INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team vertecs

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

Rennes - Bretagne-Atlantique

Theme : Embedded and Real Time Systems

Table of contents

1. Team

Research Scientists

Thierry Jéron [Team Leader, Research Director (DR),INRIA, HdR]
Nathalie Bertrand [Research Associate (CR) INRIA]
Hervé Marchand [Research Associate (CR) INRIA]
Vlad Rusu [Research Associate (CR) INRIA, until September 2010, HdR]

PhD Students

Amélie Stainer [UNIVERSITÉ DE RENNES 1, since September 2010]
Sébastien Chédor [UNIVERSITÉ DE RENNES 1, since September 2009]

Post-Doctoral Fellows

Yliès Falcone [INRIA, since December 2009]
Puneet Bhateja [INRIA, since April 2010]

Visiting Scientist

Christophe Morvan [Assistant Professor, Univ. de Marne-la-Vallée]

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, theorem proving, 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

Yliès Falcone, Thierry Jéron and Hervé Marchand together with Jean-Claude Fernandez and Laurent Mounier (Verimag, University of Grenoble), received the IFIP Best Paper Award at the ICTSS'2010 conference (22nd IFIP International Conference on Testing Software and Systems), in Natal (Brasil), in novembre 2010, for the paper More Testable Properties[18]. ICTSS, which merges TestCom and FATES is a major conference on testing of software and systems.

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 to 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 [28]. 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 correspond 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 MTBDDs (Multi-Terminal Decision Diagrams) extension for manipulating more general functions. We use these algorithms for verification and test generation.
- 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 set of states and permits exact model-checking of some well-formed infinite-state systems. However, for feasibility reasons, it is often mandatory to make the use of approximate computations, either by computing a finite abstraction and resort to graph algorithms, or preferably by using more sophisticated abstract interpretation techniques. Another way to cope with large or infinite state systems is deductive verification, which, either alone or in combination with compositional and abstraction techniques, can deal with complex systems that are beyond the scope of fully automatic methods. 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 four verification topics we are interested in: abstract interpretation, theorem proving, model-checking of infinite state and probabilistic systems 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 [30]. 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. Theorem Proving

For verification we also use theorem proving and more particularly the PVS [32] and COQ [33] proof assistants. These are two general-purpose systems based on two different versions of higher-order logic. A verification task in such a proof assistant consists in encoding the system under verification and its properties into the logic of the proof assistant, together with verification *rules* that allow to prove the properties. Using the rules usually requires input from the user; for example, proving that a state predicate holds in every reachable state of the system (i.e., it is an *invariant*) typically requires to provide a stronger, *inductive* invariant, which is preserved by every execution step of the system. Another type of verification problem is proving *simulation* between a concrete and an abstract semantics of a system. This can also be done by induction in a systematic manner, by showing that, in each reachable state of the system, each step of the concrete system is simulated by a corresponding step at the abstract level.

3.2.3. Model-checking of infinite state and 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.

3.2.4. 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 [36], 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 [35]. 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. Controller 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 [34]. 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.

The 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 [34]. 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 [34] 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 [34] 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. Panorama

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 [31], 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. We are involved with France Telecom R & D in a project on the validation of vocal services. Very recently, we also started to study the impact of our test generation techniques in the domain of network security. More specifically, we believe that testing that a network or information system meets its security policy is a major concern, and complements other design and verification techniques.

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 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. Smart-card Applications

We have also applied our test generation techniques in the context of smart-card applications. Such applications are typically reactive as they describe interactions between a user, a terminal and a card. The number and complexity of such applications is increasing, with more and more services offered to users. The security of such applications is of primary interest for both users and providers and testing is one of the means to improve it.

4.5. Control-command Systems

The main application domain for controller synthesis 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.

5. Software

5.1. TGV

Participant: Thierry Jéron [contact].

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. The first version of TGV is protected by APP (Agence de Protection des Programmes) Number IDDN.FR.001.310012.00.R.P.1997.000.2090.

5.2. STG

Participants: Vlad Rusu [contact], 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 deposit at APP (IDDN.FR.001.510006.000.S.P.2004.000.10600).

A new version in OCaml has been developed in the last two 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 [contact].

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], [29]. 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 protected by APP (Agence de Protection des Programmes).

6. New Results

6.1. Verification

6.1.1. Characterization and Analysis of infinite systems

6.1.1.1. A game approach to determinize 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 [24], [13], 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: underapproximation, and combination of under- and over-approximations.

6.1.1.2. Probabilistic Regular Graphs

Participants: Nathalie Bertrand, Christophe Morvan.

Deterministic graph grammars generate regular graphs, that form a structural extension of configuration graphs of pushdown systems. In [12], we study a probabilistic extension of regular graphs obtained by labelling the terminal arcs of the graph grammars by probabilities. Stochastic properties of these graphs are expressed using PCTL, a probabilistic extension of computation tree logic. We present here an algorithm to perform approximate verification of PCTL formulae. Moreover, we prove that the exact model-checking problem for PCTL on probabilistic regular graphs is undecidable, unless restricting to qualitative properties. Our results generalise those of Esparza et al (2006), on probabilistic pushdown automata, using similar methods combined with graph grammars techniques.

6.1.1.3. Contextual graph grammars characterising Rational Graphs

Participant: Christophe Morvan.

Deterministic graph grammars generate a family of infinite graphs which characterise context-free (word) languages. The present work introduces a context-sensitive extension of these grammars. We prove that this extension characterises rational graphs (whose traces are context-sensitive languages). We illustrate that this extension is not straightforward: the most obvious context-sensitive graph rewriting systems generate non recursive infinite graphs [22].

6.1.2. Theorem Proving

6.1.2.1. Combining narrowing and theorem proving for rewriting-logic specifications

Participant: Vlad Rusu.

We present an approach for verifying dynamic systems specified in rewriting logic, a formal specification language implemented in the Maude system. Our approach is tailored for invariants, i.e., properties that hold on all states reachable from a given class of initial states. The approach consists in encoding invariance properties into inductive properties written in membership equational logic, a sublogic of rewriting logic also implemented in Maude. The invariants can then be verified using an inductive theorem prover available for membership equational logic, possibly in interaction with narrowing-based symbolic analysis tools for rewriting-logic specifications also available in the Maude environment.We show that it is possible, and useful, to automatically test invariants by symbolic analysis before interactively proving them [23].

6.1.2.2. Equational approximations for tree automata completion

Participant: Vlad Rusu.

In [10], we deal with the verification of safety properties of infinite-state systems modeled by term rewriting systems. An over-approximation of the set of reachable terms of a term rewriting system image is obtained by automatically constructing a finite tree automaton. The construction is parameterized by a set E of equations on terms, and we also show that the approximating automata recognize at most the set of image-reachable terms. Finally, we present some experiments carried out with the implementation of our algorithm. In particular, we show how some approximations from the literature can be defined using equational approximations.

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 [11], 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 solved in [13] thanks to an approximate determinization using a game approach. 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. More Testable Properties

Participants: Yliès Falcone, Thierry Jéron, Hervé Marchand.

We explore the set of testable properties within the Safety-Progress classification where testability means to establish by testing that a relation, between the tested system and the property under scrutiny, holds. We characterize testable properties wrt. several relations of interest. For each relation, we give a sufficient condition for a property to be testable. Then, we study and delineate, for each Safety-Progress class, the subset of testable properties and their corresponding test oracle producing verdicts for the possible test executions. Furthermore, we address automatic test generation for the proposed framework. Finally, a tool implementing the results has been developped [18], [26].

6.2.3. Automatic Test Generation for Data-Flow Reactive Systems with time constraints

Participant: Hervé Marchand.

In [21], we handle the problem of conformance testing for data-flow critical systems with time constraints.We present a formal model (Variable Driven Timed automata) adapted for such systems inspired from timed automata using variables as inputs and outputs, and clocks. In this model, we consider urgency and the possibility to fire several transitions instantaneously. We present a conformance relation for this model and we propose a test generation method using a test purpose approach, based on a region graph transformation of the specification.

6.2.4. Analysis of partially observed recursive discrete-event systems

Participants: Sébastien Chédor, 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 and to detect information flow in regular discrete-event systems.

6.2.5. 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.

6.3. Controller Synthesis

6.3.1. Ensuring Security Properties

6.3.1.1. Supervisory Control for Opacity

Participant: Hervé Marchand.

In the field of computer security, a problem that received little attention so far is the enforcement of confidentiality properties by supervisory control. Given a critical system G that may leak confidential information, the problem consists in designing a controller C, possibly disabling occurrences of a fixed subset of events of G, so that the closed-loop system G/C does not leak confidential information. We consider this problem in the case where G is a finite transition system with set of events Σ and an inquisitive user, called the adversary, observes a subset Σ_a of Σ. The confidential information is the fact (when it is true) that the trace of the execution of G on Σ^* belongs to a regular set $S \subseteq \Sigma^*$, called the secret. The secret S is said to be opaque w.r.t. G (resp. G/C) and Σ_a if the adversary cannot safely infer this fact from the trace of the execution of G (resp. G/C) on Σ_a^*. In the converse case, the secret can be disclosed. We present an effective algorithm for computing the most permissive controller C such that S is opaque w.r.t. G/C and Σ_a. This algorithm subsumes two earlier algorithms working under the strong assumption that the alphabet Σ_a of the adversary and the set of events that the controller can disable are comparable [8].

6.3.1.2. Various Notions of Opacity Verified and Enforced at Runtime

Participants: Yliès Falcone, Hervé Marchand.

In [27], we are interested in the validation of opacity where opacity means the impossibility for an attacker to retrieve the value of a secret in a system of interest. Roughly speaking, ensuring opacity provides confidentiality of a secret on the system that must not leak to an attacker. More specifically, we study how we can verify and enforce, at system runtime, several levels of opacity. Besides already considered notions of opacity, we also introduce a new one that provides a stronger level of confidentiality.

6.3.1.3. Supervisory Control for Modal Specifications of Services

Participant: Hervé Marchand.

In the service oriented architecture framework, a modal specification, as defined by Larsen et al, formalises how a service should interact with its environment. More precisely, a modal specification determines the events that the server may or must allow at each stage in an interactive session. Therefore, techniques to enforce a modal specification on a system would be useful for practical applications. In this work, we investigate the adaptation of the supervisory control theory of Ramadge and Wonham to enforce a modal specification (with final states marking the ends of the sessions) on a system modelled by a finite LTS. We prove that there exists at most one most permissive solution to this control problem. We also prove that this solution is regular and we present an algorithm for the effective computation of the corresponding controller [14].

6.3.2. Supervisory control for synchronous systems

6.3.2.1. Multicriteria optimal discrete controller synthesis for fault-tolerant real-time tasks
Participant: Hervé Marchand.

We propose a technique for discrete controller synthesis, with optimal synthesis on bounded paths, in order to model, design, and optimize fault-tolerant distributed systems, taking into account several criteria (e.g., the execution costs of the tasks and their quality of service). Different combinations are explored for multi-criteria optimization [16].

6.3.2.2. Contracts for Modular Discrete Controller Synthesis
Participant: Hervé Marchand.

We describe the extension of a reactive programming language with a behavioral contract construct. It is dedicated to the programming of reactive control of applications in embedded systems, and involves principles of the supervisory control of discrete event systems. Our contribution is in a language approach where modular discrete controller synthesis (DCS) is integrated, and it is concretized in the encapsulation of DCS into a compilation process. From transition system specifications of possible behaviors, DCS automatically produces controllers that make the controlled system satisfy the property given as objective. Our language features and compiling technique provide correctness-by-construction in that sense, and enhance reliability and verifiability. Our application domain is adaptive and reconfigurable systems: closed-loop adaptation mechanisms enable flexible execution of functionalities w.r.t. changing resource and environment conditions. Our language can serve programming such adaption controllers. This work particularly describes the compilation of the language. We present a method for the modular application of discrete controller synthesis on synchronous programs, and its integration in the BZR language. We consider structured programs, as a composition of nodes, and first apply DCS on particular nodes of the program, in order to reduce the complexity of the controller computation; then, we allow the abstraction of parts of the program for this computation; and finally, we show how to recompose the different controllers computed from different abstractions for their correct coexecution with the initial program. Our work is illustrated with examples, and we present quantitative results about its implementation [15].

7. Other Grants and Activities

7.1. National Grants & Contracts

7.1.1. RNTL 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 2010, the post-doc position of Puneet Bhateja and the internship of Amélie Stainer were funded by TestTec.

7.1.2. Action Incitative GIPSy: Games of Imperfect information for Privacy and Security.

Participant: Nathalie Bertrand.

The GIPSy "Action Incitative" is a one-year [2010] project funded by Rennes 1 University to develop emerging research themes. The goal of the project is to start studying games of imperfect information and logics for privacy and security in protocols. The participants are Sophie Pinchinat (leader, S4), Sébastien Gambs (ADEPT), Loïc Hélouët and Blaise Genest (DistribCom), and Nathalie Bertrand (Vertecs). To gather researchers interested in the topic, a workshop on Games, Logics and Security has been organized in November 2010.

7.2. European and International Grants

7.2.1. Artist Design Network of Excellence

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand, Vlad Rusu.

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. The objective is to address the growth in complexity of future embedded products while reducing time and cost to market. This requires methods allowing for early exploration and assessment of alternative design solutions as well as efficient methods for verifying final implementations. This calls for a range of model-based validation techniques ranging from simulation, testing, model-checking, compositional techniques, refinement as well as abstract interpretation. The challenge will be in designing scalable techniques allowing for efficient and accurate analysis of performance and dependability issues with respect to the various types of (quantitative) models considered. The activity brings together the leading teams in Europe in the area of model-based validation.

7.2.2. Combest. European Strep Project

Participant: Nathalie Bertrand.

We are partners of the Combest European Strep Project http://www.combest.eu/home/. The aim of this project is to provide a theoretical framework as well as implemented methods and tools for the component-based design of embedded systems. Our role in Combest is to work on timed components, and more precisely develop a theory around timed modal specifications.

7.2.3. PHC Procope PIPS: Partial Information Probabilistic Systems

Participant: Nathalie Bertrand.

The objective of this bilateral collaboration [2009-2010] with the group of Prof. Christel Baier in TU Dresden (Germany) is to study partially observable probabilistic systems. This year, Christel Baier and Clemens Dubslaff visited Rennes for 1 week and Nathalie Bertrand went to Dresden for 2 weeks.

7.2.4. 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, Thomas Brihaye together with Patricia Bouyer made a 1 week visit in Rennes and Nathalie Betrand visited Mons for 1 week. Moreover, Amélie Stainer, Nathalie Bertrand, Thomas Brihaye and Patricia Bouyer met for 3 days in Cachan.

7.2.5. DGRST-INRIA grant

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

This two years collaboration [2009-2010] with ENIS Sfax Tunisia (Maher Ben Jemaa and Moez Krichen) is targetted on testing embedded systems and adaptability (with the Paris project team). It is funded by an DGRST - INRIA grant which involves visits on both sides and scholarships for Tunisian students. M. Krichen visited Vertecs during one month in summer 2010, working on test generation from timed automata.

7.2.6. Associated team (Equipe Associée) TReaTiES

Participants: Nathalie Bertrand, Thierry Jéron, Hervé Marchand.

This associated team http://www.irisa.fr/vertecs/EA-Brazil09.html with the Federal University of Campina Grande (Prof. Patrícia D. L. Machado) and University Pernambuco (Prof. Augusto Sampaio) in Brazil started in 2009. The objective is to work on test case generation, selection and abstraction for embedded real-time systems. In 2010 we had the visit of Wilkerson Andrade in december, and Y. Falcone, T. Jéron and H. Marchand visited the Brazilian team in Natal with a defense of Sidney Nogueira's thesis proposal and in Campina Grande where a workshop took place.

7.3. Collaborations

7.3.1. Collaborations with other INRIA Project-teams

We collaborate with several Inria project-teams. We collaborate with the ESPRESSO EPI for the development of the SIGALI tool inside the Polychrony environment. With the POP ART and SARDE EPI on the use of the controller synthesis methodology for the control of control-command systems (e.g. robotic systems). With DISTRIBCOM on stochastic games with partial observation. With the S4 EPI on the use of control, game theory and diagnosis for test generation as well as on the study of timed modal specifications, in the context of the Combest grant and with the TRISKELL EPI (Benoit Combemale) on analysis of domain-specific modelling languages. With the VASY EPI on the use of CADP libraries in TGV and the distribution of TGV in the CADP toolbox.

7.3.2. Collaborations with French Research Groups outside INRIA

We collaborate with Verimag in Grenoble on automatic test generation. We also work in collaboration with the LSV Cachan on topological and probabilistic semantics for timed automata. With LURPA Cachan, LaBRI Bordeaux and I3S Nice we collaborate on testing control-command systems in the context of the RNTL TesTec grant.

7.3.3. International Collaborations

Université Libre Bruxelles in Belgium (Prof. Thierry Massart) on testing and control of symbolic transitions systems. Gabriel Kalyon visited us for 1 week in september and Hervé Marchand visited ULB for 3 weeks.

University of Oxford (Matthew Hague) on Probabilistic Higher order pushdown automata.

University of Kaiserslautern (Roland Meyer) on Petri nets.

Univ. Illinois at Urbana Champaign, USA (Prof. Grigore Rosu) and Univ. Iasi, Romania (Prof. Dorel Lucanu) on formally defining and veryfying domain-specific languages by means of rewriting techniques.

ETH Zurich (Marina Egea) on formal semantics conformance in model-driven engineering

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

8. Dissemination

8.1. University courses

Nathalie Bertrand gave a course on Advanced model-cheching in the VTS module of the Master 2 Recherche at Université Rennes 1 in Fall 2010. She taught Finite Automata and regular languages to students of ENS Ker Lann preparing the Agrégation de Mathématiques.

Christophe Morvan is teaching at the University of Marne La Vallée (192h/year).

Amélie Stainer taught 8h to students of ENS Ker Lann preparing the Agrégation de Mathématiques.

8.2. PhD Thesis and Trainees

Current PhD. thesis:

Sébastien Chédor: "*Verification and Test of systems modeled by regular graphs*", second year.

Amélie Stainer: "*Quantitative verification of timed automata*", first year.

Trainees 2009-2010:

Amélie Stainer: "*Test of timed automata*"

8.3. Scientific animation

Nathalie Bertrand was PC member of QAPL'10, EXPRESS'10 and WODES'10 worshops, and QEST'10 conference and SC member of the GIPSy Workshop. She was invited at Schloss Dagstuhl for the seminar on Quantitative models in January 2010.

Yliès Falcone was an editor of the Proceedings of the 1st International Conference on Runtime Verification. He was a reviewer for several conferences (HSCC'11, FASE'11, WODES'10, VECOS', TASE'10). He gave an invited talk "What can You Verify and Enforce at Runtime ?" at INRIA Grenoble and during the GDR GPL day. He also gave a tutorial "You should Better Enforce than Verify" during the RV'10 conference. Finally, he visited NASA JPL for one week.

Thierry Jéron was PC member of ICTSS'2010, ICFEM'10, a thematic track of QUATIC'2010 and SC member of Movep 2010. He was reviewer of the PhD defense of Marius Mikucionis (University of Aalborg, Denmark, June 2010) and of the thesis proposal of Sidney Nogueira (University of Pernambuco, Brazil, November 2010). He is member of the IFIP Working Group 10.2 on Embedded Systems.

Hervé Marchand is Associate Editor of the IEEE Transactions on Automatic Control journal and member of the IFAC Technical Committees (TC 1.3 on Discrete Event and Hybrid Systems). He was PC member of the ICINCO'10, Wodes'10 Conferences and IFAC Worl Congress 2011. He visited ULB (Bruxelles) for three weeks in April and November 2010. He was member of the PhD commitee of Gabriel Kalyon ULB, Bruxelles, November 2010) and co-advisor of this thesis.

Christophe Morvan was invited to give a seminar "On probabilistic regular graphs" at LSV, Cachan.

Vlad Rusu organized the EJCP (Ecole Jeunes Chercheurs en Programmation) in June 2010. He was on the committee of the PhD thesis of Jose Escobedo (University Evry Val d'Essonne) on Symbolic Test Case Generation for Testing Orchestrators in Context (November 2010). He gave an invited talk entitled "Embedding domain-specific languages in Maude specifications" at the Univ. Iasi, (Romania) in January 2010, at the 1st International Workshop on the K Framework (Nags Head, North Carolina, USA) in August 2010 and at CWI (Amsterdam) in October 2010. He is "Foreign Advisor" for PhD students at the University of Iasi (Romania).

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] J. DUBREIL, P. DARONDEAU, H. MARCHAND. *Supervisory Control for Opacity*, in "IEEE Transactions on Automatic Control", May 2010, vol. 55, n^{o} 5, p. 1089-1100 [*DOI :* 10.1109/TAC.2010.2042008], http://hal.inria.fr/inria-00483891.

[9] M. EGEA, V. RUSU. *Formal Executable Semantics for Conformance in the MDE Framework*, in "Innovations in Systems and Software Engineering", 2010, vol. 6, p. 73-81 [*DOI :* 10.1007/s11334-009-0108-1], http://hal.inria.fr/inria-00527502.

[10] T. GENET, V. RUSU. *Equational Approximations for Tree Automata Completion*, in "Journal of Symbolic Computation", May 2010, vol. 45, n^{o} 5, p. 574-597 [*DOI :* 10.1016/J.JSC.2010.01.009], http://hal.inria.fr/inria-00495405.

International Peer-Reviewed Conference/Proceedings

[11] 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)", March 2011.

[12] N. BERTRAND, C. MORVAN. *Probabilistic Regular Graphs*, in "Infinity (International Workshop on Verification of Infinite-State Systems)", Singapore, EPTCS, September 2010, vol. 39, p. 77-90, http://hal.inria.fr/inria-00525388.

[13] 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 Computation Structures (FOSSACS)", March 2011.

[14] P. DARONDEAU, J. DUBREIL, H. MARCHAND. *Supervisory Control for Modal Specifications of Services*, in "Workshop on Discrete Event Systems, WODES'10", Berlin, Germany, August 2010, p. 428-435, http://hal.inria.fr/inria-00510013.

[15] G. DELAVAL, H. MARCHAND, E. RUTTEN. *Contracts for Modular Discrete Controller Synthesis*, in "Conference on Languages, Compilers and Tools for Embedded Systems, LCTES 2010", Stockholm, Sweden, April 2010, p. 57-66 [*DOI :* 10.1145/1755888.1755898], http://hal.inria.fr/inria-00476910.

[16] E. DUMITRESCU, A. GIRAULT, H. MARCHAND, E. RUTTEN. *Multicriteria optimal discrete controller synthesis for fault-tolerant real-time tasks*, in "Workshop on Discrete Event Systems, WODES'10", Berlin, Germany, August 2010, p. 366-373, http://hal.inria.fr/inria-00510019.

[17] Y. FALCONE. *You should Better Enforce than Verify (Tutorial)*, in "RV'10: Proceedings of the 1st International Conference on Runtime Verification", Malta, Lecture Notes in Computer Science, November 2010, vol. 6418, p. 89–105, http://hal.inria.fr/hal-00523653.

[18] Y. FALCONE, J.-C. FERNANDEZ, T. JÉRON, H. MARCHAND, L. MOUNIER. *More Testable Properties*, in "22nd IFIP International Conference on Testing Software and Systems", Natal, Brazil, Lecture Notes in Computer Science, November 2010, vol. 6435, p. 30-46, http://hal.inria.fr/inria-00510018.

[19] Y. FALCONE, M. JABER. *Towards Automatic Integration of an Or-BAC Security Policy Using Aspects*, in "World Comp 2010: Software Engineering Research and Practice (SERP'10)", Las Vegas, USA, July 2010, http://hal.inria.fr/hal-00525490.

[20] A. GAMATIÉ, V. RUSU, E. RUTTEN. *Operational Semantics of the Marte Repetitive Structure Modeling Concepts for Data-Parallel Applications Design*, in "9th International Symposium on Parallel and Distributed Computing (ISPDC'10)", IEEE Computer Society Press, July 2010, p. 25-32, http://hal.inria.fr/inria-00522787.

[21] O. LANDRY NGUENA, H. MARCHAND, A. ROLLET. *Automatic Test Generation for Data-Flow Reactive Systems with time constraints (Short paper)*, in "22nd IFIP International Conference on Testing Software and Systems", Natal, Brazil, November 2010, p. 25-30, http://hal.inria.fr/inria-00530584.

[22] C. MORVAN. *Contextual graph grammars characterising Rational Graphs*, in "Non-Classical Models of Automata and Applications (NCMA)", Jena, Germany, August 2010, p. 141-153, http://hal.inria.fr/inria-00525409.

[23] V. RUSU. *Combining narrowing and theorem proving for rewriting-logic specifications*, in "4th International Conference on Tests and Proofs (Tap'10)", Lecture Notes in Computer Science, July 2010, vol. 6143, p. 135-150, http://hal.inria.fr/inria-00527864.

Research Reports

[24] N. BERTRAND, A. STAINER, T. JÉRON, M. KRICHEN. *A game approach to determinize timed automata*, INRIA, October 2010, n^{o} 7381, http://hal.inria.fr/inria-00524830.

[25] P. DARONDEAU, J. DUBREIL, H. MARCHAND. *Supervisory Control for Modal Specifications of Services*, INRIA, April 2010, n^{o} 7247, http://hal.inria.fr/inria-00472736.

[26] Y. FALCONE, J.-C. FERNANDEZ, T. JÉRON, H. MARCHAND, L. MOUNIER. *More Testable Properties*, INRIA, April 2010, n^{o} 7279, http://hal.inria.fr/hal-00497350.

[27] Y. FALCONE, H. MARCHAND. *Various Notions of Opacity Verified and Enforced at Runtime*, INRIA, August 2010, n^{o} 7349, http://hal.inria.fr/inria-00507143.

References in notes

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

[29] L. BESNARD, H. MARCHAND, E. 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.

[30] 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.

[31] ISO/IEC 9646. *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)*, in "International Standard ISO/IEC 9646-1/2/3", 1992.

[32] S. OWRE, J. RUSHBY, N. SHANKAR, F. VON HENKE. *Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS*, in "IEEE Transactions on Software Engineering", feb 1995, vol. 21, n^{o} 2, p. 107-125.

[33] C. PAULIN-MOHRING. *Le système Coq (Habilitation Thesis, in French)*, ENS Lyon, 1997.

[34] 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.

[35] 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.

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