INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On the Validation of Robotics Control Systems Part I: High Level Specification and Formal Verification

Bernard Espiau, Konstantin Kapellos, Muriel Jourdan, Daniel Simon

N° 2719

Novembre 1995

PROGRAMME 4

ISSN 0249-6399

On the Validation of Robotics Control Systems Part I: High Level Specification and Formal Verification

Bernard Espiau, Konstantin Kapellos*, Muriel Jourdan, Daniel Simon**

Programme 4 — Robotique, image et vision
Projet Bip

Rapport de recherche n° 2719 — Novembre 1995 — 32 pages

Abstract: This report presents an extensive work on the specification and the formal verification of complex applications in advanced robotics systems. In a first part, the need for such studies is presented, and a state-of-the-art in the field is given, evolving from the computer science area to the robotics one. Then, the key features used in the paper are presented. They are called the Robot Task and the Robot Procedure respectively, and are both integrated in the ORCCAD design environment. In the following, verification issues are described in depth, from the logical point of view as well as from the temporal one. They are illustrated by real examples, in which various properties are proved and abstract views are built. The conclusion gives an evaluation of the obtained results, expresses some requirements and draw guidelines for the future. The interest of hybrid systems is particularly emphasized.

Key-words: Robotics, Control, Formal Verification

(Résumé : tsvp)

*Projet Icare, Inria Sophia-Antipolis, supported by the Esprit Basic Research Action UNION
**Projet Icare, Inria Sophia-Antipolis

Unité de recherche INRIA Rhône-Alpes
46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1 (France)
Téléphone : (33) 76 57 47 77 – Télécopie : (33) 76 57 47 54

Sur la validation des systèmes de commande en robotique
1ère partie: Spécification de haut niveau et vérification formelle

Résumé : Ce rapport présente les résultats d'un ensemble de travaux menés en collaboration entre l'avant projet Bip de l'INRIA Rhône-Alpes et le projet Icare de l'INRIA Sophia Antipolis autour de la spécification et de la vèrification formelle d'applications complexes en robotique. Dans une première partie, nous présentons les motivations de ce travail en nous appuyant sur les besoins du domaine, et nous effectuons un état de l'art de la vérification en robotique à partir des résultats disponibles en informatique. Puis nous décrivons les deux entités clés de l'approche proposée, la tâche robot et la procédure robot, toutes deux situées par rapport à l'environnement ORCCAD. Les problèmes de vérification logique et temporelle sont ensuite abordés en détail et illustrés à travers des exemples réels pour lesquels diverses propriétés sont démontrées et des vues abstraites produites. La conclusion fournit une évaluation des résultats obtenus, exprime les principaux besoins d'amélioration et propose des axes de recherche futurs. L'accent est mis sur l'intérêt des systèmes hybrides pour les problèmes de robotique avancée.

Mots-clé : Robotique, Commande, Contrôle, Vérification,

1 Introduction

A mobile robot aimed to operate in an hazardous environment, like a long range AUV or a planetary rover, is a typical example of critical system. We mean here that, for such a system, like for a satellite, any repairing or recovery operation, even a mission reconfiguration, which would involve the intervention of a human operator is always costly, often difficult and sometimes impossible. This is why such systems should be at least provided with capacities of on-line adaption, like self replanning or sensor-based control. However, this is not sufficient and we have to be sure, as far as possible, that the system will behave correctly, *before launching*. More precisely, once a mission has been defined, we would like to verify that:

- its specifications are correct, i.e. that they correspond to the desired goals,
- its programming conforms to specifications,
- the constraints induced by real-time and implementation issues do not disturb its behavior.

Therefore, and assuming that the system hardware structure (mechanics, sensors, computer architecture) is given, this points out the necessity of validating all the algorithmic and software issues, from the point of view of their implementation as well as from that of their functionnalities. This mainly concerns two aspects, which should not be considered independently: the control issues, modelled through discrete- or continuous-time differential equations, and the logical ones, represented by discrete events. The questions raised by the first aspect (control issues) are addressed in a companion paper ([32]). Concerning the last case, we are for example led back to the need for verifying the *logical* aspects (absence of deadlocks, conformity of the results for various scenarii...), checking some *temporal* characteristics (absence of temporal deadlocks, values of lower and upper bounds on the duration of specific tasks...) and, possibly, studying *hybrid* models including other kinds of variables.

If we now adopt the user's point of view, it appears that his main interest is in the specification of complex missions or applications in an easy and safe way. For that purpose it is necessary to define properly what are the activities he should handle and how to *compose* them in order to meet his requirements. From the verification side, the compositionnality principles must preserve the coherence between underlying mathematical models, in order to be able to perform formal computations at any level. As an example, the use of a single synchronous reactive language as a target for automatic translation is a way of preserving a logical structure whatever the complexity of the application. A consequence of this point of view is that the basic entities have to be carefully studied and also that composition operators should have a proper semantics.

The aim of this paper is to describe how and in what framework we address this class of problems, to present the used concepts and tools and to comment various examples of logical and temporal verification taken from robotics applications.

It is organized as follows. In section 2, we give a brief state-of-the-art of formal verification principles, firstly in the computer science area, then by describing to some extent the use of such methods in robotics, through a review of the available literature. Then (section 3) we shortly present the software environment we use in all our applications. In the following

section (4), we give some details about the basic entity we defined, the *Robot Task*, and describe how to compose them in order to construct the so-called *Robot Procedures*. In the two next sections, verification issues are addressed, firstly from a logical point of view, secondly by including temporal aspects. The paper ends with a critical analysis of the work done and the proposition of some guidelines for the future.

2 Formal Verification: From Computer Science to Robotics

2.1 Reactive systems

The research area of program formal verification begins with the work of Floyd [4] and Hoare [5] for sequential programs. They established the basis of the so-called deductive proof systems area, wherein VDM-Z [3], PVS [1], COQ [2] are some famous representative tools. These theorem provers do not perform fully automatic verification, but they discharge the user from tedious tasks and allow him to concentrate on the essential structures of the proof.

These approaches inspired the community of reactive programming. In 1977, Pnueli [7] proposed the use of temporal logics as a basis for proving correctness of reactive systems. Temporal logic formulas express in a declarative way the set of programs which satisfies the property. The verification process consists in testing that the program belongs to this set. Behavioral methods are also proposed as a complementary approach [8]. They differ from the temporal logic based methods by the formalism which is used to express properties. Behavioral properties are expressed in the same formalism as the program to verify. Then, the verification process consists in comparing the program with the property by using, for instance, equivalence relations.

Axiomatizations of these two methods have been proposed, which made the formal verification of reactive programs theoretically possible. Nevertheless, in most of cases their automation is not realizable in practice. A solution consists in restricting the class of involved programs to the one of *finite state programs*. Automation becomes then possible owing to the use of finite state machine (i.e automata) to represent programs. For this reason, the definition and the efficient implementation of algorithms for automatic verification of finite state programs became an intense activity during the eighties for both methods [9, 10]. For temporal-logic based methods, decision procedures are known under the name of *model-checking* techniques.

These algorithms are implemented through several tools: Aldebaran [11], Auto [38], The Concurrency WorkBench [12] for the behavioral approach ; EMC [37] and Xesar [13] for the logic-based one. The major technical drawback of these verification methods, which are based on automata as programs models, is due to the model size. Indeed, this one increases exponentially with the number of parallel components of the program. Some strategies have nevertheless been studied to overcome these problems : reduction of the size of the stored models [15, 14] or symbolic methods of representation (mainly BDDs [16]) [17, 18].

All the above-mentioned tools take automata as inputs. However, it is obvious that a complex system could not be described directly by an automaton which is a too low-level formalism of description. For this reason, high-level languages, the semantics of which are expressed in terms of automata, have been designed:

- The first class is the one of *Synchronous* Languages among them ESTEREL, LUSTRE AND SIGNAL (a presentation of these three languages can be found in [19]) are the most famous. They are used to program reactive systems, since the automaton model of a program could be translated into executable code automatically. This kind of languages makes the assumption that the outputs are simultaneous to the inputs that cause them. They offer large programming environment with formal verification capabilities (for instance, owing to connections with the above-mentioned tools), as well as simulating and debugging features. Argos [21] is another synchronous language which has been recently designed and is inspired from the graphical formalism of Statecharts [20].
- A second category is the one of *Asynchronous* languages: Lotos [22], Electre [23], ..., which are much known for protocol specification or real-time tasks scheduling. They also offer programming environments with various possibilities.

2.2 Hybrid systems

Hybrid systems have been a topic of growing interest in the recent years. Their study comes from the fact that physical systems are never purely continuous-time ones nor only finite state machines, but really combine intimately discrete issues and continuous components. It is interesting to note that this idea was recognized by two distinct communities (computer science and automatic control), although the approaches were very different. Engineers from the automatic control side started from differential (continuous or discretized) dynamical systems, in which for example stability, convergence or robustness issues are of considerable importance, and tried to progressively introduce in such systems discrete events (switching modes, model changes...). Theoretical tools for studying such hybrid systems have been proposed (see [76] for example). Of course, the main complexity of these systems remains located in their dynamical part, and the associated finite state machines are often of a rather low size. On the contrary, the computer science community searchs to extend his results with (discrete) finite state machines to hybrid systems modeled as finite state automata, by adding progressively real-valued variables. A state represents a control location wherein variables change continuously with time according to evolution laws. In that case, the complexity of the system is due to the size of the underlying transition system, while the addressed dynamical issues remain rudimentary. A major challenge of the next decades is to make these two approaches meeting.

We detail now the state-of-the-art from the computer science point of view. It is obvious that it is not possible to take into account the whole class of possible hybrid systems for formal verification purposes. The first class of hybrid systems for which the main theoretical

results exist, so as formal verification tools, is timed-automata. In a timed-automaton, each variable x is a "clock" the rate of change with time of which is always 1. Model-checking algorithms have been studied in [30, 40]. Kronos [40] and HyTech [6] are tools which implement model-checking algorithms for timed-automata. After this first step about the formal analysis of hybrid systems, other subclasses have been identified: integration timed automata [25] where all the variables have rates 0 or 1, and linear hybrid automata where the variables change at constant rates, as summarized in the table below. The two above-mentioned tools have been extended in order to support the analysis of some of these subclasses [26]. Some approaches complementary to the model-checking one are also under study : approximate analysis [27],[28] and methods based on duration calculus [29].

Subclass	Clocks rate
Timed Automata	$\dot{x} = 1$
Integration Timed Automata	$\dot{x} \in \{0, 1\}$
Linear Hybrid Automata	$\dot{x} \in Z$

Few high-level languages have been designed to support hybrid models. Timed Argos is a temporal extension of the synchronous language Argos aimed to describe timed-automata. It is interfaced with the Kronos tool. Another extension of Argos, named Hybrid Argos, allows attributes to be attached with states and transitions. These attributes are propagated in the resulting automaton, and this is finally a way to describe hybrid systems. Electre ([23]) can also be used to model linear hybrid systems with specific properties. Specific verification methods have been developed for this subclass inside the Electre environment [24].

Finally, and as we will see later, the community of control of Discrete Event Systems also works on timed extensions of the theory of supervisory control.

2.3 Use in Robotics

Robotics, as an area integrating mechanics, computer science, automatic control and sensor technology is a domain where a natural reflex of the engineer is to *re-use* the basic techniques of the mother areas before developing specific approaches when needed. This is true also in the domain of formal verification. Indeed, it would be more exact to speak of formal verification *and* architecture design instead of verification alone, since these two issues are commonly strongly linked in robotics. In fact, considering robots as true hybrid (continuous/discrete -time) systems and exploiting this idea is a quite recent approach. This is why we will see that a review of the literature mainly exhausts the application of rather classical methods. Researchers in robotics have in fact followed two main paths:

- The first one comes more or less from questions raised by the so-called area of *Real-Time Artificial Intelligence*, and is also related to *Intelligent Control*. Here, the idea is to design functional architectures for the control of complex robotics systems that allow to address several levels, from execution to planning, while being able to cope with uncertainties, failures, etc... by on-line reasoning and reconfiguration. In that case,

verification aspects are usually focused on task coordination and mission reachability through reactive planning.

- A second point of view is based on the popular approach of DES (Discrete Event Systems). The underlying idea is to try to build for such systems a theory aimed to be the analogous of the dynamics system theory in continuous time, allowing in that way to address problems of design, optimization, identification, control, etc... In the wellknown approach of Ramadge and Wonham ([47, 48]; see also [56] for a similar approach), based on the theory of languages and automata, both the specification of a desired behavior and the modelling of a process have the forme of finite state machines. Then, owing to the definition of a set of controllable events, a dedicated supervisory controller can be synthetized. Besides, let us mention that the *(max,+)* approach developed in parallel ([69, 70]) is able to address rather similar problems within a nice mathematical framework which allows in supplement to consider quantitative issues or concepts from the automatic control area, like asymptotic behavior, Lyapunov functions, etc... Let us also note the existence of extensions to the RW's approach, like a timed one in [53] or a temporal logics- based one [55].

  Very well suited to the design of controllers for flexible manufacturing systems, this theory is also the most largely used in the robotics literature.

Let us now give an overview of the existing approaches in robotics by taking the point of view of the application areas. We will distinguish three main domains.

The first is the one of flexible manufacturing systems. Indeed, this is the area where most of the references can be found. Of course, it is not purely robotics, and the usual approach is the synthesis of a supervisory controller *à la* RW, therefore with no major concern about verification issues. Nevertheless, and to lie in the scope of the paper, let us mention that the timed extension already cited ([53]) was applied to a simple workcell where it could be proved that the duration of the nominal production cycle was less than n time units. Another exception is the work reported in [57, 58] in which the DES and the model checking approaches are blended in order to be able to formally verify the logics of the synthetized supervisor. In [57], this is applied to a legged robot and in [58] to a simple workcell. Let us end this short review by mentioning the work of [64] where verification is done for industrial robots.

A second domain of robotics in which formal verification is clearly necessary, although only sometimes used, is the one of mobile autonomous robots. For example, in [63], coloured Petri Nets are used for the design and the validation of the logical structure of a control architecture dedicated to a mobile robot with sensors. In [62], an architecture allowing to merge AI-type reasoning and real-time behavior through a graph-based representation is proposed. The approach integrates temporal aspects, and the structure of the transition system itself can be modified on-line when necessary; here, the application is a toy mobile robot equipped with a sonar. Again in the framework of reactive AI and planning, and even if not addressing exclusively the area of mobile robots, we find the approach of [59, 61]. There, a structure based on sensory-motor modules and modelled as a network of concurrent

communicating processes is defined. This allows to build and analyse plans including on-line reactivity. It is applied to visually-guided grasping in [60].

Another typical approach is reported in [52]: the DES/RW theory is used (mainly owing to its compositionnality properties) for synthetizing a controller handling a set of visual-based behaviors in a mobile robot. A very close application is addressed in [71], but with more verification aspects, as we will see later. Let us note that this question of compositionnality is strongly raised in the classical behavior-based models ([51]), where the need for a rich structure is obvious since it cannot exist implicitly in such approaches. That is the origin of the work reported in [77].

An important field where formal verification is required is the one of underwater vehicles or robots. A few works, like [67] or [73], verify explicitly some properties. However, the interested reader may also consult [75], where several approaches of control architecture are described, some of them allowing potentially to perform different types of verification.

Finally we cannot forget in this review the area of autonomous vehicles, which clearly demands high guarantees of safety and is therefore a critical application domain for verification methods. Let us for example mention that such requirements are expressed in [68], in the Automated Highway project. Some interesting results may also be found in automatic or aided car driving applications: for example, in [66], the correctness of the behavior of a decelerating vehicle is formally proved; in [74], as detailed later, some useful properties in automatic vehicle following are also derived.

3 ORCCAD: an Hybrid Systems-Oriented Architecture

ORCCAD ([45]) is a development environment for specification, validation by formal methods and by simulation, and implementation of robotic applications.

The formal definition of a robotic action is a key point in the ORCCAD framework. It is based on the following basic principles:

- in general, physical tasks to be achieved by robots can be stated as automatic control problems which can be efficiently solved in real-time by using adequate feedback control loops. In this framework, let us mention that the *Task-Function* approach ([46]) was specifically developed for robotic systems;
- the characterization of the physical action is not sufficient for fully defining a robotic action: starting and stopping times must be considered, as well as reactions to significant events observed during the task execution;
- since the overall performance of the system relies on the existence of efficient real-time mechanisms at the execution level, particular attention must be paid to their specification and their verification.

A robotic application should therefore handle all these aspects coherently. Its specification must be modular, structured and accessible to users with different expertize. The

end-user concerned with a particular application should be provided with high level formalisms allowing to focus on specification and verification issues; the *control systems engineer* needs an environment with efficient design, programming and simulation tools to express the control laws which then are encapsulated for the end-user.

In ORCCAD, two entities are defined in order to fulfill these requirements: the *Robot-task* (RT), representing an elementary robotic action, where automatic control aspects are predominant although coherently merged with behavioral ones, and the *Robot-procedure* (RP), the RTs are the basic element of which, and where only behavioral aspects are considered. Therefore, in ORCCAD, a robotic application is fully specified by identifying and specifying all the RTs needed by the application, and then by composing them hierarchically in the form of RPs. The ORCCAD system provides a graphical environment, a link with a simulation system SIMPARC ([31]) and the generation of realtime code running under VXWORKS.

4 Specification Using Robot Tasks and Robot Procedures

4.1 The Robo Task

The RT in ORCCAD is the minimal "granularity" seen by the end-user at the application level, and the maximum one considered by the control systems engineer at the control level. It is described in details in [45]. Formally, a RT is defined as the *parametrized specification of an elementary control law, i.e. the activation of a control scheme structurally invariant along the task duration, and of a logical behavior associated with a set of signals (events) which may occur just before, during and just after the task execution.*

From the control point of view, the specification of a RT requires to define the functions, the models and the parameters which appear in the analytical expression, in continuous time, of the control outputs to be applied to the actuators in order to perform the desired physical action. Besides, the specification of the logical behavior is obtained by setting the events to be considered and their treatment. These events and the associated processings are typed. We distinguish:

• the pre-conditions: their occurence is required for starting the servoing task. Synchronization pre-conditions are related to logical conditions, while measurement ones are usually obtained through sensors. A temporal watchdog can be associated with each measurement pre-condition.

• the exceptions: they are emitted during the execution of the servoing task and indicates a failure detection. Their processing is as follows:

- type 1: the reaction to the received exception is limited to the modification of the value of at least one parameter within the control scheme,

- type 2: the exception requires the activation of new RTs. The reaction consists in killing the current one and reporting the causes of the malfunction to the adequate level. The recovering process to activate is *known* and specified in the RP to which the RT contributes,

- type 3: the exception is considered as fatal. The overall application is stopped and the robotic system must be driven to a safe position.

• the post-conditions: often related to the environment, they are handled as conditions for a normal termination of the RT. Watchdogs can be associated to their waiting.

Finally, a RT is completely specified by setting *temporal properties*, i.e. discretization of the time, durations of the computations, communication and synchronization between the involved processes. This is done by implementing each RT in terms of communicating real-time computing "tasks", called *Module-tasks* (MTs). Most of them are periodic and perform the calculations involved in the computation of the control algorithm. Others, called *observers*, monitor conditions and are therefore used to handle preconditions, exceptions and post-conditions. The non-periodic *reactive* behavior of the RT is handled by a special MT called the Robot-task Automaton (RTA) which may be awakened by signals coming from the RT itself through the outputs of the observers and is used to link the RT to the input/output signals associated with the application level. Modelling and analysis of the Module-tasks network are presented in [32]. Examples of RTs designed and implemented under ORCCAD may be found in [45, 43, 36, 72].

4.2 The Robot Procedure

The characterization of the interface of a RT with its environment in a clear way, using typed input/output events, allows to compose them easily in order to construct more complex actions, the so called *Robot-procedures*. The aim in designing this entity is to be able to define a representation of a robotic action that could fit any abstraction level needed by the mission specification system. In its simplest expression, a RP coincides with a RT, while the most complex one might represent an overall mission. Briefly speaking, it specifies in a structured way a logical and temporal arrangement of RTs in order to achieve an objective in a context-dependent and reliable way, providing predefined corrective actions in the case of unsuccessful execution of RTs.

More formally a RP is the full specification of

• *a main program, (nominal execution of the action), composed of RTs, RPs and conditions,*

• *a set of triplets {exception event, processing, assertion}, which specifies the processing to be applied for handling the exception, and the information to transmit to the planning level (if any),*

• *a local behavior defining the logical coordination of the previously considered items.*

The composition of RTs and RPs in the main program is obtained through operators ([33]) which express sequence, parallelism, conditions, iterations, rendez-vous and various levels of preemption. The exception events in the triplets are either type-2 exceptions detected by the participating RTs or ones specific to the RP. These elements are coordinated in the same way as in a RT : the main program is activated after satisfaction of the preconditions, and normally ends when the postconditions are satisfied. This nominal execution can be aborted for processing the exceptions. The semantics of the RP formalism is defined statically in [42]. The operational one is defined by a translation into the synchronous language Esterel [34], since its semantics is expressed in terms of automata.

The automatic control aspects of an application mission are therefore exclusively considered at the RT level. Here, the verification issues mainly concerns the correctness of the logical and temporal behaviors. As seen in section 5, we are in fact interested in proving critical properties and conformity with application requirements.

4.3 Examples of Specification

We present in this section three examples of specification we have conducted using the RP formalism. The first one will also be our reference example for illustrating the verification process in ORCCAD.

Automatic Vehicle Driving In the considered application, the PRAXITÈLE Project ([42]), we have two electric vehicles, the leader being driven and the second having to follow it like in a virtual train. Initially, the undriven car tries to catch the right signal in order to locate the first one. When done, the expected nominal execution is that the undriven car follows the driven one until the two be stopped by an operator intervention. During the execution the video signal may be lost or irrelevant. In this case a parking maneuver is started. The driven car is supposed to come back and the train to be reformed. This informal specification could be translated into a RP named FOLLOWME (see fig. 1). Its nominal execution is described in the RP main program specified as an infinite loop the body of which begins with the test of the external condition *TargetFound* which indicates that the driven car is detected by the second one. Whenever it is satisfied, within a specified elapsed time, a RP named GUARDEDFOLLOW, detailed below, is activated to control the second car. Let us emphasize that the programming is structured, in the sense that we can use a RP inside the definition of another one. Before starting the FOLLOWME nominal execution, a set of three preconditions must be satisfied before the indicated delays; the initialization phase (motors, sensors, ...) must have been achieved without detecting errors, the automatic mode must be activated and the "human" operator has to give the start order. The nominal execution of this main RP is stopped in two cases: either the supervisor gives a stop order, or the manual mode is activated. In the first case, the RP ends normally; in the second one it is interrupted by a global type-3 exception.

In this example, six RTs are involved: two for the driving and steering motors of the car using exteroceptive sensor information, which virtually links the driven leader car to the non-driven following one (named SENSLOC and SENSDIR respectively); two allowing the undriven car to track a reference trajectory on the basis of odometry information only (respectively named CARTLOC and CARTDIR); one (named BRAKE) using the foot-brake of the car in order to impose a desired deceleration. The RP GUARDEDFOLLOW (see fig.1) is built from the parallel composition of three RTs : SENSLOC, SENSDIR and BRAKE . The RT BRAKE is activated every time the leading car imposes strong decelerations, indicated by *MoreBrake* event. Let us note that the first two RTs may be forced to stop if the exception of type 2 concerning the loss of the video signal between the two cars is detected (*SignalLost* event). The RP GUARDEDFOLLOW handles this situation by starting a recovery program, the RP

```
Name : FollowMe
Pre-cond:OkInit[30ms],AutoMode[30ms],Start[5mn]
Main program:
       Loop
          wait TargetFound [5mn]
          start (GuardedFollow)
       EndLoop
T3 exceptions: Auto2man, MecFaill, ...
Post-cond : Stop [60mn]
```

```
Name: GuardedFollow
Nominal execution  :
    Parallel
       start (SensLoc)
       start (SensDir)
       Loop
        if  MoreBrake then start (brake)
       EndLoop
    EndParallel
T2 Exception: (TargetLost, start (parking))
```

Figure 1: Specification of FollowMe and GuardedFollow RPs

parking, the specification of which is analog to GuardedFollow RP; the difference is that RTs based on odometry information only are used instead of the sensor-based ones.

Navigation of a Mobile Robot This example concerns the navigation of a mobile robot, anis, in an indoor environment ([71]). The mission goal is to achieve a *Reaching Target* task. Informally, it is specified as follows: the robot in a room must follow a wall while handling the presence of obstacles, reach a region of interest, find a target and go in front of it. Here, two RPs are hierarchically specified, encapsulating five RTs which include advanced control laws like a sensor-based one or a time-varying feedback; their activations are triggered by events which indicate the occurence of situations like the end of the wall, the presence of the target or of an obstacle, and failures. The abstract view of this application, in terms of RTs, as obtained by the verification process, is presented fig. 6.

An assembly operation This example concerns a simple assembly cell within a flexible manufacturing system. The tasks involve two robots cooperating in an assembly operation. Parts are fed by two conveyor belts. The cell has to achieve the following tasks: RobotA approaches, using trajectory tracking for large motions (RT MoveSe3) and vision-based control for accurate positioning (RT Approach), and grasps the first part on the first conveyor (RT Grasp). Then it places the part (RT UnGrasp) on the force-controlled parallel manipulator LeftHand where it is locked. RobotB, in an analog way, after grasping the second part by the second conveyor, displaces it near the LeftHand and assembles parts, (RT InsertBA). Once the assembly is unlocked, an external agent, not modelled here, moves it away and puts it inside a storage unit (action GetObjectAB). This cycle is repeated continuously.

Seven RTs are designed and regrouped in three RPs: GetPart which drives the robot to an adequate position and then grasps the object, Transport which transport an object to a desired position, and Assembly which specifies the whole application. We present here only the main program of Assembly RP. It states that three branches must be executed in parallel: they concern the actions affected to RobotA, RobotB and the external agent

respectively. BLOCK and UNBLOCK primitives are used to express synchronization points between two parallel components of a program.

PAR (LOOP(SEQ (GETPARTA, BLOCK(LhEmpty),TRANSPORTA, UNGRASPA,UNBLOCK(LhFull))),
LOOP(SEQ (GETPARTB, BLOCK(LhFull),TRANSPORTB, INSERTBA,UNBLOCK(ObjectOk))),
UNBLOCK(LhEmpty), LOOP (BLOCK(ObjectOk), GETOBJECTAB, UNBLOCK(LhEmpty)))

In conclusion, the RP formalism allows an user to program at the "task-level", without worring about the coding of tricky things, like signal exchange between elementary tasks. A systematic translation into appropriate synchronous languages is provided, minimizing in that way the risk of errors. Even so, the complexity of programmed applications and their critical character require the use of formal methods for ensuring the correctness at the *specification* level. This is the aim of the following section.

5 Formal Verification in the ORCCAD System

RP specification deals with the logical composition of typed events and actions as well as with time dependent characteristics, which usually express actions duration and timeouts. Their complete analysis requires to make both global and temporal quantitative analysis of the specification. For the first one, the most appropriate methods are the behavioral ones, which need to make a logical abstraction of the specification model in order to eliminate the delays which appear in the specification (they are transformed into logical time-out). For the second one, real-time model-checking techniques are required, which are based on extended models where the time is represented explicitly.

In order to perform behavorial verification, Esterel is used as a language of specification; it is interfaced with the Mauto verification tool (section 5.1). For temporal verification, we have to use Timed-Argos as a specification language and the Kronos symbolic verification tool (section 5.2). This explains why the RP and RT formalisms are both translated into Esterel and Timed-Argos according to the kind of properties one would like to check (see fig. 2a).

5.1 Logical Verification

We consider here only the logical aspects of an application: time-related constraints are translated into logical events triggered by the environment. In this framework, our first goal is to formally verify the largest possible set of assertions about the logical behavior of the RTs and RPs. The systematization of the verification process requires

• an adequate systematic translation of the specification into an automaton model preserving the information pertinent for the verification methods, i.e. here into an ESTEREL program;

• the classification of the properties to verify and the association of a dedicated verification method with every class. Two main categories of properties are identified: the first are related to critical working issues and the others are operational ones related to the completion of the desired objective. The generic *safety* and *liveness* properties are examples from the

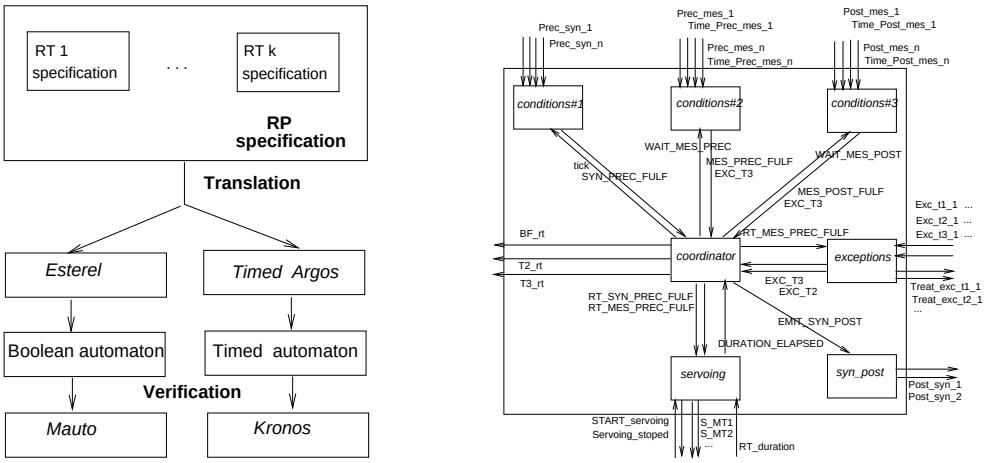

Figure 2: a) Specification-verification process in ORCCAD b) RT ESTEREL program architecture

first category, while those concerning the *coherency of the specification with the requirements* of the *application* belong to the second one. In addition, verification methods are used to create adequate *abstract views* aimed to help the user during the specification phase.

We present in the following the verification of the logical behavior of the RTs and the RPs respectively.

5.1.1 The Logical Verification of Robot Tasks

Our objective here is to outline the general approach of the verification process which proves that, by definition, the logical behavior of a RT is correct independently of a particular problem. For this, in a first paragraph, its translation into ESTEREL is discussed accompanied, for clarity purposes, with indications on the ESTEREL language and its programming methodology. The verification of its logical behavior using the ESTEREL verification environment is detailed in a second paragraph.

Translation of a Robot Task into ESTEREL

ESTEREL is a synchronous language, with a precisely defined mathematical semantics, for programming deterministic reactive systems. Its programming model is the specification of components, or *modules*, that run in parallel; modules can also have hierarchical structure. They communicate with each other and the outside world with *signals*, which are broadcasted and may carry values of arbitrary types.

The RT translation into an ESTEREL program, the architecture of which is shown figure 2b, following this programming model, is designed in such a way that each phase of the RT logical behavior is described in a distinct module: every particular specification of the RTs behavior uses these modules by substituting the external input/output signals. The phase

coordination is ensured by a separate module *independent from a particular specification.* All modules, denoted by boxes in the figure, are composed using the ESTEREL parallel operator. Arrows to/from modules indicate signals that are received/emitted by those modules. For clarity, the communication graph contains only arrows towards the modules that actually respond to the corresponding signal. Let us briefly comment their functionality.
- The *conditions* module is re-used three times with different signal substitutions. They correspond to synchronization and measurement pre-conditions and to measurement post-conditions respectively. After an external request, (ex: WAITMESPREC) it waits either for the satisfaction of the set of conditions, in which case it indicates it (ex: MESPRECFULF), or for the expiration of at least one related timeout, in which case a signal asking for a treatment of type 3 is emitted (EXCT3).
- The *servoing* module, after reception of the MESPRECFULF signal, asks the real-time operating system to start the MTs implementing the RT, or to disactivate them after the task completion. The logical signal RTDURATION. indicating that the the RT duration elapsed, causes the nominal termination of the RT.
- The *exceptions* module, after the reception of MESPRECFULF signal, waits for the exceptions that can occur during the servoing task execution. Their treatment is asked by the external environment through the emission of adequate signals.
- The *syn-post* module emits to the environment a set of signals associated with the nominal accomplishment of the RT.
- The *coordinator* module coordinates the activity of all the other modules requesting their activation and listening their responses in order to realize the RT behavior as described in 4.1. It reports, by signal broadcast, the way of the termination of the RT. Since it reflects the evolution of the RT behavior and is independent of a particular specification, it will be of particular interest in the verification of RTs critical properties.

The ESTEREL program is simply an implementation of each of these modules. For example, figure 3, shows the implementation of the *coordinator* module (language primitives are self-explanatory.)

Robot Task Verification

In relation with an ESTEREL program, one can generate, simulate, execute and verify automata. In particular MAUTO, an automatic verification tool, dedicated to analysis and transformation of finite automata, is directly linkable to the ESTEREL compiler. Three kinds of operations can be perforemed:
• *abstraction* extracts new possible behaviors (the transitions of an automaton) from sequences of actual ones. Typical simple abstractions are obtained by hiding several signals, while more elaborate may be realized using the syntax of *abstraction criteria* ([35]) to specify new behaviors. Then the "body" of a non-atomic behavioral activity may be replaced by a single one, at a different level of abstraction. Abstractions produce "abstracted automata".
• *reductions* assimilate and collapse states with the same behaviors abilities. The equivalence of states is drawn from *bisimulation* ([8]) equivalences. Combining reductions after abstractions often lead to very small automata: states are more often merged because the behaviors abilities were equated earlier.

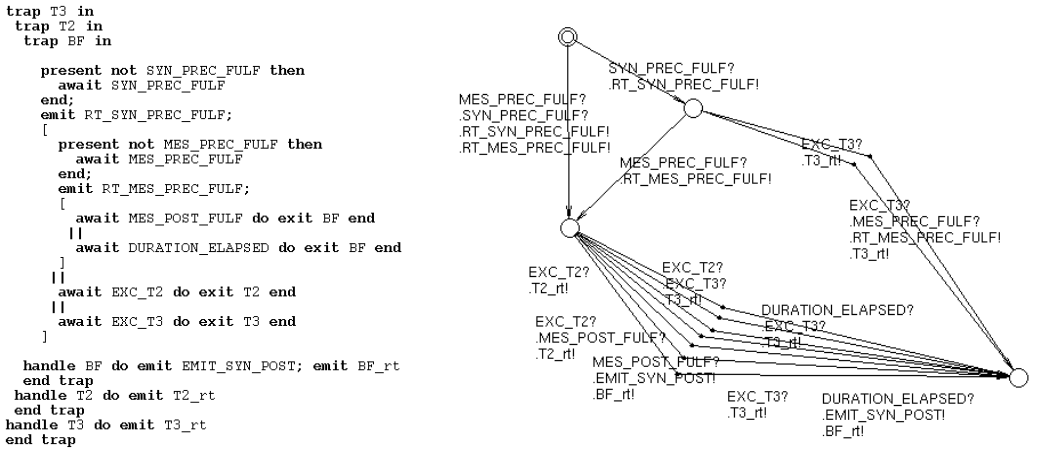

Figure 3: *coordinator* module a) implementation b) resulting automaton

• *context filtering* consists in observing the automaton through a context of possible activations. The result is an automaton derived from the early one by removing transitions and states that may not be reached in that context.

The resulted automata can be visualized before or after transformation using the MAUTO graphical interface AUTOGRAPH.

Owing to this environment the behavioral verification is performed, practically, in the following way: the global automaton is reduced with respect to the behaviors which are considered as relevant for the property to check, using the *abstraction-reduction* operations. The reduced automaton, when small, can be visually observed in order to invalidate or confirm the required property. An alternative way is to compare the reduced automaton with the specifications provided operationally in the form of an automaton. The property is verified when the reduced and the specification automata satisfy the *bisimulation* equivalence.

Automata produced by ESTEREL programs drive us to a nice modular way to verify them. The behavior of each module is checked separately ensuring its conformity with the required functionality. It then suffices to prove that their parallel composition does not alter their internal functioning.

Let us now analyze the RT logical behavior. We want to prove that it is non-blocking, and that it satisfies the *liveness* property, i.e. a successful termination of the RT can be reached from any state of its evolution, and the *safety* property, i.e any fatal exception is appropriately handled by emission of a specific signal leaving the system in a safe situation.

In a first step, we proved that each module implemented correctly its functionality and satisfied the required properties (a complete analysis is given in [43]). We present here only the *coordinator* module which reflects the RT logical evolution. Its corresponding automaton is small enough to be analyzed visually. We can therefore establish that it is non-blocking,

that a successful termination of the RT can be reached from any state and that a type 3 exception is always followed by the emission of the adequate signal.

It remained to prove that after the parallel composition of all the modules each of them still behaves as expected. The observation of the global automaton in relation with the inter-modules communication signals shown that all of them are present in the resulting automaton, that is to say their coordination did not altered their individual behavior.

Since the automaton model of the RT behavior is considered independently of a specific instantiation of RT input/output events we can conclude that, *generically*, the RT satisfies the required properties. The interest of this result is double: first, when analyzing the behavior of a RP, we can abstract the local behavior of RTs and consider it as *atomic* actions; secondly, from a software engineering point of view, the re-using of pieces of code proved correct a priori improves system reliability.

5.1.2 Logical Verification of Robot Procedures

Translation of a Robot Procedure into ESTEREL

The translation of a RP specification into an ESTEREL program is done in a structured way in order:
a) to enforce the reutilizability of the modules already designed. In addition, critical parts of the resulting program, independent from a particular specification are detected, programmed and verified once forever, improving in that way the overall reliability,
b) to insert the information needed for automatizing the verification process,
c) to clearly separate the parts concerning the local behavior of the actions from the parts dealing with their sequencing.

The resulting program architecture is shown figure 4. The following modules are running in parallel:
- $action_i$, which contains the local behavior of the i^{th} action involved in the RP specification. It reuses therefore either a RT program or a RP one. It is activated after reception by the *coordinator* module (see below) of a start signal, and reports to it the way it is finished;
- *conditions*, which indicates to the *coordinator* the RP pre and post conditions satisfaction;
- *transition*, which ensures the control of the switching mechanism between two control laws applied to the same robot. The control law associated with the next RT is started after acknowledgement that the previous was stopped;
- *coordinator*, which is the principal module of the RP since it coordinates, through local signal exchange, the activation of the involved actions in accordance to the main program of the RP as given by the end-user. It implements therefore the logics of the sequencing, taking into account the starting and ending times of the actions only.

The RP ESTEREL program is an implementation of each of these modules.

Robot Procedure Verification

The RP definition does not give us the possibility to conclude that, by construction, it verifies properties analog to those validated on the RT definition since most of them are application-dependent. Since the end-user is really involved in RP verification, this operation should take place interactively during the phase of RP design.

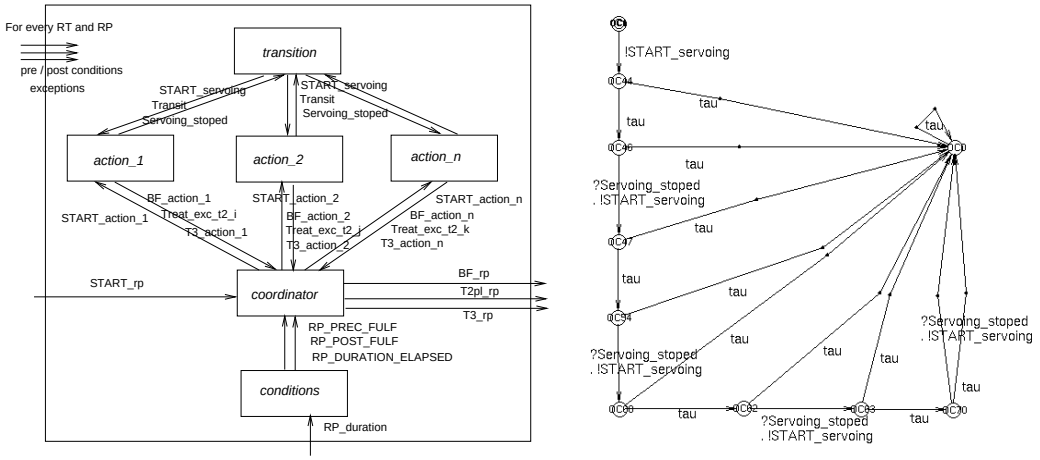

Figure 4: a) RP ESTEREL program architecture b) abstracted automaton according to signals related to starting and stopping of the servoing tasks

We describe now the classes of properties to be checked after having defined a mission in the form of a set of RPs. In the first paragraph we present properties which are *independent* of a given application; they are global crucial properties the verification of which the *end-user* would ideally just have to ask for. Then, we focus on parts of the global behavior indicating the evolution of the system in particular conditions specified by the user. The last paragraph copes with the construction of views of the logical behavior at different levels of abstraction.

Crucial properties

- **Safety property**

  For the verification of the safety property, knowing the user's specification about fatal exceptions, abstract criteria composed by a set of abstract actions of the form $Bad_i = /Type\_3\_exception_i?\ and\ (not\ /Exc\_T3!)$ can be systematically produced. Such an abstract action indicates all the possible evolutions of the system where, although the particular exception specified as a type-3 one be an input of the system, no output asking to drive it to a safe position is emitted. Then, the abstraction of the RP automaton with respect to this criterion is computed. The presence or absence in the resulting automaton of an action of the criterion invalidates or confirms the safety property.

- **Liveness property** For proving the liveness property, we examine the equivalence by bisimulation of two automata. The first is derived by abstraction of the initial automaton with respect to the criterion

$$parse-criterion\ Viv \quad = \quad tau = (not\ /BF\_rpr!)*,\ \ BF = /BF\_rpr!;$$

which renames as *tau* (invisible action) all the sequences of actions (regular operator (*)) which do not indicate the good termination, and as BF the remaining ones. The second is specified as an automaton with one state and one transition labeled BF.

- **Conflicts detection**

  We are interesting here to check that during the RP evolution it does not exist a time at which two different RTs are competing for the use of a particular resource (physical or software) of the system. For verifying that, in the case of the mobile robot ANIS, we observe the global automaton with respect to the signals *StartServoing* and *ServoingStop*. Figure 4b shows the resulting automaton. We can easily verify that the two signals appears alternatively.

Coherence with the Application Requirements The conformity of the RP behavior with respect to the mission constraints can also be verified. These constraints have to be expressed in a generic way as relationships between actions, events and actions or events themselves. In the verification process the user has to indicate the relevant set of events and/or actions related to the constraint to be checked. Global automaton is then abstracted and reduced by well-chosen signals among those considered in the translation of the RP into its automaton model via ESTEREL. Let us illustrate that through the examples of section 4.3.

- **Action/Action** In the assembly operation, the LEFTHAND robot imposes that its platform has to be free before another object be placed on it. We want to check that our specification is conform to this requirement. The relevant actions to be observed, as appearing in the specification, are GETOBJECTAB, INSERTBA and UNGRASPA. The corresponding signals are those indicating the starting and nominal termination instants (see fig. 5a); the resulting abstracted automaton clearly indicates that the assembly is always moved away from the platform before another object be brought.

- **Event/Action** Let us refer again to the example of autonomous vehicle driving. Mission specification requires that the brake should be activated every time the leading car imposes strong decelerations. The involved elements in the expression of this constraint are the *MoreBrake* event and the BRAKE action. The RP automaton is observed by *MoreBrake* and START_brake signals. The property is verified since in the resulting automaton, figure 5b, there exists a loop indicating that, as soon as *MoreBrake* signal is received, the activation of the BRAKE action is asked for.

- **Event/Event** In the same example, the specification requires that the target is searched for each time it has been lost. This is tested by specifying a behavioral property which involves the events *SignalLost* and *TargetFound*. The resulting abstracted automaton, figure 6a clearly certifies it.

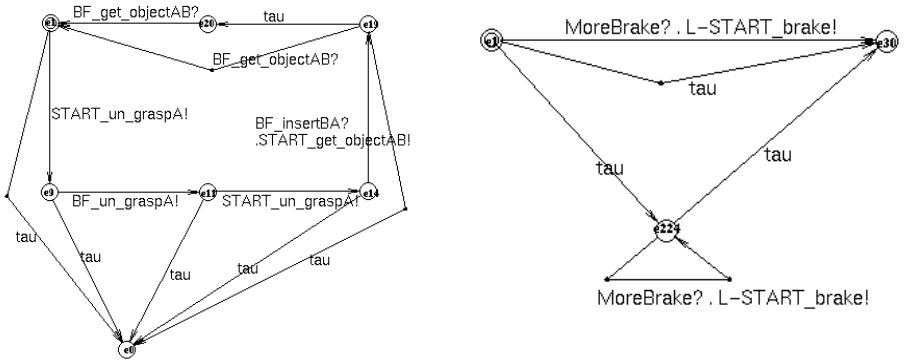

Figure 5: Abstracted Automata of a) the 'assembly' operation b) the 'follow me' mission

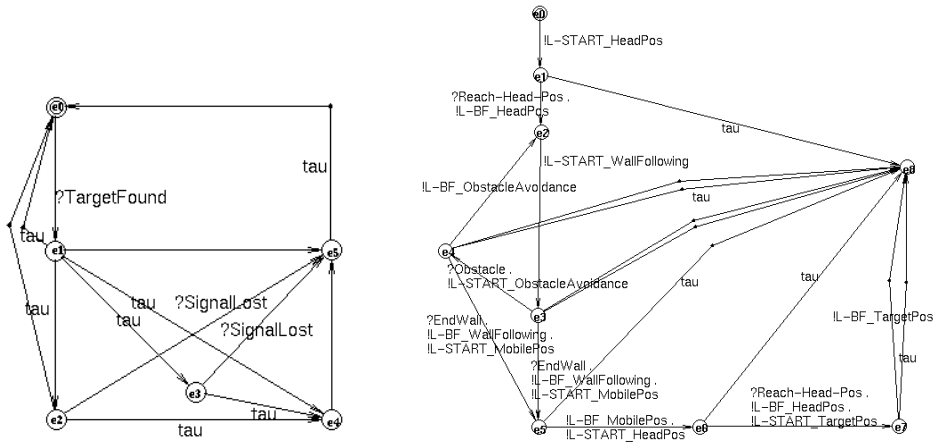

Figure 6: a) Abstracted automaton of the 'follow me' mission according to two events b) abstract view of the 'reach target' mission

Abstract views Finally, we consider that verification methods are also as a way of obtaining *abstract views* during the phase of RPs specification, in order to help the user. Actually, every mismatch in the behavior specification is reflected by the resulting automaton. So, by switching the body of a non-atomic behavioral activity to a single one at a relevant level of abstraction, useful views of the overall behavior can be given. Figure 6b illustrates the nominal execution of the global mission ReachTarget of the mobile robot Anis at a RT's level of abstraction; only starting and final instants of RTs are considered, making abstraction of their internal evolution.

5.2 Temporal Verification

The approach based on Esterel is well-suited for specifying and verifying the logical behavior of a robotic application. Nevertheless, it does not allow to take into account its temporal aspects. For example, it is possible to prove that an application always ends, but it is not possible to prove that its execution time is always lower than a constant T. The logical abstraction (delays are translated in logical timeout events) can also leads to inaccurate analysis if these ones are time-dependent. Proving that two execution laws are never executed simultaneously without taking into account the time parameters involved in the programs leads to approximate analysis only : the result is true if, whatever the time parameters are, the two execution laws are sequential.

We propose in this section another approach based on a synchronous language named Timed Argos, which allows to take into account the time parameters involved in a program during its formal analysis. As above-mentioned, Timed Argos has a semantics in terms of Timed Automata. We therefore have to translate the controller specification into a Timed Argos program, to use the Argos compiler to build the Timed automaton model of the program and finally to use the verification tool Kronos to check some relevant time-dependent properties.

This section is organized as follows. We first present the Timed Argos language and the Kronos verification tool (especially the real time temporal logic TCTL which is used by Kronos as the formalism to express properties). Then, we present a classification of relevant time-dependent properties which gives an idea of the way we would like to integrate this temporal verification inside ORCCAD. In a third subsection we give some experimental results and we discuss some major concerns of the used method. Finally, we present another kind of experiment with the Hybrid Argos language and the Polka tool the aim of which is to synthetize linear invariants on linear hybrid automata.

5.2.1 Timed-Argos and the Kronos tool

The Timed-Argos Language

Timed Argos [41] is an extension of the synchronous language Argos [21]. Argos was originally inspired by Statecharts. It provides the user with a set of operators that can be applied to elementary automata components to build more complex systems. These

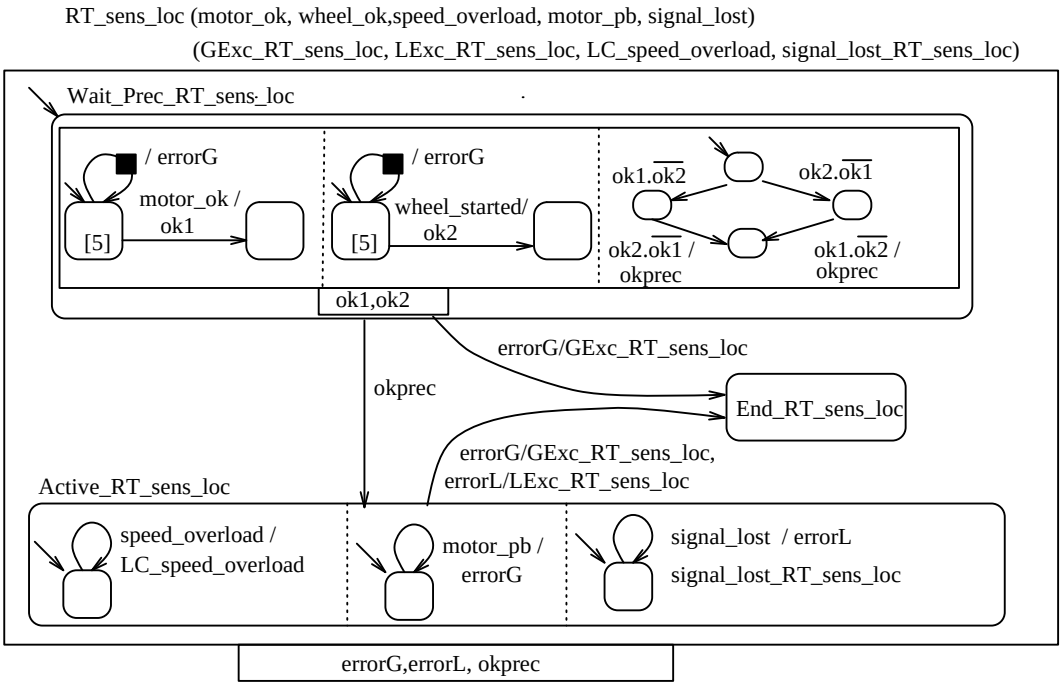

Figure 7: RT_sens_loc specification translated into Timed Argos

operators include parallel and hierarchic composition. The Argos semantics is expressed in terms of boolean automata.

Argos has been recently extended with a delay construction, leading therefore to Timed Argos, which allows to express watchdogs and timeouts easily. The semantics of Timed Argos is expressed in terms of timed automata. For this reason, Timed Argos is a high-level language to describe this kind of extended automata and Timed Argos programs could support quantitative analysis. Until now, Timed Argos is more specifically interfaced with the Kronos verification tool [40].

The first step to integrate quantitative timing analysis of the controller specification into ORCCAD is to translate it into a Timed Argos program. The translation is structurally defined : the Robot Tasks involved in the Procedure Robot which specifies the controller are first translated into Timed Argos subprograms; then the structure of the Robot Procedure is taken into account in order to derive the main program. Since this translation is presented in [42], we only illustrate it here by an example taken from the automatic vehicle driving mission already mentioned.

Figure 7 gives the result of the translation of a RT which includes two preconditions : `motor_ok, wheel_ok`, one type-1 exception : `speed_overload`, one type-2 exception `signal_lost` and one type-3 exception `motor_pb`.

The program handles five inputs : `motor_ok, wheel_ok, speed_overload, motor_pb` and `signal_lost`; and four outputs : `GExc_RT_sens_loc, LExc_RT_sens_loc, LC_speed_overload` and `signal_lost_RT_sens_loc`. The first two outputs indicates the detection of a global or a local exception during the execution of the Robot Task. The last ones are self-explanatory.

The other events used in the subprogram : `ok1,ok2, errorG, errorL` and `okprec`, are local ones used to force the communication between automata of the program. The communicating process will be explained later.

Each object which composes an Argos program is an automaton the transition labels of which are of the form *inputs/outputs*. The *inputs* part expresses the condition under which the transition is triggered. Absence of events is denoted by overlining. The *outputs* part is the set of output events emitted when the transition is triggered. An arrow without an associated source state denotes the initial state of the automaton.

Some of these automata have temporized states, which are drawn with labels of the form $[n]$. These temporized states also have an outgoing transition the label of which is replaced by a square root. This is the timeout transition. Their intuitive semantic is as follows : once a temporized state is entered, it must be left before the indicated amount of time has elapsed. The automaton can leave the temporized state through a "normal" transition (if such a transition exists), or through the special timeout one when the delay expires.

This program is built hierachically. The outermost automaton has three states according to the current status of the RT : while waiting for the preconditions to be satisfied, active (the execution law is alive) and finished. The first of these status could end either when the two preconditions are satisfied or when a global exception is detected, i.e. the delay associated to a precondition expires. The `Wait_Prec_RT_sens_loc` state is refined by three automata which are composed in parallel : they evolve simultaneously. The box with an attached subbox which contains `ok1, ok2`, indicates that these two events are local. They are used both as inputs and outputs. The communication process is called *synchronous broadcast*. This is the same as in the Esterel language. If a local event is emitted by a component, each automaton could react to this emission by triggering other transitions. All these transitions participate to the same global reaction of the program.

The Kronos Verification Tool

The Kronos tool implements a symbolic model-checking algorithm for TCTL [30] (a real-time extension of the branching-time logic CTL) on Timed Automata. It means that the property is expressed by a TCTL formula and that Kronos computes the set of states of the Mealy Machines associated with the Timed Graph which satisfy it. The property is satisfied if and only if the initial state belongs to this set. The algorithm implemented by Kronos is symbolic, since the Mealy Machine associated with the Timed Graph is always represented implicitly.

A TCTL formula ϕ is built following the grammar :

$$\phi ::= p|\neg\phi|\phi \wedge \phi|\forall\exists_{\#c}\phi|\exists\forall_{\#c}\phi|\forall\forall_{\#c}\phi|\exists\exists_{\#c}\phi$$

belongs to $\{<, \leq, >, \geq\}$. c is an integer value. p is a property of *states* (i.e. nodes and valuation of the clocks), and can be identified to the set of states where it is true. The set can be given in extension, but it is usually described by using a function which builds *state properties* out of *transition properties*. For instance, `enable`(l) computes the set of states q such that there exists at least one transition sourced in q and labelled by l.

Let us illustrate the semantics of TCTL with the following example: $\exists\exists_{<4}\{q\}$. A node q' satisfies this formula if and only if there exists one execution sequence from q' such that a state satisfying q is reached before 4 units of time. It expresses the *possibility* to reach q' before 4 units of time. Some formulas do not have temporal restrictions (given by the $\#c$ expression): $\forall\exists\phi$ expresses that ϕ will be satisfied *eventually*, i.e. for each execution sequence from q' there exists a state satisfying ϕ. $\forall\forall\phi$ expresses that ϕ is an *invariant* property and $\exists\forall\phi$ is satisfied if and only if there exists one execution of the program on which ϕ is always satisfied.

5.2.2 A Classification of Relevant Temporal Properties

The aim of this part is to present a classification of temporal properties that it could be interesting to check inside the ORCCAD environment. These properties are intended to help the designer in specifying his application. The idea is to avoid the designer using TCTL to express properties, by identifying generic ones which could be automatically translated into TCTL formulas. This automatic translation is closely related to the automatic translation of the controller specification into Timed-Argos.

- **Time bound property**
  This property is concerned by the boundedness of the maximum execution time of a Robot Procedure. The time bound property is the only one which could be checked automatically (without any designer's participation) whenever a complete specification of a Robot Procedure is built.

- **Event / Event properties**
  This class of properties aims to check that in case of a complete execution (without detecting a type 3 exception) of a Robot Procedure, the maximum (resp. the minimum) laps of time between two events of the robotic controller is lower (resp. greater) than a value T. For instance, during the specification of an automatic car parking, the designer would like to check that the maximum time-lag between :

  - the permission to enter given by the controller to a car and
  - the parking acknowledgement given by the same car to the controller

  is bounded by a value T specified by the designer. He has to specify the name of the two events involved by the property, the constant value T, and its choice between a maximum or a minimum.

- **Event / Action properties**
  This class of properties expresses that, in case of a complete execution of a Robot

Procedure, the maximum (resp. the minimum) laps of time between one event of the robotic system and the start of either a Robot Task, or a Robot Procedure, or a Robot Task execution law, is lower (resp greater) than a value T. Let us take again the example of the automatic vehicle driving mission. The designer would like to check that the maximum time-lag between the `more_brake` event, which indicates that the engine-braking is not sufficient to stop the car, and the beginning of the Robot Task RT_brake execution law, is smaller than a value T.

- **Action / Action properties**
  This class may be splitted into two subclasses :
  - Sequential Action / Action properties. The property to verify is that the maximum (resp. minimum) time-lag between two sequential actions (with the above-mentioned meaning) is lower (resp. greater) than a value T.
  - Overlapping Action / Action properties. The property to verify is that the maximum (resp. minimum) time-lag between two actions which could overlap themselves is lower (resp. greater) than some value T.

5.2.3 Performance Evaluation

The experiment we present here concerns the automatic vehicle driving mission described above. The time performances we give have been obtained on a middle-size workstation.

The first quantitative results of the experiments we performed concern the Argos compiler performances. We give in the following table the information about the computed timed-automaton: the numbers of transitions, states, local events, clocks and inputs — each of these quantities is involved in the compilation time of a Timed-Argos program.

The main program is the Timed-Argos program associated with the main PR which describes the controller of the mission. The program named *Reduced* is obtained by associating one waiting time limit with each *set* of preconditions, instead of each precondition as in the main one. We will see below that this slight difference of specification has significant consequences on the performances of the Argos compiler and the Kronos prover.

Example	Trans	States	Local Events	Clocks	Inputs	Compilation Time
Main	31012	105	40	15	30	0h 20
Reduced	12978	105	40	11	30	0h 10

In the next table, we focus on the time performances of Kronos while checking that the application is time-bounded. The timed automaton computed by the compiler is not optimal for verification purposes. For instance, given the fact that the property we check does not depend on the timed automaton labels, it is possible to abstract them and then to minimize the timed automaton. This reduction phase can be achieved using Aldebaran [11], the time performances of which are very good.

Example	Clocks	Trans	States	Verification Time
Main (after optimizations)	9	4004	105	0h 50
Reduced (after optimizations)	11	705	44	0 h 01

6 Concluding Remarks

It begins to be recognized in the robotics area that verification is a major concern, especially when robots have to be launched towards hostile environments where they should survive. We have proposed in this paper an approach to specify and to verify, in a highly structured way, such complex robotics applications. Logical as well as temporal issues were considered. In order to make easier and more reliable the designer's activity, we tried to use as far as possible available and efficient softwares from the computer science area: the ESTEREL, ARGOS, TIMED-ARGOS languages for specification and programming; the MAUTO, AUTOGRAPH, KRONOS, ALDEBARAN tools for automata handling and formal proofs. We also addressed several applications in order to show that the proposed approach was really pertinent.

However, although we obtained some successes in that work, for example in proving generic properties, we also found that nice improvements could be done in order to make the approach fully utilizable in the applications. This mainly concerns the "timed" part of the analysis:

- although the example we implemented was not so complex, the size of the timed automaton of the controller was large (about 100 states, 30000 transitions and 15 clocks). This timed automaton should be much smaller for verification purposes. Therefore, instead of building the large-size automaton and then minimizing it, it is necessary to define a specific compilation process which would integrate the minimization phase, in order to improve its time performances.

- the identification of generic properties allows the user to avoid learning TCTL. It is a first (big) step in the integration process of real-time quantitative properties into the ORCCAD environment. Nevertheless, we have not yet found a way to solve the "error diagnosis problem". Indeed, when a property is not satisfied the diagnosis given by Kronos does not allow to find easily the error in the program: this diagnosis is expressed in terms of timed-automaton information and not in terms of the source program. Given the fact that the semantics of Argos is such that it is easy to rely the information model on the source program, this drawback should be removed in the future.

Another important point comes from the fact that the approach based on Timed Argos and Kronos provides the user with a way to verify if its program satisfies a time-dependent constraint. However, from the designer's point of view, nothing is done to help him to select the right delay values which will necessarily satisfy this constraint. This complementary problem of *synthesis* is indeed very relevant for the application area since its lies at the

design level. It cannot be solved by Kronos since the delays values have to be known during the analysis. A first step to address this problem is to combine the Hybrid Argos language and the Polka analysis tool [27].

Hybrid Argos is an extension of Argos allowing to model linear hybrid systems. We can use it in order to produce the model of a controller wherein some delays are *not* quantized. Polka is a tool for synthesizing linear invariants about numerical variables of a program. It computes an upper approximation of the set of variable valuations that can be reached at a given control point of a program. A very common usage of Polka results is the determination of unreachable states and transitions, when the associated linear constraints cannot be satisfied. As an example, Hybrid Argos and Polka can be used for answering the following question:
Given a Robot Task with some delays, what should be the delay values to ensure that the maximum execution time of the Robot Task is lower than T?
The solution is based on the so-called technique of synchronous observers used in synchronous programming environments. The idea is to add to the program to verify a component which observes if the constraint to satisfy is violated. If yes, a special error state is reached. Hybrid Argos is used to compute the linear hybrid model of the system {Robot Task and observer}, and Polka computes the values of the delays which make the error state reachable. We have realized some tentative tests with this approach. Results were promising, although it appeared that performances should be considerably improved if we want to deal with large scale applications.

More generally, we believe that all the area of hybrid systems (modelling, programming, formal verification) is a key research domain for the future, the results of which will find particularly relevant applications in robotics.

References

[1] S. Owre, J. Rushby, N. Shankar:" PVS : A prototype verification system." In *Automated Deduction* CADE, volume 607 of Lectur Notes in Artificial Intelligence pages 748-752, Springer Verlag 1992.

[2] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin, B. Werner. "The Coq Proof assistant User's Guide, Version 6.6" *INRIA Technical report 134,* Dec 1991.

[3] C.B. Jones " Program specification and verification in VDM" *Technical report N. 083 UNIVERSITY OF MANCHESTER,* Nov. 1986.

[4] R. W. Floyd," Assigning Meaning to Programs" Proceedings of Symposis in Applied Mathematics, in *Mathematical Aspects of computer science*, J.T. Schwartz editor, vol. 19, pp 19-32, 1967.

[5] C. A. R. Hoare, " An Axiomatic Basis for Compter Programming," *Communications ACM*, vol 12, n. 10, pp 576-583, 1969.

[6] R. Alur, T.A, Henzinger and P.S. Ho " Automatic Symbolic Verification of Embedded Systems",*14th annual Real-Time Systems Symposium*, 1993.

[7] A. Pnueli, " The temporal Logic of Programs", *18th Annual symposium on Foundations of computer Science*, Providence, pp 46-57, 1977.

[8] R. Milner, " A Calculus of Communication Systems", *LNCS 92*, Springer Verlag, 1980.

[9] E. M. Clarke, E. A. Emerson and A. P. Sistla , " Automatic Verification of Finite-state Concurrent Systems Using Temporal-logic Specifications", *ACM Transactions on Programming Languages and Systems*, Vol. 8, 1986.

[10] J. Queille and J. Sifakis, " Specification and Verification of Concurrent Systems in CESAR", *LNCS 137, 5th International Symposium in Programming*, Springer Verlag, 337–351, 1981.

[11] J.C. Fernandez, " An Implementation of an Efficient Algorithm for Bisimulation Equivalence", *Science of Computer Programming*, Vol. 13, N. 2-3, may 1990.

[12] R. Cleaveland, J. Parrow, B. Steffen, " The Concurrency Workbench, Workshop on Automatic Verification Methods for Finite State Systems", *Vol. 407, LNCS, june 1989.*

[13] J.-L. Richier, C. Rodriguez, J. Sifakis, Voiron J , *Xesar : A Tool for Protocol Validation. User's Guide*, Technical report of LGI-IMAG, Grenoble, France, 1987.

[14] A. Bouajjani, J.-C. Fernandez and N. Halbwachs, " Minimal model generation", *International Workshop on Computer Aided Verification*, Rutgers, june, 1990.

[15] C. Courcoubetis, M. Vardi, P. Wolper and M. Yanakakis, " Memory Efficient Algorithms for the Verification of Temporal Properties" *International Workshop on Computer Aided Verification*, Rutgers, june, 1990.

[16] R. E. Bryant, " Graph-based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, Vol. C-35, N. 8, 1986.

[17] O. Coudert, C. Berthet, J.-C. Madre, " Formal Boolean Manipulations for the Verification of Sequential Machines" *IMEC-IFIP Internationnal Workshop on Applied Formal Methods For Correct VLSI Design*, 1990.

[18] J.-C. Fernandez, A. Kerbrat, L. Mounier, " Symbolic Equivalence Checking", *Fifth Conference on Computer-Aided Verification*, Elounda (Greece), LNCS 697, Springer Verlag, 1993.

[19] A. Benveniste and G. Berry," Another Look at Real-Time Programming", *Proceedings of the IEEE*, N. 9, Vol. 79, 1991.

[20] D. Harel,*Statecharts: a Visual Approach to Complex Systems,* Weizmann Institute of Science, 1984.

[21] F. Maraninchi, " Operational and Compositional Semantics of Synchronous Automaton Compositions", *CONCUR, LNCS 630*, Springer Verlag, 1992.

[22] T. Bolognesi and E. Brinksma, " Introduction to the ISO Specification Language LOTOS", *Computer Networks and ISDN Systems*, Vol. 14, N. 1, 25–29, 1988.

[23] J. Perraud, O. Roux, M. Huou :" Operational Semantics of a Kernel of the Language Electre",*Theoretical Computer Science*, N. 100, novembre 1992.

[24] O. Roux and V. Rusu: " Decidable Hybrid Systems to Modelize and Verify Real-Time Applications", *2nd European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, 31-5/2-6 1995.

[25] Y. Kesten, A. Pnueli, J. Sifakis and S. Yovine, "Integration graphs: a class of decidable hybrid systems" *Workshop on Theory of Hybrid Systems,* LNCS 736, 1993.

[26] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine, " The Algorithmic Analysis of Hybrid Systems", *Theoretical Computer Science* , Vol. 137, 1995.

[27] N. Halbwachs, Y.-E. Proy and P. Raymond, " Verification of Linear Hybrid Systems by Means of Convex Approximations", *International Symposium on Static Analysis, SAS'94*, LNCS 864, 1994.

[28] T. A. Henzinger and P.-H. Ho, "Model Checking Strategies for Hybrid Systems", *Conference on Industrial Applications of Artificial Intelligence* and Expert Systems, 1994.

[29] A. Bouajjani, Y. Laknech, R. Robbana *From Duration Calculus to Linear Hybrid Automata* To appear in the proceedings of the 7th International Conference on Computer-Aided Verification 1995.

[30] R. Alur, C. Courcoubetis and D. Dill , " Model-checking for real-time systems" *Proceedings of the fifth annual IEEE symposium on Logics In Computer Science 1990*, IEEE Computer Society Press.

[31] C. Astraudo, J.J. Borrelly, "Simulation of Multiprocessor Robot Controllers", *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, May 1992.

[32] D. Simon, P. Freedman and E. Castillo: *On the Validation of Robotics Control Systems. Part II: Analysis of real-time closed-loop control tasks*, Inria Research Report #2720, November 1995

[33] E. Coste-Manière, B. Espiau, E. Rutten: " A Task-Level Robot Programming Language and its Reactive Execution", *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 2751-2756.

[34] G. Berry, G. Gonthier: " The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation", *Science Of Computer Programming*, Vol 19 no 2, pp 87-152, 1992.

[35] G. Boudol, V. Roy, R. de Simone, D. Vergamini: " Process calculi, from theory to practice: Verification tools", in *International Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, LNCS 407, Springer Verlag, 1990.

[36] E. Castillo, *Principes, techniques et outils de simulation, vérification et exécution d'actions robotiques*, PhD dissertation, INPG Grenoble, France, November 1994.

[37] E.M. Clarke, A. Emerson, A.P. Sistla: "Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications: a practical approach", *Proc. 10th ACM Symp. on Principles of Programming Languages*, pp 117-126, 1983.

[38] R. de Simone, D. Vergamini: *Aboard* AUTO, INRIA Technical Report no 111, 1989.

[39] J.C. Fernandez, *Aldébaran, a tool for verification of communicating processes*, Tech. Report Spectre 14,LGI-IMAG, Grenoble, 1988.

[40] T. Henzinger, X. Nicollin, J. Sifakis and S. Yovine, " Symbolic Model-Checking for Real-Time Systems",*LICS 92*, IEEE Computer Society Press, June 1992.

[41] M. Jourdan, F. Maraninchi and A. Olivero " Verifying quantitative real-time properties of synchronous programs", *5th International Conference on Computer-aided Verification*, LNCS 697, Springer Verlag, June 1993.

[42] M. Jourdan, *Integrating formal verification methods of quantitative real-time properties into a development environment for robotic controllers*, Inria Research Report n. 2540., 1995

[43] K . Kapellos: *Environnement de programmation des applications robotiques réactives*, PhD dissertation, Ecole des Mines de Paris, Sophia Antipolis, France, November 1994.

[44] D. Simon, P. Freedman and E. Castillo, " Analyzing the Temporal Behavior of Real-time Closed-loop Robotic Tasks", *IEEE Int. Conf. on Robotics and Automation*, San Diego, 1994.

[45] D. Simon, B. Espiau, E. Castillo, K. Kapellos: " Computer-aided Design of a Generic Robot Controller Handling Reactivity and Real-time Control Issues", *IEEE Trans. on Control Systems Technology*, vol 1, no 4, December 1993.

[46] C. Samson, M. Le Borgne, B. Espiau: *Robot Control: the Task-Function Approach*, Clarendon Press, Oxford Science Publications, U.K., 1991.

[47] P.J. Ramadge, W. M. Wonham," The Control of Discrete Events Systems", *Proc. of the IEEE*, 77(1), 1989.

[48] P.J. Ramadge, W. M. Wonham," Supervisory Control of a Class of Discrete Event Processes", *SIAM J. Contr. Optimization*, 25(1), pp 206-230, 1987.

[49] D. Simon, K. Kapellos, B. Espiau, M. Jourdan: " Formal Verification of Missions and Tasks" *2nd European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, 31-5/2-6 1995.

[50] E. Rutten, P. Le Guernic, " Sequencing Data Flow Tasks in SIGNAL" *Workshop on Languages, Compilers and Tool Support for Real-Time Systems*,)rlando, USA, june 1994.

[51] R. Brooks: " A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, March 1986, pages 14-23.

[52] J. Kosecka, H. Christensen, R. Bajcsy, "Discrete Event Modelling of Visually Guided Behaviors", *Int. J. of Computer Vision*, 14, pp 179-191, 1995.

[53] B.A. Brandin, W.M. Wonham, *Supervisory Control of Timed Discrete Event Systems*, Technical Report 9210, Systems Control Group, University of Toronto, Canada, 1992.

[54] W. Kohn, J. James, A. Nerode, K, Harbison, A. Agrawala, "A Hybrid Systems Approach to Computer-Aided Control Engineering" *IEEE Control Systems Magazine*, pp 14-25, april 1995.

[55] K.T. Seow, R. Denavathan, "A Temporal Logic Approach to Discrete Event Control", *IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, may 1995.

[56] R. Kumar, V. K. Garg, S.I. Marcus, "Predicate and Predicate Transformers for Supervisory Control of Discrete Event Dynamical Systems" *IEEE Trans. on Automatic Control*, vol 38, no8, pp 1214-1227, august 1993.

[57] M. Antoniotti, B. Mishra, "Discrete Event Models + Temporal Logics = Supervisory Controller: Automatic Synthesis of Locomotion Controllers", *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995.

[58] M. Antoniotti, M. Jafari, B. Mishra, "Applying Temporal Logic Verification and Synthesis to Manufacturing Systems", *IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, Canada, Oct. 1995.

[59] D.M. Lyons, "A Process-Based Approach to Task-Plan Representation", *IEEE Int. Conf. on Robotics and Automation* Cincinatti, USA, may 1990.

[60] T.G. Murphy, D.M. Lyons, A.J. Hendriks, "Visually Guided Multi-Fingered Grasping as Defined by Schemas and a Reactive System", *Workshop on Neural Architectures and Distributed AI*, USC, Los Angeles, USA,

[61] D.M. Lyons, A.J. Hendriks, "Safely Adapting a Hierarchical Reactive System", *SPIE Symp. on Intelligent Robots and Computer Vision, XII*, Boston, USA, 1993.

[62] D.J. Musliner, E.H. Durfee, K.G. Shin, "Reasoning about Bounded Reactivity to Achieve Real-Time Guarantees", *in Proc. AAAI Spring Symposium on Selective Perception*, March 1992.

[63] O. Causse, H.I. Christensen, "Hierarchical Control Design Based on Petri Net Modelling for an Autonomous Mobile Robot", *Intelligent Autonomous Systems Conf (IAS 4)*, Karlsruhe, Germany, march 1995.

[64] Rahimi, Xia, "A Framework for Software Safety Verification of Industrial Robot Operations", *Computer and Industrial Engineering*, vol 20 no 2 , pp 279-287, 1991.

[65] Z. Ying, A.K. Mackworth, "Specification and Verification of Constraint Based Dynamic Systems", in *2nd Int. Workshop on Principles and Practice of Constraint Programming*, Springer Verlag, 1994.

[66] N. Lynch, H.B. Weinberg, "Proving Correctness of a Vehicle Maneuver: Deceleration" *2nd European Workshop on Real-Time and Hybrid Systems,* Grenoble, France, 31-5/2-6 1995.

[67] E. Coste-Manière, M. Perrier, A. Peuch "Mission Programming: Application to Underwater Robots" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995.

[68] A. Deshpande, P. Varaiya, "Design and Evaluation Tools for Automated Highway Systems" *2nd European Workshop on Real-Time and Hybrid Systems,* Grenoble, France, 31-5/2-6 1995.

[69] S. Gaubert, "Timed Automata and Discrete Event Systems", *Proc. European Control Conf.*, Groningen, july 1993

[70] G. Cohen, P. Moller, J.P. Quadrat, M. Viot, "Algebraic Tools for the Performance Evaluation of Discrete Event Systems", *IEEE Proceedings: Special Issue on Discrete Event Systems*, 77(1), january 1989.

[71] R. Pissard-Gibollet, K. Kapellos, P. Rives, J.J. Borrelly, "Real-Time Programming of Mobile Robot Actions Using Advanced Control Techniques" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995.

[72] P. Rives, R. Pissard-Gibollet, K. Kapellos : " Development of a Reactive Mobile Robot Using Real Time Vision", *Third International Symposium on Experimental Robotics*, Kyoto, Japan, Oct 28-30, 1993.

[73] D. Simon, K. Kapellos, B. Espiau "Formal Verification of Missions and Tasks: Application to Underwater Robotics" *Int. Conf. on Advanced Robotics, ICAR'95*, Barcelons, Spain, sept. 1995.

[74] K. Kapellos, S. Abdou, M. Jourdan, B. Espiau "Specification, Formal Verification and Implementation of Tasks ans Missions for an Autonomous Vehicle" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995.

[75] K.P. Kalavanis and al., editors, "International Program Development in Undersea Robotics and Intelligent Control", *Proc. of the Joint US Portugal Workshop*, Lisboa, Portugal, march 1995.

[76] Y. Lu, J. Buisson, "Analysis and Representations of Hybrid Systems with Singular Systems", *First Asian control Conference*, Tokyo, Japan, July 1994.

[77] J. Kosecka, H. Christensen," Experiments in Behavior Composition", *3rd Int. Symp. on Intelligent Robotic Systems*, Pisa, Italy, July 1995.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399