability Status of FO Properties*, in "IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science", Bombay, India, S. CHAKRABORTY, A. KUMAR (editors), Leibniz International Proceedings in Informatics (LIPICS), Dagstuhl Publishing, 2011, http://hal.inria.fr/inria-00627657/en.

[20] Y. FALCONE, M. JABER, T.-H. NGUYEN, M. BOZGA, S. BENSALEM. *Runtime Verification of Component-Based Systems*, in "Software Engineering and Formal Methods", Montevideo, Uruguay, October 2011, http://hal.inria.fr/hal-00642969/en.

[21] G. KALYON, T. LE GALL, H. MARCHAND, T. MASSART. *Global State Estimates for Distributed Systems*, in "31th IFIP International Conference on FORmal TEchniques for Networked and Distributed Systems.", Reykjavik, Iceland, Lecture Notes in Computer Science, Springer, 2011, vol. 6722, p. 198-212 [*DOI :* 10.1007/978-3-642-21461-5_13], http://hal.inria.fr/inria-00581259/en.

[22] G. KALYON, T. LE GALL, H. MARCHAND, T. MASSART. *Synthesis of Communicating Controllers for Distributed Systems*, in "IEEE Conference on Decision and Control and European Control Conference", Orlando, United States, IEEE, 2011, http://hal.inria.fr/inria-00627574/en.

[23] H. YU, J.-P. TALPIN, L. BESNARD, T. GAUTIER, H. MARCHAND, P. LE GUERNIC. *Polychronous Controller Synthesis from MARTE CCSL Timing Specifications*, in "ACM/IEEE Ninth International Conference on Formal Methods and Models for Codesign (MEMOCODE)", Cambridge, United Kingdom, July 2011, http://hal.inria.fr/inria-00594942/en.

National Conferences with Proceeding

[24] G. DELAVAL, É. RUTTEN, H. MARCHAND. *Intégration de la synthèse de contrôleurs discrets dans un langage de programmation*, in "Modélisation des Systèmes Réactifs (MSR'11)", Lille, France, 2011, http://hal.inria.fr/inria-00629104/en.

Research Reports

[25] N. BERTRAND, T. JÉRON, A. STAINER, M. KRICHEN. *Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata*, INRIA, January 2011, n^{o} RR-7501, http://hal.inria.fr/inria-00550923/en.

References in notes

[26] *Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework - Part 1 : General Concept - Part 2 : Abstract Test Suite Specification - Part 3 : The Tree and Tabular Combined Notation (TTCN)*, 1992, n^{o} International Standard ISO/IEC 9646-1/2/3.

[27] R. ALUR, D. L. DILL. *A Theory of Timed Automata*, in "Theor. Comput. Sci.", 1994, vol. 126, n^{o} 2, p. 183-235.

[28] L. BESNARD, H. MARCHAND, É. RUTTEN. *The Sigali Tool Box Environment*, in "Workshop on Discrete Event Systems, WODES'06 (Tool Paper)", Ann-Arbor (MI, USA), July 2006, p. 465-466.

[29] P. COUSOT, R. COUSOT. *Abstract intrepretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in "Conference Record of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles (CA, USA)", January 1977, p. 238-252.

[30] P. J. RAMADGE, W. M. WONHAM. *The Control of Discrete Event Systems*, in "Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems", 1989, vol. 77, n^{o} 1, p. 81-98.

[31] V. RUSU, L. DU BOUSQUET, T. JÉRON. *An approach to symbolic test generation*, in "International Conference on Integrating Formal Methods (IFM'00)", Lecture Notes in Computer Science, 2000, vol. 1945, p. 338-357.

[32] J. TRETMANS. *Test Generation with Inputs, Outputs and Repetitive Quiescence.*, in "Software - Concepts and Tools", 1996, vol. 17, n^{o} 3, p. 103-120.