INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team VerTeCs

Verification models and techniques applied to the Testing and Control of reactive Systems

Rennes - Bretagne Atlantique

THEME COM

Table of contents

1. Team

Head of project-team
Thierry Jéron [Research Director (DR) Inria, HdR]

Administrative assistants
Lydie Mabil [TR INRIA, (80%)]

Research Fellows (Inria)
Nathalie Bertrand [Research Associate (CR) Inria, since October 2007]
Hervé Marchand [Research Associate (CR) Inria]
Vlad Rusu [Research Associate (CR) Inria, HdR]
Florimond Ployette [Technical staff, IR (50% with ASCII)]

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

Ph. D. Students
Camille Constant [INRIA, ATER since October 2007]
Jéremy Dubreil [INRIA, since March 2006]
Hatem Hamdi [in collaboration with UNIVERSITY OF SFAX, since October 2005]
Tristan Le Gall [MENRT, ATER since October 2007]

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 of the year

Arrival of Nathalie Bertrand. N. Bertrand has been hired as Inria researcher in 2007 and arrived in October 1st. She comes with a strong background in formal verification, in particular on qualitative and quantitative verification on Markovian models.

End of ACI Potestat grant. This project started in 2004 and ended in september 2007 has been our first involvement in security. It allowed us to learn a lot in this domain, to make some contributions by adaptating some work in test generation, diagnosis and control, and to start a larger collaboration in the RNTL Politess grant.

3. Scientific Foundations

3.1. Underlying Models.

Keywords: *controllable/uncontrollable events*, *implicit transition relation*, *input/output events*, *labeled transition systems*, *symbolic*.

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 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

Most of our research, and in particular controller synthesis and conformance testing, relies on the ability to solve some verification problems. A large part of these problems reduces to reachability and coreachability problems 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.

Verification in its full generality consists in checking that a system, which is specified in a formal model, satisfies a required property. When the state space of the system is finite and not too large, verification can be carried out by graph algorithms (model-checking). For large or infinite state spaces, we can perform 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.

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 (and also by deductive methods).

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 [36].

Abstract Interpretation is a theory of approximate solving of fixpoint equations applied to program analysis. Most program analysis problems, among others reachability analysis, come down to solving a fixpoint equation

$$x = F(x), x \in C$$

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 exact computation of such an equation is generally not possible for undecidability (or complexity) reasons. 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. Those two principles are well illustrated by the Interval Analysis [35], which consists in associating to each numerical variable of a program an interval representing an (upper) set of reachable values:

1. One substitutes to the concrete domain $\wp(R^n)$ induced by the numerical variables the abstract domain $(I_R)^n$, where I_R denotes the set of intervals on real numbers; a set of values of a variable is then represented by the smallest interval containing it;
2. An iterative computation on this domain may not converge: it is indeed easy to generate an infinite sequence of intervals which is strictly growing. The "standard" widening operator extrapolates by $+\infty$ the upper bound of an interval if the upper bound does not stabilize within a given number of steps (and similarly for the lower bound).

In this example, the state space $\wp(R^n)$ 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.

Programs performing dynamic allocation of objects in memory have an even more complex state space. Solutions have been devised to represent in an approximate way the memory heap and pointers between memory cells by graphs (*shape analysis* [49], [48]). Values contained in memory cells are however generally ignored.

In the same way, programs with recursive procedure calls, parameter passing and local variables are more delicate to analyse with precision. The difficulty is to abstract the execution stacks which may have a complex structure, particularly when parameters passing by reference are allowed, as it induces aliasing on the stack [30].

3.2.2. Theorem Proving

For verification we also use theorem proving and more particularly the PVS [45] and COQ [46] 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.

3.3. Automatic Test Generation

In testing, we are mainly interested in conformance testing. Conformance testing 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.

Models: We use IOLTS (or IOSTS) as formal models for specifications, implementations, test purposes, and test cases. Most often, specifications are not directly given in such low-level models, but are written in higher-level specification languages (e.g. SDL, UML, Lotos). The tools associated with these languages often contain a simulation API that implements their semantics under the form of IOLTS. On the other hand, the IOSTS model is expressive enough to allow a direct representation of most constructs of the higher-level languages.

Conformance testing theory: We adapt a well established theory of conformance testing [51], which formally defines conformance as a relation between formal models of specifications and implementations. This conformance relation, called **ioco** is defined in terms of visible behaviors (called *suspension traces*) of the implementation I (denoted by $STraces(I)$) and those of the specification S (denoted by $STraces(S)$). 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. The conformance relation $ioco$ was originally written in [51] as follows:

$$I \; ioco \; S \triangleq \forall \sigma \in STraces(S), Out(I \; after \; \sigma) \subseteq Out(S \; after \; \sigma)$$

where $M \; after \; \sigma$ is the set of states where M can stay after the observation of the suspension trace σ, and $Out(M \; after \; \sigma)$ is the set of outputs and quiescence allowed by M in this set. 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

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

Intuitively, this means that 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. In fact $STraces(S).\Lambda_!^\delta \setminus STraces(S)$ characterizes finite unexpected behaviours. Thus conformance with respect to S is clearly a safety property of I which negation can be specified by a "non conformance" observer $A_{\neg ioco \; S}$ built from S and recognizing these unexpected behaviours. However, as I is a black box, one cannot check conformance exhaustively, but may only experiment I using test cases, expecting the detection of some non-conformances. In fact the non-conformance observer $A_{\neg ioco \; S}$ can also be thought as the *canonical tester* of S for $ioco$, i.e. the most general testing process of S for $ioco$. It then serves also 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. Usually 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. Based on these models, the execution semantics, and the conformance relation, one can then define required properties of test cases and test suites (sets of test cases). Typical properties are soundness (only non conformant implementations should be rejected by a test case) and exhaustiveness (every non conformant implementation may be rejected by a test case). Soundness is not difficult to obtain, but exhaustiveness is not possible in practice and one has to select test cases.

Test selection: in the literature, in particular in white box testing, test selection is often based on coverage of some criteria (state coverage, transition coverage, etc). But in practice, test cases are often associated with *test purposes* describing some particular behaviors targeted by a test case. We have developed test selection algorithms based on the formalization of these *test purposes*. 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). We note $ASTraces(S, TP)$ the suspension traces of these accepted runs. Now selection of test cases amounts at selecting these traces $ASTraces(S, TP)$, and then complement them with unspecified outputs leading to *Fail*. Alternatively, this can be seen as the computation of a sub-automaton of the canonical tester $A_{\neg ioco\ S}$ whose accepting traces are $ASTraces(S, TP)$ and failed traces are a subset of $[STraces(S).\Lambda^{\delta}_{!} \setminus STraces(S)]$. 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).

Test selection algorithms are based on the computation of the visible behaviors of the specification $STraces(S)$, involving the identification of quiescence (δ actions) followed by determinisation, the construction of a product between the specification and test purpose which accepted behavior is $ASTraces(TP)$, and finally the selection of these accepted behaviors. Selection can be seen 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. We have developed on-the-fly algorithms, which consist in performing a lazy exploration of the set of states that are reachable in both the specification and the test purpose [4]. This technique is implemented in the TGV tool (see 5.1). 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 [7]. This is a promising technique whose main 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 [47]. The models used are LTSs, say G, and the associated languages, say $\mathcal{L}(G)$, 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 [47]. 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 [47]. 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 exists 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 [47] 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 [47] 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.

Control of Structured Discrete Event System. In many applications and control problems, LTS are the starting point to model fragments of a large scale system, which usually consists of several composed and nested sub-systems. Knowing that the number of states of the global system grows exponentially with the number of parallel and nested sub-systems, we have been interested in designing algorithms that perform the controller synthesis phase by taking advantage of the structure of the plant without expanding the system [2].

Similarly, in order to take into account nested behaviors, some techniques based on model aggregation methods [52], [32] have been proposed to deal with hierarchical control problems. Another direction has been proposed in [31]. Brave and Heimann in [31] introduced Hierarchical State Machines which constitute a simplified version of the STATECHARTS. Compared to the classical state machines, they add concurrency and hierarchy features. Some other works dealing with control and hierarchy can be found in [37], [39]. This is the direction we have chosen in the VERTECS Team [3].

Optimal Control. We are also interested in the Optimal Control Problem. The purpose of optimal control is to study the behavioral properties of a system in order to generate a supervisor that constrains the system to a desired behavior according to quantitative and qualitative requirements. In this spirit, we have been working on the optimal scheduling of a system through a set of multiple goals that the system must visit one by one [40]. We have also extended the results of [50] to the case of partial observation in order to handle more realistic applications [41]. Symbolic algorithms have also be developped and implemented in Sigali [42]

4. Application Domains

4.1. Panorama

Keywords: *Telecommunication*, *control-command Systems*, *smart-cards*, *software embedded systems*, *transportation systems*.

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 [38], 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 composionality 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 (see 7.1). 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

Keywords: *Conformance Testing*, *IF*, *Lotos*, *SDL*, *TGV*, *TTCN*, *UML.*

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 same API is used for a connection with the UMLAUT validation framework of UML models. This version has been transfered in the SDL ObjectGéode toolset as part of the TestComposer tool. A new version of TGV has been adapted to a new API of the IF simulator (VERIMAG) allowing test generation from IF and UML models (via a compilation from UML to IF). This new version TGV-IF extends the previous one with new functionalities for coverage based test generation combined with test purposes based test generation. This year some CADP libraries used in TGV-IF have been replaced with STL libraries in order to gain some independency with respect to CADP and allow easier porting. 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. TGV-IF is currently being deposit at APP.

5.2. STG

Keywords: *Conformance Testing*, *Symbolic Testing*, *Symbolic Verification*.

Participants: Vlad Rusu [contact], Florimond Ployette, 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.

5.3. SIGALI

Keywords: *Controller Synthesis*, *symbolic techniques*, *verification*.

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).

5.4. Ctrl-S

Keywords: *Controller Synthesis*, *structured systems*.

Participant: Hervé Marchand [contact].

CTRL-S is a tool dedicated to the control and simulation of structured discrete event systems. CTRL-S is a graphical tool connected with Oris dedicatedto (1) of synchronous products of finite state machines, and (2) the integration of toolboxes that compute their controllers. It now encompasses the former tool Syntool that was developped in our team during the past years.

6. New Results

6.1. Controller Synthesis

Keywords: *Hierarchical models*, *controller synthesis methodology*, *symbolic methods*.

Participant: Hervé Marchand.

6.1.1. Efficient Modular Method for the Control of Concurrent Discrete Event Systems: A Language-Based Approach

For several years we have been interested in the control of Concurrent Discrete Event Systems defined by a collection of components that interact with each other. We investigate the computation of the supremal controllable language contained in the language of the specification. We make the use of a modular centralized approach and perform the control on some approximations of the plant derived from the behavior of each component. The behavior of these approximations is restricted so that they respect a new language property for discrete event systems called *partial controllability condition* that depends on the safety property. It is shown that, under some assumptions (the objectives have to be *locally consistent* [11]), the intersection of these "controlled approximations" corresponds to the supremal controllable language contained in the property with respect to the plant. This computation is performed without building the whole plant. Further, we relax the usual assumption that all shared events are controllable by introducing two new structural conditions relying on the global mutual controllability condition. The novel concept used as a sufficient structural condition is strong global mutual controllability. The main result uses a weaker condition called global mutual controllability together with local consistency of the specification. An example illustrates the approach. This work has been done in cooperation with Jan Komenda (Academy of Sciences, Brno, Czech Republic), Jan van Schuppen (CWI, The Netherlands) and Benoit Gaudin (UCD, Dublin) [12].

6.1.2. Optimal discrete controller synthesis for the modeling of fault-tolerant distributed systems

Embedded systems require safe design methods based on formal methods, as well as safe execution based on fault-tolerance techniques. This year, we propose a safe design method for safe execution systems: it uses optimal discrete controller synthesis (DCS) to generate a correct reconfiguring fault-tolerant system. The properties enforced concern consistent execution, functionality fulfillment (whatever the faults, under some failure hypothesis), and several optimizations (of the tasks' execution time). We propose an algorithm for optimal DCS on bounded paths. We propose model patterns for a set of periodic tasks with checkpoints, a set of distributed, heterogeneous and fail-silent processors, and an environment model that expresses the potential fault patterns. We describe an implementation of our method, using the Sigali symbolic DCS tool and Mode Automata.This work has been done in cooperation with Emil Dumitrescu, Alain Girault and Eric Rutten [18], [23], [17].

6.1.3. Ctrl-S Tool Development

This year, we have pursued, in collaboration with Sophie Pinchinat from the INRIA project S4 at IRISA, the development of the open platform, named CTRL-S, dedicated to (1) the simulation of synchronous products of finite state machines, and (2) the integration of toolboxes that compute their controllers. This development has started in 2005 as a demo for the 30th Birthday of IRISA. Programming tasks have been assigned to Samer Maroun, an MSc. student from "École Supérieure d'ingénieurs de Beyrouth" (Liban), and was supported by an INRIA INTERSHIP. We also pursued the integration of the tool syntool, by considering new controller synthesis algorithms. A generic 3D libraries of components has been developped allowing an easy devising of demonstrations[27].

6.2. Test Generation on Enumerative and Symbolic Models

Keywords: *symbolic transition systems*, *test generation*, *testing*, *transition systems*.

6.2.1. Integrating formal verification and conformance testing for reactive systems

Participants: Camille Constant, Thierry Jéron, Hervé Marchand, Vlad Rusu.

In this work we describe a methodology integrating verification and conformance testing. A specification of a system - an extended input-output automaton, which may be infinite-state - and a set of safety properties ("nothing bad ever happens") and possibility properties ("something good may happen") are assumed. The properties are first tentatively verified on the specification using automatic techniques based on approximated state-space exploration, which are sound, but, as a price to pay for automation, are not complete for the given class of properties. Because of this incompleteness and of state-space explosion, the verification may not succeed in proving or disproving the properties. However, even if verification did not succeed, the testing phase can proceed and provide useful information about the implementation. Test cases are automatically and symbolically generated from the specification and the properties, and are executed on a black-box implementation of the system. The test execution may detect violations of conformance between implementation and specification; in addition, it may detect violation/satisfaction of the properties by the implementation and by the specification. In this sense, testing completes verification. The approach is illustrated on simple examples and on a Bounded Retransmission Protocol [9]

6.2.2. Automatic test generation from interprocedural specifications

Participants: Camille Constant, Thierry Jéron.

This work is done in collaboration with Bertrand Jeannet (Inria Rhône-Alpes) and partly supported by France Telecom R & D. It adresses the generation of test cases for testing the conformance of a reactive black-box implementation with respect to its specification. We aim at extending the principles and algorithms of model-based testing for recursive interprocedural specifications that can be modeled by Push-Down Systems (PDS). Such specifications may be more compact than non-recursive ones and are more expressive. The generated test cases are selected according to a test purpose, a (set of) scenario of interest that one wants to observe during test execution. The test generation method we propose in this paper is based on program transformations and a coreachability analysis, which allows to decide whether and how the test purpose can still be satisfied. However, despite the possibility to perform an exact analysis, the inability of test cases to inspect their own stack prevents it from using fully the coreachability information. We discuss this partial observation problem, its consequences, and how to minimize its impact [15].

6.2.3. Application to security

Participants: Jérémy Dubreil, Hatem Hamdi, Thierry Jéron, Hervé Marchand, Vlad Rusu.

While a lot of work has been done on formal verification of security, in particular for cryptographic protocols, very little has been done on formal security testing. As a consequence, testing security often resort on the expert knwoledge and leads to ad hoc solutions. The general challenge is to study how formalization of security policies and information systems can help in automatically (or systematically) performing security testing. Several approaches are already investigated. In the context of ACI Potestat and RNRT Politess, we study how test generation techniques, and in particular test generation from safety properties [9], can be used for the automatic generation of possible attacks, attacks which should then be tested on the real system, due to the abstraction used in modelling and generation.

Finally, during our collaboration with University of Nijmegen we studied the combinaison of verification, testing and learning. The verification of cryptographic protocol specifications is an active research topic and has received much attention from the formal verification community. By contrast, the black-box testing of actual implementations of protocols, which is, arguably, as important as verification for ensuring the correct functioning of protocols in the "real" world, is not much studied. We propose an approach for checking secrecy and authenticity properties not only on protocol specifications, but also on black-box implementations. The approach is compositional and integrates ideas from verification, testing, and learning. It is illustrated on the Basic Access Control protocol implemented in biometric passports [21].

6.3. Verification and Abstract Interpretation

Keywords: *Abstract Interpretation*, *Communicating Finite State Machines*, *FIFO channels*, *Reachability Analysis*, *rewriting logic*, *theorem proving*.

6.3.1. Verification of Communication Protocols using Abstract Interpretation of FIFO queues

Participant: Tristan Le Gall.

The PhD thesis of Tristan Le Gall, co-supervised by Bertrand Jeannet (Pop-Art project team) is concerned by the verification of asynchronous systems communicating through FIFO channels and its applications. Communication protocols can be formally described by the Communicating Finite-State Machines (CFSM) model. This model is expressive, but not expressive enough to deal with complex protocols that involve structured messages encapsulating integers or lists of integers. That is the reason why we studied, this year, more complex models with an infinite alphabet of messages. We thus propose a new abstract domain for languages on infinite alphabets, which acts as a functor taking an abstract domain for a concrete alphabet and lift it to an abstract domain for words on this alphabet. The abstract representation is based on lattice automata, which are finite automata labeled by elements of an atomic lattice. We define a normal form, standard language operations and a widening operator for these automata. We apply this abstract lattice for the verification of symbolic communicating machines, and we discuss its usefulness for interprocedural analysis [20], [25].

6.3.2. Theorem proving for rewriting logic

Participant: Vlad Rusu.

This is common work with Manuel Clavel from the University of Madrid. In [26] we present an approach based on inductive theorem proving for verifying invariance properties of systems specified in Rewriting Logic (RL) [43], an executable specification language implemented, among others, in the Maude tool [33] . Since theorem proving is not directly available for rewriting logic, we define an encoding of rewriting logic into its Membership Equational (sub)Logic (MEL) [44]. Then, inductive theorem provers for MEL, such as the ITP tool [34], can be used for verifying the resulting membership equational logic specification, and, implicitly, for verifying the original RL specification. The approach is illustrated on the 2-process Bakery algorithm and also on the parametric, n-process version of the algorithm.

6.3.3. Probabilistic and Topological Semantics for Timed Automata

Participant: Nathalie Bertrand.

Like most models used in model-checking, timed automata are an idealized mathematical model used for representing systems with strong timing requirements. In such mathematical models, properties can be violated, due to unlikely (sequences of) events. In [14], we propose two new semantics for the satisfaction of LTL formulas, one based on probabilities, and the other one based on topology, to rule out these sequences. We prove that the two semantics are equivalent and lead to a PSPACE-Complete model-checking problem for LTL over finite executions.

6.4. Diagnosis and application to Security

Keywords: *Diagnosis*, *Discrete event system*, *Information flow*, *Security*.

6.4.1. Predictability of Sequence Patterns in Discrete Event Systems

Participants: Thierry Jéron, Hervé Marchand.

Following our preliminary results on diagnosis of discrete event systems, we studied in [24] the problem of predicting the occurrences of a pattern in a partially-observed discrete-event system. The system is modeled by a labeled transition system. The pattern is a set of event sequences modeled by a finite-state automaton. The occurrences of the pattern are predictable if it is possible to infer about any occurrence of the pattern before the pattern is completely executed by the system. An off-line algorithm to verify the property of predictability is presented. The verification is polynomial in the number of states of the system. An on-line algorithm to track the execution of the pattern during the operation of the system is also presented. This algorithm is based on the use of a diagnoser automaton. The results are illustrated using an example from computer systems. This work has been done in cooperation with S. Lafortune and S. Genc (University of Michigan, USA).

6.4.2. Construction of monitor for the supervision of security properties

Participants: Jérémy Dubreil, Thierry Jéron, Hervé Marchand.

Regarding security, besides our work on test generation for security properties, we have been interested in constructing monitors for the detection of confidential information flow in the context of partially observed discrete event systems modelled by finite labelled transitions systems. We focused on the case where the secret information is given as regular languages. First, we characterised the set of observations allowing an attacker to infer secret information. Further, based on the diagnosis of discrete event systems theory, we provided necessary and sufficient conditions under which detection and prediction of secret information flow can be ensured and construct a monitor allowing an administrator to detect it. We consider the general case where the attacker and the administrator have different partial views of the system [16].

7. Contracts and Grants with Industry

7.1. France Telecom R&D

Keywords: *test generation*, *testing*, *vocal phone services*.

Participants: Camille Constant, Thierry Jéron, Vlad Rusu.

The goal of this 3-year project (starting October 2004) is to build a platform for the formal validation of France Telecom's vocal phone services. Vocal services are based on speech recognition and synthesis algorithms, and they include automatic connection to the callee's phone number by pronouncing her name, or automatic pronounciation of the callee's name whose phone number was dialed in by the user. Here, we are not interested in validating the voice recognition/synthesis algorithms, but on the logic surrounding them. For example, the system may allow itself a certain number of attempts for recognizing a name, after which it switches to normal number-dialing mode, during which the user may choose to go back to voice-recognition mode by pronouncing a certain keyword. This logic may become quite intricate, and this complexity is multiplied by the number of clients that may be using the service at any given time. Its correctness has been identified by France Telecom as a key factor in the success of the deployment of voice-based systems. To validate them we are planning to apply a combination of formal verification and conformance testing techniques (cf. Section 6.2.1). In the context of Camille Constant's PhD, we also study test generation from models of programs with recursion (pushdown automata and extensions).

8. Other Grants and Activities

8.1. National Grants & Contracts

8.1.1. CNRS ACI Sécurité Potestat: Security Policies: Test Directed Analysis of Open Network Systems

Participants: Jérémy Dubreil, Thierry Jéron, Hervé Marchand, Vlad Rusu.

The POTESTAT project [2004-2007] (http://www-lsr.imag.fr/POTESTAT/) addresses the problem of testing security policies for open networked systems. It was a joint project of 5 teams in 3 laboratories (The Vasco team of LIG Grenoble, the DCS team of VERIMAG Grenoble and Distribcom, Lande and VerTeCs project-teams in INRIA Rennes.

In the framework of open service implementations, based on the interconnection of heterogeneous systems, the security managers lack of well-formalized analysis techniques. The security of such systems is therefore organized from pragmatic elements, based on well-known vulnerabilities and their associated solutions. It then remains to verify if such security policies are correctly and effectively implemented in the actual system. This is usually carried out by auditing the administrative procedures and the system configuration. Tests are then performed, for instance by probing, to check the presence of some particular vulnerabilities. Although some tools are already available for specific tests (like password crackers), there is no solution to analyse the whole system conformance with respect to a security policy. The initial approach to the problem was based on previous experience of the partners. We had experience on the use of formal models either to test the conformance of a distributed implementation to a specification (conformance testing for network protocols) or to analyse downloaded code (where testing can complement static analysis techniques). Based on this background, we proposed the two following different directions.

- Diagnosis. Whereas protocol testing is usually done through active tests, it turns out that passive testing techniques may be better related to the control of security requirements, through monitors or access controllers for instance [16].
- Generation of attacks. We investigated the use of test generation techniques for the generation of attacks from security policies (modeled as observers) and network models (an abstraction of the network behavior) [9].

8.1.2. RNRT POLITESS: Security Policies for Network Information Systems: Modeling, Deployment, Testing and Supervision

Participants: Jérémy Dubreil, Hatem Hamdi, Thierry Jéron, Hervé Marchand, Vlad Rusu.

The POLITESS project (http://www.rnrt-politess.info/) [2006-2008] involves GET (INT Evry and ENST Rennes), INPG-IMAG (LSR and VERIMAG laboratories), France Telecom R&D Caen, Leyrios Technologies, SAP Research, AQL Silicomp Rennes and Irisa. In a sense, this project is an extension of the Potestat project. The objective of the project is to study and provide methodological guidelines and software solutions for a formal approach to security of networks. This encompasses the specification of high level security policies with clear semantics, their deployment on the network in terms of security artifacts and the analysis of this deployment, testing and monitoring of security based on models of security policies and abstract models of networks. Our team is involved in several activities, in particular in modelling (defining adequate models for both the system and security policies), testing (modelling security testing, test generation/selection), supervision (intrusion detection, diagnosis) and case studies.

8.2. European and International Grants

8.2.1. ARTIST2 Network of Excellence

Participants: Thierry Jéron, Hervé Marchand, Vlad Rusu.

We are partners of the ARTIST2 Network of Excellence on Embedded Systems (http://www.artist-embedded.org/), involved in the Testing and Verification cluster with Brics in Aalborg (DK), University of Twente (NL), University of Liège (B), Uppsala (SE), VERIMAG Grenoble, ENS Cachan, LIAFA Paris, EPFL Lausanne (S). The aim of the cluster is to develop a theoretical foundation for real-time testing, real-time monitoring and optimal control, to design data structures and algorithms for quantitative analysis, and to apply testing and verification tools in industrial settings. For security, we plan to create a common semantic framework for describing security protocols including notion of "trust" and to develop tools and methods for the verification of security protocols. Test and verification tools developed by partners will be made available via a web-portal and with dedicated verification servers.

In ARTIST2, the main role of VERTECS is to integrate our research on testing and test generation based on symbolic transition systems with other works based on timed models.

8.3. Collaborations

8.3.1. Collaborations with other INRIA Project-teams

We collaborate with several Inria project-teams. We collaborate with the LANDE project-team in two ACI-Sécurité grants (V3F and POTESTAT). With ESPRESSO project-team for the development of the SIGALI tool inside the Polychrony environment. With the Pop-Art project-team on the use of the controller synthesis methodology for the control of control-command systems (e.g. robotic systems). With DISTRIBCOM on security testing in the context of Potestat and Politess grants. With the S4 project-team on the use of control, game theory and diagnosis for test generation. With the VASY project-team on the use of CADP libraries in TGV and the distribution of TGV in the CADP toolbox.

8.3.2. Collaborations with French Research Groups outside INRIA

Our main collaborations in France are with Vérimag. Beyond formalized collaborations, (ACI Potestat and APRON, RNRT Politess, Rex ARTIST2), we also collaborate on the connection of NBAC with Lurette for the analysis of Lustre programs, as well as the connection of SIGALI and Matou. We are also involved in several collaborations with LSR Imag (ACI Potestat and RNRT Politess).

8.3.3. International Collaborations

ENIS Sfax in Tunisia (M. Tahar Bhiri) on security testing. Thierry Jéron is co-supervisor of a PhD student Hatem Hamdi working on robustness and security testing.

Institute of Mathematics, Czech Academy of Sciences (Jan Komenda) on supervisory control of concurrent systems.

University of Madrid (Prof. Manuel Clavel) on theorem proving for rewriting logic.

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

9. Dissemination

9.1. University courses

C. Constant is teaching in License and Master in the Univeristy of Rennes 1 (96h/year).

J. Dubreil is teaching in INSA of Rennes (30h in 2006-2007), on the Scheme programming language

T. Jéron is teaching on Model-based Testing in Research Master of Computer Science at the University of Rennes 1.

T. Le Gall is teaching in License and Master in the Univeristy of Rennes 1 (96h/year)

9.2. PhD Thesis and Trainees

Current PhD. theses:

Camille Constant: "*Verification and symbolic test generation for reactive systems*", 3rd year,

Tristan Le Gall: "*Abstract lattice of fifo channels for verification and control synthesis*", 3rd year,

Hatem Hamdi: "*Testing of network security*", In collaboration with University of Sfax, 2nd year,

Jérémy Dubreil: "*Formal methods for testing and monitoring security of open networks*", 1st year.

Trainees 2005-2006:

Samer Maroun: "*A tool for the simulation of controlled discrete-Event systems* ", Internship-ESIB (Liban) (2,5 months).

9.3. Scientific animation

Thierry Jéron was PC member of Testcom/Fates'07 (Tallinn, Estonia) in June 2007 and Rosatea'07 (Boston) in July 2007. He is PC member of the forthcoming Testcom/Fates'08 (Osaka, Japan) June 2008, IEEE ICST 2008 (Lillehammer, Norway) in April 2008. Thierry Jéron gave a keynote. He is member of the steering committee and co-organiser of Movep 2008 (Orléans) in June 2008. He was reviewer and president of the PhD defense of Tarik Nahhal (Verimag, Grenoble, October 2007), reviewer of the PhD defense of Moez Krichen (Verimag, Grenoble, December 2007), and member of the PhD defense of Alexandra Desmoulins (Univ. Rennes 1, December 2007). He is member of the IFIP Working Group 10.2 on Embedded Systems (http://jerry.c-lab.de/ifip-wg-102/).

Hervé Marchand was PC member of the MSR'07 conference on modeling of reactive systems as well as of the ICINCO'07 Conference. He is member of the IFAC Technical Committees (TC 1.3 on Discrete Event and Hybrid Systems) for the 2005-2008 triennium. He is PC member of the forthcoming Wodes'08 and ICINCO'08 Conferences. He is member of the "Commissions de Spécialistes 27e section" at the University of Rennes 1.

Vlad Rusu Vlad Rusu was PC member of TestCom/Fates'07 (Tallinn, Estonia). He gave invited talks at LILFL (Lille) in May 2007, at LIFC (Besancon) in June 2007, and at LORIA (Nancy) in June 2007. He was a referee in the PhD committee of Delphine Longuet (Univ. Evry, Oct. 2007).

Tristan Le Gall was invited to give seminars on *Verification of communicating protocols / Abstract interpretation of regula languages* in VERIMAG Grenoble (June 2006) and Liafa Paris (October 2006).

Jérémy Dubreil gave a talk on "the construction of monitor for the supervision of security properties" during the summer school FOSAD 2007. He is president of the ADOC (PhD student association).

10. Bibliography

Major publications by the team in recent years

[1] C. CONSTANT, T. JÉRON, H. MARCHAND, V. RUSU. *2*, in "Combinaison entre vérification et test pour la validation de systèmes réactifs, Traité I2C. Systèmes Temps Réel: Techniques de Description et de Vérification - Théorie et Outils", vol. 1, Hermès Science, 2006, p. 59-88.

[2] B. GAUDIN, H. MARCHAND. *Modular Supervisory Control of a class of Concurrent Discrete Event Systems*, in "Workshop on Discrete Event Systems, WODES'04", September 2004, p. 181-186, http://www.irisa.fr/vertecs/Publis/Ps/2004-WODES-Language.pdf.

[3] B. GAUDIN, H. MARCHAND. *Supervisory Control of Product and Hierarchical Discrete Event Systems*, in "European Journal of Control", vol. 10, n^{o} 2, 2004.

[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)", vol. 6, October 2004.

[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", vol. 10, n^{o} 4, Octobre 2000, p. 347-368, http://www.irisa.fr/vertecs/Publis/Ps/J-DEDS.ps.gz.

[7] V. RUSU, L. DU BOUSQUET, T. JÉRON. *An approach to symbolic test generation*, in "International Conference on Integrating Formal Methods (IFM'00), Volume 1945 of LNCS", Springer Verlag, 2000, p. 338-357.

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

Year Publications

Articles in refereed journals and book chapters

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

[10] C. CONSTANT, T. JÉRON, H. MARCHAND, V. RUSU. *Validation of Reactive Systems*, in "Modeling and Verification of Real-TIME Systems – Formalisms and software Tools", to appear, chap. 2, Hermès Science, 2007, p. 57-79.

[11] 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", vol. 17, n^{o} 2, 2007, p. 179-209.

[12] J. KOMENDA, J. VAN SCHUPPEN, B. GAUDIN, H. MARCHAND. *Supervisory Control of Modular Systems with Global Specification Languages*, in "Automatica", to appear, 2007.

[13] S. PICKIN, C. JARD, T. JÉRON, J.-M. JÉZÉQUEL, Y. LE TRAON. *Test Synthesis from UML Models of Distributed Software*, in "IEEE Transactions on Software Engineering", vol. 33, n^{o} 4, April 2007, p. 252-269.

Publications in Conferences and Workshops

[14] C. BAIER, N. BERTRAND, P. BOUYER, T. BRIHAYE, M. GROESSER. *Probabilistic and Topological Semantics for Timed Automata*, in "Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07), New Delhi, India", Lecture Notes in Computer Science, To appear, Springer-Verlag, Dec 2007.

[15] C. CONSTANT, B. JEANNET, T. JÉRON. *Automatic test generation from interprocedural specifications*, in "TestCom/Fates07, Tallinn, Estonia", LNCS, June 2007.

[16] J. DUBREIL, T. JÉRON, H. MARCHAND. *Construction de moniteurs pour la surveillance de propriétés de sécurité*, in "6ème Colloque Francophone sur la Modélisation des Systèmes Réactifs, Lyon, France", October 2007.

[17] E. DUMITRESCU, A. GIRAULT, H. MARCHAND, E. RUTTEN. *Optimal discrete controller synthesis for the modeling of fault-tolerant distributed systems*, in "First IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07), Paris, France", June 2007.

[18] E. DUMITRESCU, A. GIRAULT, H. MARCHAND, E. RUTTEN. *Synthèse optimale de contrôleurs discrets et systèmes répartis tolérants aux fautes*, in "6ème Colloque Francophone sur la Modélisation des Systèmes Réactifs, Lyon, France", October 2007.

[19] B. JEANNET, T. JÉRON, V. RUSU. *Model-based test selection for infinite state reactive systems*, in "Formal Methods of Components and Objects – FMCO 2006, Amsterdam, Netherlands, Revised Lectures", F. DE BOER, M. M. BONSANGUE, S. GRAF, W.-P. DE ROEVER (editors), Lecture Notes in Computer Science, vol. 4709, Springer-Verlag, 2007, p. 47–69.

[20] T. LE GALL, B. JEANNET. *Lattice automata: a representation of languages over an infinite alphabet, and some applications to verification*, in "The 14th International Static Analysis Symposium, SAS 2007, Kongens Lyngby, Denmark", August 2007.

[21] M. OOSTDIJK, V. RUSU, J. TRETMANS, R. DE VRIES, T. WILLEMSE. *Integrating verification, testing, and learning for cryptographic protocols*, in "Integrated Formal Methods (IFM'07)", Lecture Notes in Computer Science, Springer Verlag, 2007.

Internal Reports

[22] C. CONSTANT, B. JEANNET, T. JÉRON. *Automatic Test Generation from Interprocedural Specifications*, Technical report, n^{o} 1835, IRISA, March 2007.

[23] E. DUMITRESCU, A. GIRAULT, H. MARCHAND, E. RUTTEN. *Optimal discrete controller synthesis for the modeling of fault-tolerant distributed systems*, Technical report, n^{o} 6137, INRIA, March 2007, http://hal.inria.fr/inria-00134550.

[24] T. JÉRON, H. MARCHAND, S. GENC, S. LAFORTUNE. *Predictability of Sequence Patterns in Discrete Event Systems*, Technical report, n^{o} 1834, IRISA, March 2007.

[25] T. LE GALL, B. JEANNET. *Analysis of Communicating Infinite State Machines using Lattice Automata*, Technical report, n^{o} 1839, IRISA, March 2007.

[26] V. RUSU, M. CLAVEL. *Theorem proving for Maude's Rewriting Logic*, Technical report, n^{o} 1873, Irisa, 2007.

Miscellaneous

[27] S. MAROUN. *A tool for the simulation of controlled discrete event systems*, September 2007, Internship INRIA trainee report.

[28] F. PLOYETTE, B. JEANNET, T. JÉRON. *Stg: a symbolic test generation tool for reactive systems*, June 2007, TESTCOM/FATES07 (Tool Paper).

References in notes

[29] L. BESNARD, H. MARCHAND, E. RUTTEN. *The Sigali Tool Box Environment*, July 2006, p. 465-466.

[30] F. BOURDONCLE. *Sémantique des langages impératifs d'ordre supérieur et interprétation abstraite*, Ph. D. Thesis, Ecole Polytechnique, Paris, 1992.

[31] Y. BRAVE, M. HEIMANN. *Control of Discrete Event Systems Modeled as hierarchical State Machines*, in "IEEE Transacations on Automatic Control", vol. 38, n^{o} 12, December 1993, p. 1803–1819.

[32] P. CAINES, V. GUPTA, G. SHEN. *The hierarchical control of ST-finite-state machines*, in "Systems and Control Letters", vol. 32, 1997, p. 185-192.

[33] M. CLAVEL, F. DURAN, S. EKER, P. LINCOLN, N. MARTÍ-OLIET, J. MESEGUER, C. TALCOTT. *All About Maude, A High-Performance Logical Framework*, Lecture Notes in Computer Science, vol. 4350, Springer, 2007.

[34] M. CLAVEL, M. PALOMINO, A. RIESCO. *Introducing the* ITP *Tool: a Tutorial.*, in "J. Universal Computer Science", vol. 12, n^{o} 11, 2006, p. 1618-1650.

[35] P. COUSOT, R. COUSOT. *Static determination of dynamic properties of programs*, in "2nd Int. Symp. on Programming", Dunod, Paris, 1976.

[36] 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.

[37] P. GOHARI-MOGHADAM, W.M. WONHAM. *A linguistic Framework for controller hierarchical DES*, in "4th International Workshop on Discrete Event Systems, Cagliari(Italy)", August 1998, p. 207-212.

[38] 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.

[39] R.J. LEDUC. *Hierarchical Interface Based Supervisory Control*, Ph. D. Thesis, Dept. of Elec. & Comp. Engrg., Univ. of Toronto, 2002.

[40] H. MARCHAND, O. BOIVINEAU, S. LAFORTUNE. *On the Synthesis of Optimal Schedulers in Discrete Event Control Problems with Multiple Goals*, in "SIAM Journal on Control and Optimization", vol. 39, n^{o} 2, 2000, p. 512-532.

[41] H. MARCHAND, O. BOIVINEAU, S. LAFORTUNE. *On Optimal Control of a class of partially-Observed Discrete Event Systems*, in "Automatica", vol. 38, n^{o} 11, October 2002, p. 1935-1943.

[42] H. MARCHAND, M. LE BORGNE. *On the Optimal Control of Polynomial Dynamical Systems over Z/pZ*, in "4th IEE International Workshop on Discrete Event Systems, Cagliari, Italie", August 1998, p. 385–390.

[43] N. MARTÍ-OLIET, J. MESEGUER. *Rewriting logic: roadmap and bibliography.*, in "Theor. Comput. Sci.", vol. 285, n^{o} 2, 2002, p. 121-154.

[44] J. MESEGUER. *Membership algebra as a logical framework for equational specification.*, in "WADT", F. PARISI-PRESICCE (editor), Lecture Notes in Computer Science, vol. 1376, Springer, 1997, p. 18-61.

[45] 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", vol. 21, n^{o} 2, feb 1995, p. 107-125.

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

[47] 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", vol. 77, n^{o} 1, 1989, p. 81-98.

[48] M. SAGIV, T. REPS, R. WILHELM. *Parametric shape analysis via 3-valued logic*, in "ACM Transactions on Programming Languages and Systems", vol. 24, n^{o} 3, 2002.

[49] M. SAGIV, T. REPS, R. WILHELM. *Solving shape-analysis problems in languages with destructive updating*, in "ACM Transactions on Programming Languages and Systems", vol. 20, n^{o} 1, 1998.

[50] R. SENGUPTA, S. LAFORTUNE. *An Optimal Control Theory for Discrete Event Systems*, in "SIAM Journal on Control and Optimization", vol. 36, n^{o} 2, march 1998.

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

[52] K. C. WONG, W. M. WONHAM. *Hierarchical Control of Discrete-Event Systems*, in "Discrete Event Dynamic Systems", vol. 6, 1996, p. 241-273.