INRIA

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

THEME COM

Table of contents

1. Team

Head of project-team

Thierry Jéron [CR INRIA]

Administrative assistants

Lydie Mabil [TR INRIA, January-May and October-December (80%)]
Florence Santoro [from June 2005 to September 2005]
Céline Ammoniaux [TR CNRS from October 2005 (50%)]

Research Fellows (Inria)

Bertrand Jeannet [CR]
Hervé Marchand [CR]
Vlad Rusu [CR]
Florimond Ployette [Technical staff, IR (50% with ASCII)]

Post-doctoral fellows

Benoît Gaudin [LECTURER U. RENNES 1, until September 2005]
Jérôme Leroux [Post-doc CNRS, until September 2005]

Ph. D. Students

Camille Constant [INRIA]
Hatem Hamdi [in collaboration with UNIVERSITY OF SFAX, since October 2005]
Tristan Le Gall [MENRT]
Valery Tschaen [PH.D. AND LECTURER U. RENNES 1 until September 2005]

2. Overall Objectives

2.1. Overall Objectives

The VERTECS team is focused on the reliability of reactive software using formal methods. By reactive software we mean software that continuously reacts with its environment. The environment can be a human user for a complete reactive system, or another software using the reactive software as a component. 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. Correctness is also essential for less critical applications, in particular for COTS components whose behavior should be trusted before integration in software systems.

For this, the VERTECS team promotes the use of formal methods, i.e. formal specification and mathematically founded analysis methods. During the analysis and design phases, correctness of specifications with respect to requirements or higher level specifications can be established by formal *verification*. Alternatively, *control* consists in forcing specifications to stay within desired behaviours by coupling them with a supervisor. During validation, *testing* can be used to check the conformance of implementations with respect to their specifications. *Test generation* is the process of automatically generating test cases from specifications.

More precisely, the aim of the VERTECS project is to improve the reliability of reactive systems by providing software engineers with methods and tools for automating the **verification**, the **test generation** and **controller synthesis** from formal specifications. We adapt or develop formal models for the description of testing and control artifacts, e.g. specifications, implementations, test cases, supervisors. We formally describe correctness relations (e.g. conformance or satisfaction). We also formally describe interaction semantics between testing artifacts. From these models, relations and interaction semantics, we develop algorithms for automatic test and controller synthesis that ensure desirable properties. We try to be as generic as possible in terms of models and techniques in order to cope with a wide range of specification languages and application domains. We implement prototype tools for distribution in the academic world, or for transfer to 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.

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

The techniques described above, which are dedicated to the analysis of LTS, are already mature. It seems natural to extend them to IOSTS or data-flow applications that manipulate variables taking their values into possibly infinite data domains.

The techniques we develop for test generation or controller synthesis require to solve 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* [47], [46]). 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 [32].

3.2.2. Theorem Proving

For verification we also use theorem proving and more particularly the PVS [43] and COQ [44] 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 too can 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 [49], 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 [49] 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

$$I\ ioco\ S \Leftrightarrow STraces(I) \cap STraces(S).\Lambda_!^\delta \subseteq STraces(S)$$

Intuitively, this says that suspension traces of I which are suspension traces of S prolongated by an output or quiescence, are still suspension traces of S. An alternative characterization of ioco is then

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

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, giving them the status of automata or observers accepting sequences of actions or visible behaviors (suspension traces), denoted $ASTraces(TP)$. Now selection of test cases amounts at selecting visible behaviors of the specification that are accepted by the test purpose i.e. $STraces(S) \cap ASTraces(TP)$, 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 $[STraces(S).\Lambda_!^\delta \setminus STraces(S)] \cap ASTraces(TP)$. 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 $STraces(S) \cap ASTraces(TP)$, and finally the selection of these accepted behaviors. Selection can be seen as a model-checking problem where one wants to identify states (and transition between them) that

are 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. This technique is implemented in the TGV tool (see 5.1). However, this enumerative technique suffers from some limitations.

More recently, we have explored symbolic test generation techniques for IOSTS specifications. 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 is 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. These techniques are implemented in the STG tool (see 5.3), 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 [45]. 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 [45]. 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 [45]. 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

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

where $\parallel$ 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.

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

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 [45] 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 [45] 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.

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 had to visit one by one [7]. We have also extended the results of [48] to the case of partial observation in order to handle more realistic applications [42].

Control of Hierarchical 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 [9]. In other words, given the modular structure of the system, it becomes of interest, for computational reasons, to be able to synthesize a supervisor on each sub-part of the system w.r.t. the specification and then to infer a global supervisor from the local ones.

In order to reduce the complexity of the supervisor synthesis phase, several approaches have been considered in the literature. Modular control [52] and modular plant [53] are natural ways to handle this problem. Similarly, in order to take into account nested behaviors, some techniques based on model aggregation methods [51], [34] have been proposed to deal with hierarchical control problems. Another direction has been proposed in [33]. Brave and Heimann in [33] 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 [38], [41]. This is the direction we have chosen in the VERTECS Team [9]. Details are given in Section 6.1.1.

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 [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 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 also have to 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 systems 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.*

Participants: Thierry Jéron [contact], Valéry Tschaen.

TGV (Test Generation with Verification technology) is a tool for test generation of conformance test suites from specifications of reactive systems [5]. 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. NBAC

Keywords: *Abstract Interpretration, Reactive Systems, boolean and numerical types, polyhedra*.

Participant: Bertrand Jeannet [contact].

NBAC is a verification/slicing tool developed in collaboration with Vérimag. This tool analyses synchronous and deterministic reactive systems containing combination of Boolean and numerical variables and continuously interacting with an external environment. Its input format is directly inspired by the low-level semantics of the LUSTRE dataflow synchronous language. Asynchronous and/or non-deterministic systems can be compiled in this model. The kind of analyses performed by NBAC are: reachability analysis from a set of initial states, which allows to compute invariants satisfied by the system; coreachability analysis from a set of final states, which allows to compute sets of states that may lead to a final state; and combination of the above. The result of an analysis is either a set of states together with a necessary condition on states and inputs to stay in this set during an execution, either a verdict of a verification problem. The tool is founded on the theory of abstract interpretation: sets of states are approximated by abstract values belonging to an abstract domain, on which fix-point computations are performed. The originality of NBAC resides in

- the use of a very general notion of control structure in order to very precisely tune the trade-off between precision and efficiency;
- the ability to dynamically refine the control structure, and to guide this refinement by the needs of the analysis;
- sophisticated methods for computing postconditions and preconditions of abstract values.

More recently, NBAC has been extended with auxiliary translation tools AUTO2NBAC and NBAC2AUTO. This allows to specify systems to be analyzed as a product of hybrid automata with constant differential inclusion (*e.g.*, $1 \leq \dot{x} + 2\dot{y} \leq 3$) and to get the result of the analysis on the product automaton.

5.3. STG

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

Participants: Vlad Rusu [contact], Florimond Ployette, Bertrand Jeannet, Thierry Jéron.

STG (Symbolic Test Generation) [2] 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 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 is now under development. This version will be 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).

5.4. 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 sets of solution, 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 [8]. 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).

5.5. SYNTOOL

Keywords: *Controller Synthesis, structured systems.*

Participants: Benoit Gaudin, Hervé Marchand [contact].

SYNTOOL is a tool dedicated to the control of structured discrete event systems. It implements the theory developed during the Ph.D. of Benoit Gaudin [3]. SYNTOOL has an API allowing the user to graphically describe the different LTSs modeling the plant, to perform some controller synthesis computations solving the forbidden state avoidance problem for structured systems, and finally to simulate the result (i.e. the behavior of the controlled system). This tool is currently under testing.

5.6. RAPTURE

Keywords: *Markov Decision Processes, Probabilistic verification.*

Participant: Bertrand Jeannet [contact].

RAPTURE is a verification tool developed jointly by BRICS and INRIA [40]. The tool is designed to verify reachability properties on Markov Decision Processes (MDP), also known as Probabilistic Transition Systems. This model can be viewed both as an extension to classical (finite-state) transition systems extended with probability distributions on successor states, or as an extension of Markov Chains with non-determinism. We have developed a simple automata language that allows to describe a set of processes communicating over a set of channels *à la* CSP. Processes can also manipulate local and global variables of finite type. Probabilistic reachability properties are specified by defining two sets of initial and final states together with a probability bound. The originality of the tool is to provide two reduction techniques that limit the state space explosion problem: automatic abstraction and refinement algorithms, and the so-called essential states reduction [37].

6. New Results

6.1. Controller Synthesis

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

6.1.1. Supervisory Control of Structured Discrete Event Systems

Participants: Benoit Gaudin, Hervé Marchand.

In many applications (as e.g. manufacturing systems, control-command systems, protocol networks, etc) and control problems, systems are also often modeled by several components acting concurrently. We are concerned with the control of a system where its construction is assumed not to be feasible (due to the state space explosion resulting from the composition), making the use of classical supervisory control methodologies impractical. Given a concurrent system and a *safety property, modeled as a language*, also called specification that have to be ensured on this system, we investigated in [18] and [22] the computation of the supremal controllable language contained in the expected language. We do not adopt the decentralized approach. Instead, we have chosen to use a modular centralized approach and to 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 either *locally consistent* w.r.t. the plant [18] or *locally controllable* w.r.t. the plant [22]), the intersection of these "controlled approximations" corresponds to the supremal controllable language contained in the specification with respect to the plant. This computation is performed without building the whole plant, hence avoiding the state space explosion induced by the concurrent nature of the plant. It is finally shown that the class of specifications on which our method can be applied strictly subsumes that considered in [50].

Meanwhile, we have also been interested in ensuring safety properties that are more related to the notion of states rather than to the notion of trajectories of the system (the mutual exclusion problem for example). For this class of problem, one of the main issue is the *state avoidance control problem*, i.e. the supervisor has to control the plant so that the controlled plant does not reach a set of forbidden states. Note that if one wants to used a language-based approach to encode this problem, as e.g. [50] or [18], then the obtained specification may be of the size of the global system itself. This leads us to develop techniques totally devoted to the state avoidance control problem. Assuming the system is modeled as concurrent FSMs, we thus provide algorithms that, based on a particular decomposition of the set of forbidden configurations, solve the control problem locally (i.e. on each component without computing the whole system) and produce a global supervisor ensuring the desired property [15]. We then provide sufficient conditions under which the obtained controlled system is non-blocking [17]. This kind of objectives may be useful to perform dynamic interactions between different parts of a system. Finally, we apply these results to the case of Hierarchical Finite State Machines [16].

6.1.2. Supervisory Control of Symbolic and Hybrid Transition systems

Participants: Tristan Le Gall, Bertrand Jeannet, Hervé Marchand.

This year, we have been interested in solving the safety controller synthesis problem for various models (from finite transition systems to hybrid systems). Within this framework, we have been mainly interested in an intermediate model: symbolic transition systems. Due to the infiniteness of the alphabet, we have chosen to redefine the concept of controllability by introducing the notion of dynamic uncontrollable transitions (the controllability status is carried on by the symbolic transitions by means of guards, instead of the events). We focus on *safety requirements*, modeled by observers that encode the negation of a safety property. We then defined synthesis algorithms based on abstract interpretation techniques so that we can ensure convergence of fixpoint computations in a finite number of steps [23]. We finally generalized our methodology to the control of hybrid systems, which gives an unified framework to the supervisory control problem for several classes of models [28].

6.2. Test Generation on Enumerative and Symbolic Models

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

6.2.1. Symbolic Test Generation and Selection

Participants: Camille Constant, Bertrand Jeannet, Thierry Jéron, Vlad Rusu.

We address here the generation of symbolic test cases for testing the conformance of a black-box implementation with respect to a specification. More specifically, the problem we consider is the selection of test cases according to a test purpose, which is here a set of scenarii of interest that one wants to observe during test execution. Because of the interactions that occur between the test case and the implementation, test execution can be seen as a game involving two players, in which the test case attempts to satisfy the test purpose (in addition to its role of detecting conformance errors).

Efficient solutions to this problem have been proposed in the past in the context of finite-state models, based on the use of fixpoint computations. In [20], we extend them in the context of infinite-state symbolic models (IOSTS), by showing how approximate fixpoint computations can be used in a conservative way. We also formalize a quality criterion for test cases, and we provide a result relating the quality of a generated test case to the approximations used in the selection algorithm.

Instead of considering the extension of the finite-state IOLTS model with variables, as in [20], one can also consider the extension of the IOLTS model with recursion, which corresponds to a pushdown system. A preliminary study was done in 2004 with the master thesis of Liva Randriamanohisoa. One of the main problems still to be solved is the determinisation of a pushdown system. This determinisation is a necessary operation for test generation, but is undecidable. To overcome this problem, we study context-free grammars in order to find a sub-class of these grammars that can be translated into a deterministic pushdown system. We are also working on the combination of IOSTS and pushdown systems, which gives the full expressiveness of a programming language.

6.2.2. From Safety Verification to Safety Testing

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

In this work, we present a combination of verification and conformance testing techniques for the formal validation of reactive systems. A formal specification of a system, which may be infinite-state, and a set of safety properties are assumed. Each property is verified on the specification using automatic techniques based on abstract interpretation, which are sound, but, as a price to pay for automation, are not necessarily complete. Next, for each property, a test case is automatically generated from the specification and the property, and is executed on a black-box implementation of the system to detect violations of the property by the implementation and non-conformances between implementation and specification. If the verification step did not conclude, the test execution may also detect violations of the property by the specification [26]. This work generalizes our previous works on test selection with test purposes, allowing selection based on safety properties (while test purposes describe reachability properties). Moreover generated test cases are decorated with new verdicts.

We have also extended the combination of verification and conformance testing techniques by considering more expressive properties, allowing a better and finer selection of test cases. These properties are represented by observers with internal actions (the same as the specification) that are unobservable actions. Unfortunately, compared to [26], without any assumption about a mapping between the internal actions of the specification and the ones of the implementation, we cannot verify whether the implementation violates these properties or not (internal actions of the implementation are not known). Nevertheless, we can still detect non-conformances between the implementation and the specification and violations of the property by the specification.

6.2.3. Conformance Testing and Run-time Verification

Participants: Sophie Quinton, Vlad Rusu.

The Master's thesis of Sophie Quinton consisted in comparing and combining run-time verification and conformance testing techniques. The basic idea of run-time verification is to execute a program together with a *monitor* that checks the validity of some properties on the program's behaviour; typically, the monitor is automatically synthesized from logical annotations written in the program's code. On the other hand, conformance testing consists in comparing the observable behaviour of a black-box implementation of a system with respect to that described by an operational specification; this comparison is typically performed by executing the implementation in parallel with *test cases* generated from the specification and test purposes that may be, e.g., reachability or safety properties. This produces *test verdicts* about the conformance between implementation and specification and about the satisfaction of the properties by the implementation.

Clearly, there are similarities between the two approaches: for example, monitors and test cases play the same role of observing the program's behaviour and checking whether it respects some required properties. There are also some differences: run-time verification typically checks one particular implementation, namely, the executable program obtained by compiling the given (annotated) source code. In contrast, in conformance testing the implementation is not necessarily derived from the specification - the only requirement is that both views of the system have the same interface. Moreover, a test case not only observes, but may also control the implementation in attempting to satisfy a given test purpose (reachability property), or to violate a given safety property.

We have shown that many errors that can be discovered by run-time verification can be formalised as non-conformances with respect to an adequately chosen specification.

We also propose a methodology for replacing runtime verification by conformance testing in situations where runtime verification cannot work, for example, when not all the code of functions constituting the program under test is available, but only the annotations for those functions are available.

6.3. Verification and Abstract Interpretation

Keywords: *Abstract Interpretation*, *Acceleration*, *Communicating Finite State Machines*, *Exact Widening* , *FIFO channels*, *Reachability Analysis*, *Shape Analysis*.

6.3.1. Abstract Lattices for the Analysis of Systems with Unbounded FIFO Channels

Participants: Bertrand Jeannet, Thierry Jéron, Tristan Le Gall.

The PhD thesis of Tristan Le Gall holds on the verification of asynchronous systems communicating through FIFO channels and applications of it. This year, we tackled the reachability analysis of finite-state systems communicating through unbounded FIFO channels, using a finite set of messages [27]. Instead of related works relying on exact acceleration techniques, that may non terminate, our approach is based on abstract interpretation. We proposed a set of abstract lattices based on regular languages for representing sets of possible configurations of FIFO queues. We first focused on the simple case of systems with only one queue. We then generalized our results to systems with several queues, for which we proposed both non-relational and relational abstract lattices (a relational lattice preserves relations between contents of different queues, while a non relational one does not). Our experiments show that our method is generally as precise as exact methods founded on acceleration techniques, while it is arguably simpler.

6.3.2. A Relational Abstraction for Functions

Participant: Bertrand Jeannet.

In [19] we are interested in abstracting sets of functions. Main applications where one needs to infer properties on functions are interprocedural analysis, analysis of higher-order programs, and shape analysis, where one may use functions to associate to memory records their field contents. In [19] we give an overview of existing methods, which are illustrated with applications to shape analysis, and we formalize a new family of relational abstract domains that allows sets of functions to be abstracted more precisely than with known approaches, while being still machine-representable.

6.3.3. Flat Acceleration in Symbolic Model Checking

Participant: Jérôme Leroux.

Symbolic model checking provides partially effective verification procedures that can handle systems with an infinite state space. So-called "acceleration techniques" enhance the convergence of fixpoint computations by computing the transitive closure of some transitions. We have developed a new framework for symbolic model checking with accelerations. We have also proposed new symbolic algorithms using accelerations to compute reachability sets.

We have shown that flatness appears as a central notion in the verification of counter automata. A counter automaton is called flat when its control graph can be “replaced”, equivalently w.r.t. reachability, by another one with no nested loops.

From a practical view point, we have proved that flatness is a necessary and sufficient condition for termination of accelerated symbolic model checking, a generic semi-algorithmic technique implemented in successful tools like FAST, LASH or TREX.

From a theoretical view point, we have also proved that many known semilinear subclasses of counter automata are flat: reversal bounded counter machines, lossy vector addition systems with states, reversible Petri nets, persistent and conflict-free Petri nets, etc. Hence, for these subclasses, the semilinear reachability set can be computed using a *uniform* accelerated symbolic procedure (whereas previous algorithms were specifically designed for each subclass) [29], [14], [25], [24].

6.4. Transversal Results

Keywords: *Data Flow Analysis*, *Determinisation*, *Diagnosis*, *Fault Model*, *Logic*, *Proof*, *Symbolic Transition Systems*, *Type Theory*.

6.4.1. Determinisation of Symbolic Automata

Participants: Thierry Jeron, Hervé Marchand, Vlad Rusu.

Determinisation of symbolic transition systems is a crucial problem for the verification of properties on external traces, test generation, and diagnosis based on these models. However STSs cannot be determinised in general. In this work, we define a symbolic determinisation procedure for a class of STS. The class consists of extended automata operating on symbolic variables and synchronizing with the environment (a simplified version of STS). The subclass of extended automata for which the procedure terminates is characterized. This class named “Bounded Lookahead Automata” corresponds to systems where non-deterministic choices can be find out after the observation of a bounded number of actions. Decidable sufficient conditions for checking termination are also given. This work is under submission.

6.4.2. Supervision Patterns for the Diagnosis of Discrete Event Systems

Participants: Thierry Jéron, Hervé Marchand.

In this work, we are interested in the diagnosis of finite transition systems. We propose a model of supervision patterns corresponding to reachability properties (i.e. violation of a safety property). This allows to generalize the properties to be diagnosed and to render them independent of the description of the system. We thus deduce techniques for the verification of diagnosability and the construction of a diagnoser based on

standard operations on transition systems. We show that these techniques are general enough to express and solve in a unified way a broad class of diagnosis problems found in the literature, e.g. diagnosing permanent faults, multiple faults, fault sequences and some problems of intermittent faults [21]. This work has been done in cooperation with Marie-Odile Cordier (Dream project-team) and Sophie Pinchinat (S4 project-team).

Our aim is now to extend these results to infinite state systems and to apply these techniques to the automatic generation of passive testers (intruder detection systems) in order to test on-line whether an implementation respects a given security policy.

6.4.3. Extracting a Data Flow Analyser in Constructive Logic

Participant: Vlad Rusu.

This work has been done in cooperation with David Cachera, Thomas Jensen, and David Pichardie from the Lande project-team of Irisa. A constraint-based data flow analysis is formalized in the specification language of the Coq proof assistant. This involves defining a dependent type of lattices together with a library of lattice functors for modular construction of complex abstract domains. Constraints are represented in a way that allows for both efficient constraint resolution and correctness proof of the analysis with respect to an operational semantics. The proof of existence of a solution to the constraints is constructive which means that the extraction mechanism of Coq provides a provably correct data flow analyser in Ocaml from the proof. The library of lattices and the representation of constraints are defined in an analysis-independent fashion that provides a basis for a generic framework for proving and extracting static analysers in Coq [12].

6.4.4. Defining and Reasoning About General Recursive Functions in Type Theory

Participant: Vlad Rusu.

This is common work with David Pichardie, formerly in the Lande project of Irisa, currently in the Everest project-team at Sophia Antipolis.

In [30] we describe practical method for defining and proving properties of general (i.e., not necessarily structural) recursive functions in proof assistants based on type theory. The idea is to define the *graph* of the intended function as an inductive relation, and to prove that the relation actually represents a function, which is by construction the function that we are trying to define. Then, we generate *induction principles* for proving other properties of the function.

The approach has been experimented in the Coq proof assistant, but should work in like-minded proof assistants as well. It allows for functions with mutual recursive calls, nested recursive calls, and works also for the standard encoding of partial functions using total functions over a dependent type that restricts the original function's domain.

We present simple examples and report on a larger case study (sets of integers represented as ordered lists of intervals) that we have conducted in the context of certified static analyses[12].

In ongoing independent work, yet unpublished, Barthe and Forest from the Everest project at Inria Sophia Antipolis are developing an approach for synthesizing a recursive function from an arbitrary inductive relation. We have submitted a paper to FLOPS'06 (Int. Symposium on Functional and Logic Programming) on a general framework that encompasses both our approaches, as well as on an implementation of these techniques in a prototype tool within Coq.

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 6.2.2).

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: Thierry Jéron, Hervé Marchand, Vlad Rusu.

The Potestat project (http://www-lsr.imag.fr/POTESTAT/) [2004-2006] involves LSR-IMAG Grenoble, Verimag Grenoble and Lande and Vertecs project teams in Irisa.

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. This lack may be explained by several factors. First, there is currently no complete study about the formal modeling of a security policy, even if some particular aspects have been more thoroughly studied. Furthermore, verification based researches about security usually concerned more precise elements, like cryptographic protocols or code analysis. Finally, most of these works are dedicated to a priori verification of the coherency of security policies before their implementation. We are concerned here by the conformance of a system configuration with respect to a given policy. In the framework of the POTESTAT project we plan to tackle this problem according to the following items:

- Formal modeling of security policies, allowing a test directed analysis.
- Definition of a conformance notion between a system configuration and some security policies elements. The goal is to obtain a test theory similar to the one existing in the protocol testing area (like the Z.500 norm).
- Definition of methods to test this conformance notion, including the testability problems, the environment of execution, code analysis and test selection.

A long-term of this project is to offer some tools allowing security managers to model information flow, network elements (protocols, node types and their associated security policy, etc) to better describe the security policy for conformance testing and to provide some practical tools to perform coherency verification and vulnerabilities detection.

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

Participants: Thierry Jéron, Hervé Marchand.

The POLITESS project has just been accepted as an RNRT project. Partners of the project are GET (INT Evry and ENST Rennes), INPG-IMAG (LSR and Verimag laboratories), France Telecom R&D Caen, Leyrios Technologies, SAP Research, AQL Silicomp 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.

8.1.3. CNRS ACI Sécurité APRON: Analysis of Numerical Programs

Participant: Bertrand Jeannet.

The APRON (Analyse de PROgrammes Numériques) project (http://www.cri.ensmp.fr/apron/) [2004-2006] involves ENSMP, LIENS-ENS, LIX-Polytechnique, VERIMAG and Vertecs-Irisa.

The goal is to develop methods and tools to analyse statically embedded software with high-level of criticity for which the detection of errors at run-time is unacceptable for safety or security reasons. Such safety and security software is found in the context of transportation, automotive, avionics, space, industrial process control and supervision, etc. One characteristics of such software is that it is based on physical models whence involve a lot of numerical computations. Moreover, *counters* play an important role in the control of reactive programs (e.g., delay counting in synchronous programming). Critical properties depending on these counters are generally outside the scope of model-checking approaches, while being simple enough to be accurately analysed by more sophisticated numerical analyses.

The goal of the project is the static analysis of large specifications (e.g. à la LUSTRE) and corresponding programs (e.g. of 100 to 500 000 LOCs of C), made of thousands of procedures, involving a lot of numerical floating-point computations, as well as boolean and counter-based control in order to verify critical properties (including the detection of possible runtime errors), and to help in automatically locating the origin of critical property potential violation.

An example of such critical properties, as found in control/command programs, is of the form “under a condition holding on boolean and numerical variables for some time, the program must imperatively establish a given boolean and/or numerical property, in given bounded delay”.

Vertecs contributes to the following topics within the APRON project:

- The design and implementation of a common interface to several abstraction libraries (intervals, linear equalities, octagons, polyhedra, ...and their combination).
- The study of adaptative techniques for adjusting the trade-off between the efficiency and the precision of analyses, among other dynamic partitioning techniques [39]. Results have already been obtained in the intraprocedural case, but to a less extend in the interprocedural case.

Vertecs focuses mainly on LUSTRE specifications and provides with the NBAC tool one of the main experimental platforms of the project for the verification of critical properties on such specifications.

In 2005, most of the effort of Vertecs was spent on the design and implementation of the common interface, a task which should be soon completed.

8.1.4. CNRS ACI Sécurité V3F: Validation and Verification of Programs with Floating Point Numbers

Participants: Bertrand Jeannet, Thierry Jéron, Jérôme Leroux.

V3F (http://lifc.univ-fcomte.fr/~v3f/)[2003-2005] is a project involving LIFC Besançon, Inria-I3S Nice, LIST-CEA Saclay and project teams Lande and Vertecs in Irisa. The goal of this project is to provide tools to support the verification and validation process of programs with floating-point numbers. More precisely, project V3F investigates techniques to check that a program satisfies the calculations hypothesis on the real numbers that have been done during the modeling step. The underlying technology will be based on constraint programming. Constraints solving techniques have been successfully used during the last years for automatic test data generation, model-checking and static analysis. However in all these applications, the domains of the constraints were restricted either to finite subsets of the integers, rational numbers or intervals of real numbers. Hence, the investigation of solving techniques for constraint systems over floating-point numbers is an essential issue for handling problems over the floats.

The results obtained in the course of the project V3F are a clean design of constraint solving techniques over floating-point number, and a study of the capabilities of these techniques in the software validation and verification process. An open and generic prototype of a constraint solver over the floats was developped. We also paid attention on the integration of floats into various formal notations (e.g., B, Lustre, UML/OCL) to allow an effective use of the constraint solver in formal model verification, automatic test data generation (functional and structural) and static analysis.

Our contribution to this project is to precisely formalize a conformance testing theory for programs with floating point with respect to their specifications, and second, to describe test generation algorithms in this framework. We consider the IOSTS model for the specification and the test purpose. An important point is to obtain a computable conformance relation. The solution that we propose takes into account the unprecision of floating points computations w.r.t. real semantics by allowing a limited skew of floating points values in conformant traces. In order to be able to recognize conformant traces/execution during test execution, and to check that the allowed skew does not diverge, we use a projection technique that allows the tester to use safely the values emitted by the implementation for its own execution. A nice point of our approach is that we can fully reuse the test generation and selection techniques implemented in our STG tool, the only change being in the implementation of the test driver. This result has been presented in the last meeting but has not yet been published.

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.

This year we participated in a cluster meeting that took place in Oldenborg (Germany) in March 2005. A Summer School on Components & Modeling, Testing & Verification, and Static Analysis of Embedded Systems was organized in Nässlingen, Sweden in September by three clusters. T. Jéron gave a lecture on

"Testing and Model-checking" in this summer school. We also participated to the first review meeting of Artist2 in Grenoble in October. Finally, a Strep proposal on quantitative testing and verification, initiated by our cluster was submitted this year.

8.2.2. UIUC/CNRS/INRIA Collaboration

Participants: Vlad Rusu, Sophie Quinton, Thierry Jéron.

This grant involving three groups, IRISA/INRIA, Verimag (Stavros Tripakis), and University of Illinois at Urbana-Champaign (Grigore Roşu), is concerned with various aspects of runtime analysis of software systems, and aims both at advancing theoretical foundations and at developing and improving supporting tools and prototypes. Our activity this year has been on conformance testing and rewriting logic. Vlad Rusu and Sophie Quinton visited Urbana for 2 weeks (resp. 1 month) during Summer 2005. Grigore Roşu from Urbana visited us this summer and gave a talk on runtime analysis of distributed systems.

8.2.3. Bilateral CNRS/CONICET Collaboration

Participant: Bertrand Jeannet.

This bilateral collaboration grant, between France and Argentina, involves 4 teams: the team MOVE of LIF (Laboratoire d'Informatique Fondamentale) of Marseille (Peter Niebert) and the Vertecs project, on the French side, the university of Cordoba (Pedro d'Argenio) and the La Empesa University of Buenos Aires (Alfredo Olivero), on the Argentinian side.

The aim of this starting collaboration (august 2005- august 2007) is to make progress in the verification of probabilistic timed concurrent systems. The Vertecs project brings its expertise in algorithms and abstraction techniques implemented in the RAPTURE tool for Markov Decision Processes.

Our ambition is to progress in the following directions:

1. Apply the abstraction techniques implemented in the RAPTURE tool and the probabilistic partial order reduction techniques to *timed* probabilistic automata, instead of untimed systems only;
2. Extend the same techniques to the verification of untimed and timed temporal logical formula (expressed in the (timed) LTL logic);
3. Explore the applicability of program slicing and abstract interpretation techniques to quantitative model checking;
4. Explore the possibility to use these techniques for performance analysis;
5. Implement the fundamental results;
6. Analyse case studies in order to help to understand the performances of the tools.

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 DART 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. With the S4 project-team on the use of control, game theory and diagnosis for test generation. With the DREAM project-team on the diagnosis of discrete event systems. 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

University of Twente and Nijmegen in The Netherlands (E. Brinksma, J. Tretmans) on test generation (symbolic in particular) following the Van Gogh bilateral cooperation (1999-2001).

CNR Pisa in Italy (A. Bertolino) on using TGV for test generation for software architectures.

University of Wisconsin (T. Reps) on shape analysis. Bertrand Jeannet published a paper [19] with T. Reps and D. Gopan.

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

University Libre Bruxelles in Belgium on testing. Thierry Massart, Bram de Wachter and Cédric Meuter visited us in November. Thierry Jéron visited ULB for the defense of Bram de Watchter in December and gave a talk on Test Generation using Model Checking.

University Ottawa in Quebec (Guy-Vincent Jourdan) on security testing. G.-V. Jourdan visited us in summer 2005 (one week).

Institute of Mathematics, Czech Academy of Sciences on supervisory control of concurrent systems (with Jan Komenda). He visited us in November (one week). Benoit Gaudin and Hervé Marchand published a paper [22] with Jan Komenda.

9. Dissemination

9.1. University courses

C. Constant is teaching in INSA of Rennes (32h in 2004-2005), on constraint programming.

B. Gaudin was teaching at the University of Rennes 1 (92h/year).

B. Jeannet is teaching in the Master of Computer Science in Rennes, on abstract interpretation,

T. Jéron is teaching in Master of Computer Science at the University of Rennes 1 and in the engineering school EnstB in Rennes, on testing and test generation.

T. Le Gall is teaching in License in the Univeristy of Rennes 1 and in "Magistère Informatique-Télécom" in ENS Cachan-Bretagne (64h/year)

V. Rusu teaches in the Master of Computer Science in Rennes, on deductive verification methods.

V. Tschaen was teaching at the University of Rennes 1 (92h/year).

9.2. PhD Thesis and Trainees

Current PhD. theses:

Camille Constant: "*Verification and symbolic test generation for reactive systems*", 2nd year,

Tristan Le Gall: "*Abstract lattice of fifo channels for verification and control synthesis*", 2nd year,

Hatem Hamdi: "*Testing of network security*", In collaboration with University of Sfax, 1st year.

Trainees 2004-2005:

Sophie Quinton: "*Conformance testing and runtime verification*", Master student (6 months).

Hugo Métivier: "*representation of set of numerical values for verification purposes*", Master student (6 months).

Hatem Hamdi: "*Generation of test cases with TGV-Agedis*" in collaboration with ENIS Sfax, Master student (6 month).

Florence Charreteur: "*Test generation and execution with the STG Tool*" 4th year INSA Student, (2 months).

Trainees 2005-2006:

Jérémy Dubreil: "*Diagnosis for intrusion detection*", Master student, university of Uppsala and Enst Brest, (5 months).

9.3. Participation to Jurys, Committee

Thierry Jéron was jury of the following PhD defenses: Pierre Bontron (University Grenoble, March 2005, reviewer), Hélène Le Guen (University Rennes 1, June 2005), Cyril Pachon (University Grenoble, October 2005, reviewer), Gregory Lestienne (LRI Orsay, October 2005), Franck Lebeau (LIFC Besançon, December 2005, reviewer), Alban Gratien (University of Rennes, December 2005), Bram de Wachter (ULB Bruxelles, December 2005, private and public defenses). He served as reviewer of ARA SSIA and RNTL projects and Dutch Technology Foundation STW project (The Netherlands).

Bertrand Jeannet was jury of the PhD defense of David Merchat (University Grenoble 1, May 2005).

Hervé Marchand is member of the "Commissions de Spécialistes 27e section" at University of Rennes 1.

Camille Constant is member of the Ifsic Council as PhD representative and is Vice-president of the ADOC (association of PhD students).

9.4. Conferences, Seminars, Invited Talks

Bertrand Jeannet was the organizer of the thematic seminar “68 NQRT” (http://www.irisa.fr/NQRT/index.html) in Irisa.

Thierry Jéron is PC member of Fates’05 (Edinburgh, July 2005), Testcom 2005 (Montreal, June 2005), Tacas’05 (Edimburgh, April 2006). He is PC member of the forthcoming Testcom 2006 (New-York, May 2006), and PC member and Tool Chair of Tacas’06 (Vienna, March 06). He is SC of the Movep summer school (Bordeaux, June 2006). He is reviewer for Zentralblatt Math. Thierry Jéron gave a lecture on Testing and model-checking at the Artist 2 Summer School (Sept 2005). He was invited to give presentations in LIFC Besançon (Nov. 2006) and ULB Bruxelles (Dec. 2006) on Test generation using model-checking, and a presentation on Testing in DGA seminar “Enjeux opérationnels de la modélisation des systèmes complexes à logiciels prépondérants” in Toulouse (Nov. 2006).

Jérôme Leroux was invited to give a seminar on *“A polynomial time Presburger criterion and synthesis for number decision diagrams”* at LIAFA (Paris, France, Apr. 2005). He was also invited to give seminars on *“The convex hull of a Number Decision Diagram is an effectively computable polyhedron”* at LIPN (Villetaneuse, France, Mar. 2005) and at LORIA, (Nancy, France, Nov. 2004). He gave an invited talk on *“De l’automate binaire à la formule”* at “Journée des systèmes infinis 2005”, at ENS-Cachan, (Cachan, France, Dec. 2005).

Hervé Marchand is member of the Organizing Committee of ACSD’05 (St Malo, 06/05). He is PC member of MSR’05 conference on modeling of reactive systems (Autrans, 10/05). He is PC member of the forthcoming WODES’2006 and is member of the IFAC Technical Committees (TC 1.3 on Discrete Event and Hybrid Systems) for the 2005- 2008 triennium. He was invited to give a seminar on "Méthodes pour automatiser la génération de sytèmes fiables: Une approche par contrôle modulaire" at ENS Cachan, antenne Bretagne.

Vlad Rusu was invited to give a seminar on “combining conformance testing and formal verification” at "Formalisation des activités concurrentes" workshop (LAAS, Toulouse, march 2005). He also gave a seminar on the same topic at the Univ. Urbana Champaign (august 2005) and a lecture on the same topic at the Real-time summer school, Nancy, sept. 2005.

Tristan Le Gall is member of the Organizing Committee of MajecStic 2005 (Rennes, Nov. 2005).

10. Bibliography

Major publications by the team in recent years

[1] M. BOZGA, J.-C. FERNANDEZ, L. GHIRVU, C. JARD, T. JÉRON, A. KERBRAT, P. MOREL, L. MOUNIER. *Verification and test generation for the SSCOP protocol*, in "Journal of Science of Computer Programming, special issue on Formal Methods in Industry", vol. 36, n^o 1, Janvier 2000, p. 27-52.

[2] D. CLARKE, T. JÉRON, V. RUSU, E. ZINOVIEVA. *STG: a Symbolic Test Generation tool*, in "(Tool paper) Tools and Algorithms for the Construction and Analysis of Systems (TACAS’02), Volume 2280 of LNCS", Springer-Verlag, 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-TACAS.ps.gz.

[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] B. JEANNET, N. HALBWACHS, P. RAYMOND. *Dynamic Partitioning in Analyses of Numerical Properties*, in "Static Analysis Symposium, SAS’99, Volume 1694 of LNCS", 1999.

[5] T. JÉRON. *TGV: théorie, principes et algorithmes*, in "Techniques et Sciences Informatiques, numéro spécial Test de Logiciels", n° 21, 2002.

[6] T. JÉRON, P. MOREL. *Test generation derived from model-checking*, in "CAV'99, Trento (Italy), Volume 1633 of LNCS", N. HALBWACHS, D. PELED (editors). , Springer-Verlag, Juillet 1999, p. 108-122.

[7] 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° 2, 2000, p. 512-532.

[8] 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° 4, Octobre 2000, p. 347-368, http://www.irisa.fr/vertecs/Publis/Ps/J-DEDS.ps.gz.

[9] H. MARCHAND, B. GAUDIN. *Supervisory Control Problems of Hierarchical Finite State Machines*, in "41th IEEE Conference on Decision and Control, Las Vegas (USA)", December 2002, http://www.irisa.fr/vertecs/Publis/Ps/2002-CDC.ps.gz.

[10] 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.

[11] V. RUSU. *Compositional verification of an ATM protocol*, in "Formal Methods Europe (FME'03), Volume 2805 of LNCS", Springer-Verlag, 2001, p. 223-243.

Articles in refereed journals and book chapters

[12] D. CACHERA, T. JENSEN, D. PICHARDIE, V. RUSU. *Extracting a data flow analyser in constructive logic*, in "Theoretical Computer Science", vol. 342, n° 1, September 2005, p. 56–78.

[13] V. TSCHAEN. *Test generation algorithms based on preorder relations, Chapter 8 of Model-Based Testing of Reactive Systems, Volume 3472 of LNCS*, M. BROY, B. JONSSON, J.-P. KATOEN, M. LEUCKER, A. PRETSCHNER (editors). , Springer Verlag, 2005.

Publications in Conferences and Workshops

[14] S. BARDIN, A. FINKEL, J. LEROUX, P. SCHNOEBELEN. *Flat acceleration in symbolic model checking*, in "3rd Int. Symp. on Automated Technology for Verification and Analysis (ATVA'05), Taipei (Taiwan), Volume 3707 of LNCS", October 2005, p. 474–488.

[15] B. GAUDIN, H. MARCHAND. *Efficient Computation of supervisors for loosely synchronous Discrete Event Systems: A State-Based Approach*, in "6th IFAC World Congress, Prague (Czech Republic)", July 2005, http://www.irisa.fr/vertecs/Publis/Ps/2005-ifac.pdf.

[16] B. GAUDIN, H. MARCHAND. *Safety Control of Hierarchical Synchronous Discrete Event Systems: A State-Based Approach*, in "13th Mediterranean Conference on Control and Automation, Limassol (Cyprus)", June 2005, p. 889-895, http://www.irisa.fr/vertecs/Publis/Ps/2005_med.pdf.

[17] B. GAUDIN, H. MARCHAND. *Supervisory Control and Deadlock Avoidance Control Problem for Concurrent Discrete Event Systems*, in "44nd IEEE Conference on Decision and Control (CDC'05) and Control and European Control Conference ECC 2005, Seville (Spain)", December 2005.

[18] B. GAUDIN, H. MARCHAND. *Une approche modulaire pour le contrôle de systèmes à événements discrets concurrents*, in "5ième Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR'05), Autrans (France)", October 2005.

[19] B. JEANNET, D. GOPAN, T. REPS. *A Relational Abstraction for Functions*, in "The 12th International Static Analysis Symposium, SAS '05, Volume 3672 of LNCS", September 2005, p. 186–202.

[20] 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), Edinburgh (Scottland), Volume 3440 of LNCS", April 2005, http://www.irisa.fr/vertecs/Publis/Ps/tacas05.pdf.

[21] T. JERON, H. MARCHAND, M.-O. CORDIER. *Motifs de surveillance pour le diagnostic de systèmes à événements discrets finis*, in "15e congrès francophone Reconnaissance des Formes et Intelligence Artificielle (RFIA 2006), Tours (France)", January 2006.

[22] J. KOMENDA, H. VAN SCHUPPEN, B. GAUDIN, H. MARCHAND. *Modular supervisory control with general indecomposable specification languages*, in "44nd IEEE Conference on Decision and Control (CDC'05) and Control and European Control Conference ECC 2005, Seville (Spain)", December 2005.

[23] T. LE GALL, B. JEANNET, H. MARCHAND. *Supervisory Control of Infinite Symbolic Systems using Abstract Interpretation*, in "44nd IEEE Conference on Decision and Control (CDC'05) and Control and European Control Conference ECC 2005, Seville (Spain)", December 2005.

[24] J. LEROUX. *A Polynomial Time Presburger Criterion and Synthesis for Number Decision Diagrams*, in "20th IEEE Symp. Logic in Computer Science (LICS'2005), Chicago, USA", June 2005, http://www.irisa.fr/vertecs/Publis/Ps/2005-LICS.pdf.

[25] J. LEROUX, G. SUTRE. *Flat counter automata almost everywhere!*, in "3rd Int. Symp. on Automated Technology for Verification and Analysis (ATVA'05), Taipei (Taiwan), Volume 3707 of LNCS", October 2005, p. 489–503.

[26] V. RUSU, H. MARCHAND, T. JÉRON. *Automatic Verification and Conformance Testing for Validating Safety Properties of Reactive Systems*, in "Formal Methods 2005 (FM05), Volume 3582 of LNCS", J. FITZGERALD, A. TARLECKI, I. HAYES (editors). , Springer-Verlag, July 2005, http://www.irisa.fr/vertecs/Publis/Ps/final.pdf.

Internal Reports

[27] B. JEANNET, T. JÉRON, T. LE GALL. *Abstract lattices for the analysis of systems with unbounded FIFO channels*, Technical report, n° 1767, IRISA, Dec 2005.

[28] T. LE GALL, B. JEANNET, H. MARCHAND. *Contrôle de systèmes symboliques, discrets ou hybrides*, Technical report, n° 1683, IRISA, January 2005, http://www.irisa.fr/vertecs/Publis/Ps/PI-1683.ps.gz.

[29] J. LEROUX. *Structural Presburger-definable Digit Vector Automata*, Technical report, n° 1718, IRISA, May 2005, http://www.irisa.fr/vertecs/Publis/Ps/PI-1718.pdf.

[30] D. PICHARDIE, V. RUSU. *Defining and Reasoning About General Recursive Functions in Type Theory: a Practical Method*, Technical report, n° 1766, Irisa, November 2005.

Bibliography in notes

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

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

[34] 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.

[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. D'ARGENIO, B. JEANNET, H.E. JENSEN, K.G. LARSEN. *Reduction and Refinement Strategies for Probabilistic Analysis*, in "Process Algebra and Probabilistic Methods - Performance Modelling and Verification, PAPM-PROBMIV 2002, Copenhagen (Denmark), Volume 2399 of LNCS", July 2002, http://www.irisa.fr/prive/bjeannet/djjl02.ps.gz.

[38] 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.

[39] B. JEANNET. *Dynamic Partitioning In Linear Relation Analysis. Application To The Verification Of Reactive Systems*, in "Formal Methods in System Design", vol. 23, n° 1, July 2003, p. 5–37.

[40] B. JEANNET, P. D'ARGENIO, K.G. LARSEN. *RAPTURE: A tool for verifying Markov Decision Processes*, in "Tools Day, International Conference on Concurrency Theory, CONCUR'02, Brno (Czech Republic", August 2002, http://www.irisa.fr/prive/bjeannet/rapture02.ps.gz.

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

[42] H. MARCHAND, O. BOIVINEAU, S. LAFORTUNE. *On Optimal Control of a class of partially-Observed Discrete Event Systems*, in "Automatica", vol. 38, nº 11, October 2002, p. 1935-1943.

[43] 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, nº 2, feb 1995, p. 107-125.

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

[45] 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, nº 1, 1989, p. 81-98.

[46] M. SAGIV, T. REPS, R. WILHELM. *Parametric shape analysis via 3-valued logic*, in "ACM Transactions on Programming Languages and Systems", vol. 24, nº 3, 2002.

[47] 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, nº 1, 1998.

[48] R. SENGUPTA, S. LAFORTUNE. *An Optimal Control Theory for Discrete Event Systems*, in "SIAM Journal on Control and Optimization", vol. 36, nº 2, march 1998.

[49] J. TRETMANS. *Test Generation with Inputs, Outputs and Repetitive Quiescence.*, in "Software - Concepts and Tools", vol. 17, nº 3, 1996, p. 103-120.

[50] Y. WILLNER, M. HEYMANN. *Supervisory control of concurrent discrete-event systems*, in "International Journal of Control", vol. 54, nº 5, 1991, p. 1143-1169.

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

[52] W. M. WONHAM, P. J. RAMADGE. *Modular Supervisory Control of Discrete Event Systems*, in "Mathematics of Control Signals and Systems", vol. 1, 1988, p. 13–30.

[53] M.H. DEQUEIROZ, J.E.R. CURY. *Synthesis and implementation of local modular supervisory control for a manufacturing cell*, in "Proceedings of the 6th International Workshop on Discrete Event Systems, Zaragoza (Spain)", October 2002, p. 377-382.