The ORCCAD Architecture

The ORCCAD Team:
Jean-Jacques Borrelly
Eve Coste-Manière
Bernard Espiau
Konstantinos Kapellos
Roger Pissard-Gibollet
Daniel Simon
Nicolas Turro
INRIA Sophia Antipolis
INRIA Rhônes-Alpes

Abstract

The ORCCAD *programming environment for robotic systems allows users to address automatic control laws in continuous time at the lower levels, and aspects of discrete-time logic at the higher levels.* ORCCAD *provides tools of specification, formal verification, simulation, and real-time code generation integrated within a set of dedicated graphical interfaces. Basic robot actions, which are intrinsically hybrid entities, are handled by the* ROBOT-TASK *structure, which smartly interfaces aspects of continuous and discrete time.* ROBOT-TASK*s are further logically composed into more complex actions,* ROBOT-PROCEDURE*s, through a dedicated language. While system performance can be checked using simulations, crucial properties such as deadlock avoidance, safety, and liveness can be formally verified at both levels. The approach is illustrated with an underwater inspection mission.*

1. Introduction

It has long been a goal in robotics to build autonomous vehicles to replace humans in hostile or unreachable environments, such as in nuclear plants, mines, underwater areas, and on other planets. Some good solutions have been found to parts of the problems raised by these autonomous robots, thus decreasing the gap between fiction and reality. These solutions touch upon numerous issues: mechanics, sensing, modeling, automatic control, etc.

Of increasing importance is the role played by the embedded software required to integrate all of these aspects. To design any operational robotics system, it is crucial to formally organize this software within a control architecture to tackle the classical software concerns of reuse, flexibility, validation, and real-time performances. In addition, the specifics of the targeted robotics domain must be preserved. This leads to the development of an architecture that handles all of the dynamics aspects without affecting the traditional software engineering aspects.

This problem typically falls under the *hybrid system* paradigm, because controlling a robot intimately merges discrete and continuous aspects. Hybrid systems are currently addressed by two distinct communities that have different beliefs about the difficulties. From the computer science point of view, the complexity generally lies in the size of the event-based transition system (related to the discrete aspects), while the dynamics issues that are considered inside the states are very simple. Therefore, the related analysis tools rely on various logic-based techniques. By contrast, the approach from the automatic control domain privileges models with algebro-differential equations describing complex dynamics and transition, while the number of states is small. The associated analysis methods are, for example, the differential systems with abrupt changes. To date, there has been little overlap between the two approaches. Indeed, the gap remains large and the problem encountered by the robotics community is that an autonomous robot system often falls precisely in this gap: complex dynamics on the one hand, complex event-based behavior on the other hand. Therefore, we have aimed to address these two issues with similar care, although there is presently no way of handling them formally in a unified approach. In this paper, we show how we consider this problem by carefully defining the adequate levels of analysis and the related modeling tools.

In addition, an autonomous robot belongs to the class of *critical systems*. Indeed, system safety issues are critical because any dysfunction of the system (often impossible to overcome while in operation) can lead to significant dam-

The International Journal of Robotics Research,
Vol. 17, No. 4, April 1998, pp. 338–359,
© 1998 Sage Publications, Inc.

age. For example, the consequences of a robot failing to repair a leakage in a nuclear plant can be dramatic, from both social and economic viewpoints. Thus, system safety issues play an essential role. Because hardware has improved both in terms of power and reliability, the major problems in this area now come from system's software components. In this case, "safety" means that the specification, programming, and implementation must be as correct as possible. This requires the application of validation approaches at all possible levels before releasing the system into operation.

These problems are not new; they have long been encountered in other areas, for example, in satellites, planes, and high-speed trains. However, they have appeared more recently in robotics because only a few complex systems have actually reached an operational level that justifies the related studies. The robotics community is now facing these problems and should therefore learn from the studies performed in these other areas. In particular, as shown in the remainder of the paper, our work takes advantage of techniques developed for safety purposes in the real-time domain.

As a consequence, we propose an approach to robotics programming that handles hybrid aspects as well as safety issues within a single dedicated framework implemented in a system called ORCCAD. Our system is organized around two key entities: an elementary task, called the ROBOT-TASK, and the ROBOT-PROCEDURE, which enforces a modular construction of complex missions. ORCCAD allows one to use various specification and validation tools, but it is also implementation oriented. By this, we mean that it automatically produces real-time code to download. It targets various domains, such as automated highways, autonomous mobile robots, underwater systems, and manufacturing manipulators. To date, it has been used on several complex missions performed by operational systems, for example, pipeline inspection by an underwater vehicle (Figure 1a), automatic vehicle driving (Figure 1b), indoor exploration (Figure 1d), and complex manipulation tasks (Figures 1c and 1e).

ORCCAD relies on a set of simple, well-defined, and original concepts:

1. The continuous-time control part (data flow) and the associate discrete-event handling are precisely merged at a single level. Therefore, we define a behavior in an original and formal way.
2. All the models of the discrete-event-based specifications rely, at all levels, on the synchrony assumption. Because we use a single synchronous language for their programming, a strong semantics is inherited, which opens the way to formal verification; furthermore, determinism is ensured.
3. Dedicated interfaces and a mission-oriented language are defined to hide the related technical aspects to a nonexpert user. Owing to their careful design, we can guarantee that the underlying formal models are preserved.

The outline of this paper is as follows. In Section 2, we present the basic definitions of the ROBOT-TASK and the ROBOT-PROCEDURE. Section 3 describes the specification tools. Section 4 presents formal verification aspects. Section 5 is oriented toward implementation issues. The approach is illustrated in Section 6 using an underwater application. We present our conclusion and trends for the future in Section 7.

2. Basic Concepts

2.1. Motivation

Traditional control architectures[1] proposed in the technical literature can be classified into three categories: hierarchical, behavioral, and hybrid. The first category adopts a top-down approach (Albus, Lumia, and McCain 1988). It highlights the supremacy of high-level control and restricts low-level horizontal communications. Hierarchical control also has poor flexibility, and it is not well adapted to the control of new-generation robots, which have to handle many sensors in reactive and reflex loops. By contrast, a bottom-up approach is adopted in the second category (Brooks 1986). This approach includes a group of communicating software modules known as "behaviors," which are based on models from psychology and organized in a "subsumption" architecture. Using this approach, it is clearly difficult to design high-level control to achieve nontrivial objectives. Furthermore, composition laws do not have enough semantics to allow safety issues to be considered in a clever way. The third approach, hybrid control, is the most recent (Bellingham and Consi 1990; Byrnes 1993; Alami et al. 1998; Schneider et al. 1998); it was developed to overcome the inherent defects in the previously mentioned approaches. A hybrid architecture permits the design of efficient low-level control, with an easy connection to high-level reasoning.

Our approach falls within this latter class. The following considerations have motivated this choice. When considering a control architecture through a top-down analysis, it is clear that the entities handled are further specialized as one draws near the lowest levels. In a symmetric way, a bottom-up approach progressively changes the considered abstraction level from the physical world to an end-user's view of a robot's mission. At the highest level, the viewed entities have a complex semantic content but only roughly involve the dynamic properties of the controlled system. In fact, here, the

1. Here, we do not consider approaches aimed at the rapid prototyping of control applications, such as the well-known `Matlab_simulink` development chain (MathWorks-Inc 1997). Although interesting features are proposed, they are not yet dedicated to the control and supervision of complex critical systems.

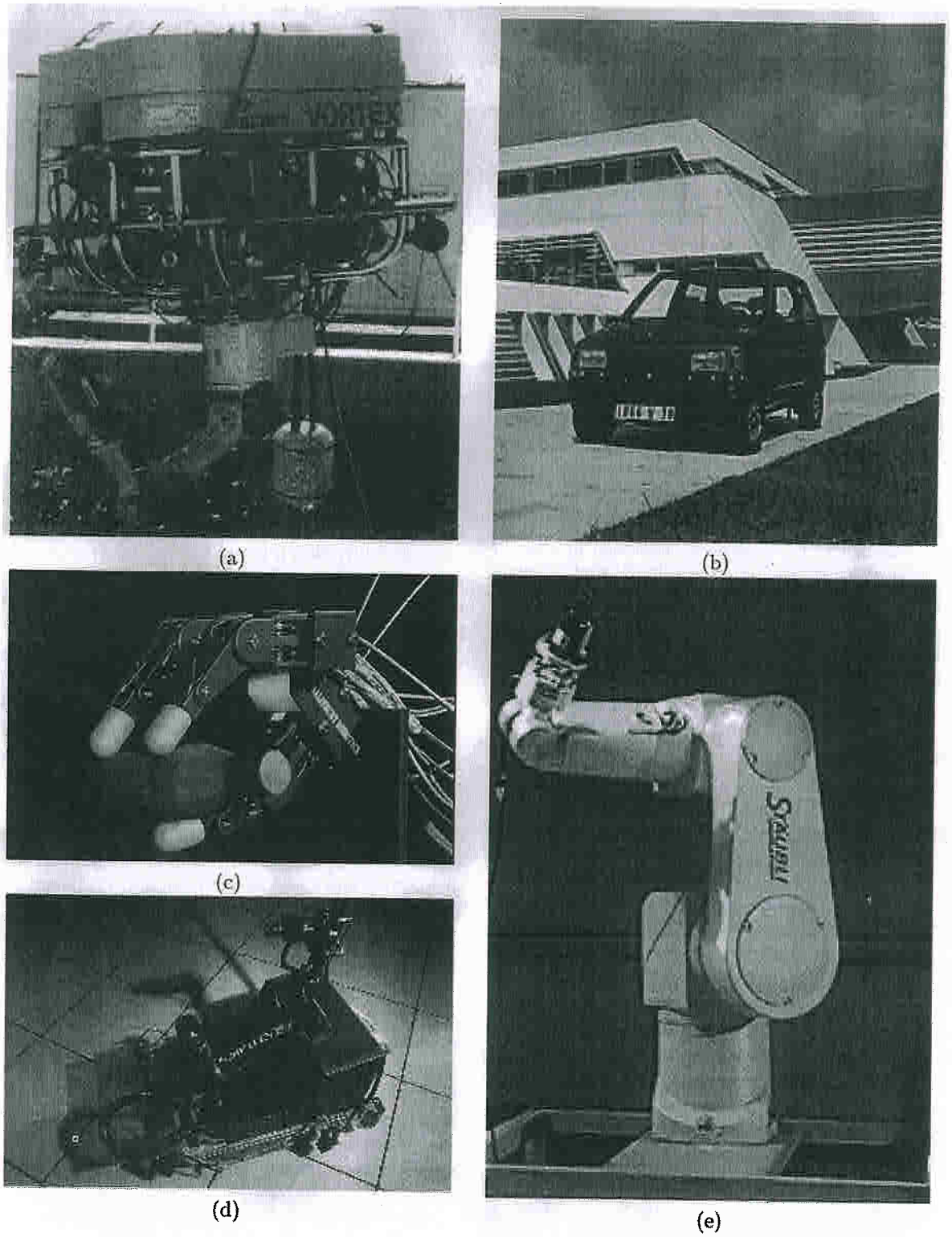

Fig. 1. Complex operational systems that have employed ORCCAD. (a) VORTEX/IFREMER underwater manipulator. (b) LIGIER/PRAXITÈLE electrical car. (c) The Salisbury hand. (d) The INRIA wheeled manipulator. (e) STAUBLI RX90 manipulator arm.

user mostly handles discrete events, global parameters, and symbolic data. In general, the related models are the ones used by planning and reasoning techniques. On the contrary, when staying close to the physical aspects of the robot and its controller, we must consider very local issues, such as events or numerical data, which reflect the dynamic behavior of the system interacting with the environment. At this level, the main models considered come from system theory, for example, continuous differential equations, discrete recurrent equations, and automata.

In addition, it is common to classify the different levels according to their time scales from the millisecond at the bottom to the hour at the top. A complementary way to view this problem is by using the nature of the time itself: continuous time at the physical level, discretized time in the control implementation, event-based time at the highest levels. Indeed, all time models have to coexist within the overall system, with some overlap of their influence areas.

However, in any efficient architecture there exists a particular intermediary layer where all time issues are of equivalent importance. In this case, the system is truly a *hybrid* one. When going down, the continuous dynamics and its control become the main issues; when going up, these latter aspects are masked, with the only remaining view being event driven and symbolic. It is clear that the design of the entities at the hybrid level is crucial for the system, for two reasons. First, they will be further assembled to build a complex application and therefore must include all the necessary information. Second, they have to run on the real system without any risk of damaging it. This means that the specified control must work correctly. In the next section, we present our approach to such a key entity, which we call the ROBOT-TASK.

2.2. *A Key Feature: The* ROBOT-TASK

The ideas that underlie the ROBOT-TASK concept are particularly simple.

1. We can now easily do many complex tasks using automatic control in robotics: visual servoing, force-based control, nonstationary control of nonholonomous vehicles, etc. Therefore, it is natural to offer the designer advanced approaches to the automatic control *at a higher level than is usually done.*
2. In a hybrid system, discrete events are closely related to continuous aspects. In a way similar to the automatic control approach, we can now handle the events associated with a system control and programming problem in a clever way, using, for example, the Ramadge-Wonham theory of discrete-event systems (Ramadge and Wonham 1989) or the synchronous approach (Benveniste and Berry 1991). We therefore propose to use such an approach coherently with the automatic control part. This leads us to apply advanced event-based techniques *at a level lower than is usually done.*
3. In real-time systems, either simple or as complex as an autonomous robot, user-defined *exceptions* must be handled. It is known that their specification as well as their harmonious handling in real time are difficult problems (and even more difficult when full preemptive actions are authorized). To efficiently handle exceptions, we propose to strongly constrain their design in the ROBOT-TASK and to process them hierarchically.

As a consequence, these assumptions led to the design of the ROBOT-TASK entity in ORCCAD, which is defined as the complete specification (Simon et al. 1993) of

- a control law in continuous time, which has an invariant structure along the whole duration of the ROBOT-TASK (which is assumed known, even if infinite); and
- a set of events to be received and emitted at the beginning of the ROBOT-TASK, during its execution and at its end, and the associated processing.

Thus, a ROBOT-TASK represents an elementary robotic action, just like the point-to-point free motion of an arm, or the sensor-aided grasping of an object, or the stationary lateral driving control of a vehicle. It can also be viewed as a formally defined behavior, which includes reactivity through the event-based specification along with true sensor-based control through its explicit continuous-time part.

To simplify the specification of a ROBOT-TASK and improve the reliability of the resulting system, several software-engineering issues have been adopted: simplicity of the specification by graphical user interfaces, modularity through an object-oriented approach, automatic real-time code generation, and the possibility of validation by hybrid simulation or formal verification. These aspects will be considered further in Section 5.

ROBOT-TASKs will later be detailed according to different views. In Section 3.1, the specification of ROBOT-TASKs data-flow aspects is detailed. In Section 3.2, its event-based part is presented in-depth. Verification issues are addressed in Section 4, and simulation aspects are illustrated in Section 5.4.

2.3. *The* ROBOT-PROCEDURE

Given a set of ROBOT-TASKs, the ROBOT-PROCEDURE entity is used to logically and hierarchically compose ROBOT-TASKs into structures of increasing complexity. The ROBOT-PROCEDURE allows for the description of robotic actions that range from elementary actions all the way up to full mission specifications. With no need for replanning, the ROBOT-PROCEDURE permits one to choose adequate sequences of

actions given observed situations and to manage predefined exception-recovery procedures when "weak" failures occur.

For instance, in the mission example of underwater manipulation operations that is detailed in Section 6, a typical ROBOT-PROCEDURE demands the alternative execution of vision-based and sonar-based ROBOT-TASKs (with the objective to stabilize the vehicle in front of an offshore structure) and, in parallel, the execution of point-to-point motions of the arm (to subsequently operate on this structure). Other examples concern indoor explorations, which coordinate vision-based ROBOT-TASKs with the elementary action of stationary lateral driving control of a vehicle (Pissard-Gibollet et al. 1995).

We define a ROBOT-PROCEDURE as the complete specification (Espiau, Kapellos, and Jourdan 1995) of:

- a main program, characterizing the nominal execution of the action, i.e., defining the logical and temporal arrangements of the ROBOT-TASKs and ROBOT-PROCEDURE; and
- a set of triplets (event, processing, assertion) that specifies the processing to apply to handle an event and the information to transmit to the planning level (if provided).

Here also, software-engineering issues form the basis of the ROBOT-PROCEDURE specification and implementation: A dedicated formal language is designed and presented in Section 3.2.3, logical verification aspects are addressed in Section 4.2, and these are integrated into the ORCCAD tools in Section 5.

3. Specification

The thorough specification of a robotics application requires the description of both the continuous (algorithm design) and discrete (logical events) aspects, as well as their real-time characteristics (execution periods, synchronization). In our approach, this specification is obtained thanks to the ROBOT-TASK and ROBOT-PROCEDURE entities previously introduced. As outlined below, the continuous aspects (cf. Section 3.1) are handled within the structure of a ROBOT-TASK, while the discrete aspects of the application (cf. Section 3.2) are mainly modeled within the structure of a ROBOT-PROCEDURE.

3.1. *Continuous Time and Data Flow*

3.1.1. *Generalities*

The ROBOT-TASK in ORCCAD is the minimal granularity seen by an end user at the application level and the maximum granularity considered by control-system engineers at the control level. It characterizes, in a structured way, continuous-time closed-loop control laws, along with their temporal features related to implementation and management of associated events. Here, data-flow and discrete-event aspects are mixed. The design of a ROBOT-TASK is achieved by using *modules* that exchange data through typed ports. They belong to three classes.

3.1.1.1. *The "Physical Resource" Class*

A module of this class is used to specify an interface between the ROBOT-TASK and physical entities (actuators, sensors, etc.) of the controlled process. It requires user-defined drivers and uses driver-type ports to characterize inputs and outputs in relation to the corresponding physical devices.

3.1.1.2. *The "Algorithm" Class*

This class is used to specify the algorithms necessary to compute the control law of the ROBOT-TASK. Modules in this class consume and produce data to and from the data-type input-output ports. An algorithm can be specialized by taking into account the particularities of the application domains. Although we have handled a number of domains (cf. Figure 1), we have placed the emphasis on handling robot manipulators. As stated in (Samson, Le Borgne, and Espiau 1991), a wide class of rigid robot-control laws can be derived more or less from a decoupling and feedback linearization approach. As detailed in Simon et al. (1993), this allows one to define a set of hierarchical subclasses, which can considerably help the user in the specification, including task functions, working spaces, dynamical models, Jacobians, and control methods. In this class, we also find *observers*. An observer is a module that takes data flows as inputs and produces events as outputs. These events are fed to the robot task automaton.

3.1.1.3. *The "Robot Task Automaton" Class*

The automaton specified in this module is the model of the transition system that locally controls the ROBOT-TASK and thus represents its local behavior. It requires the declaration of event-type input-output ports to characterize the related events as the preconditions, the postconditions, and the exceptions. The concept is further detailed in Section 3.2.

The design of a ROBOT-TASK can be achieved by creating (or selecting from available libraries) and then connecting instances of these module classes. As an introductory example, consider Figure 2 (see Section 6 for an elaborate ROBOT-TASK). It represents a computed torque-control law for a pendulum. The module PENDULE encapsulates the physical access to the pendulum, and the module ATRPENDULE represents the local behavior of the ROBOT-TASK. The other modules belong to the algorithm class: JOINTSTATE consumes the angle of the pendulum and, given the angle limits, signals when they are encountered by producing events toward the ATRPENDULE; Trajectory computes the desired trajectory to follow in the joint space; ERROR and CMD compute

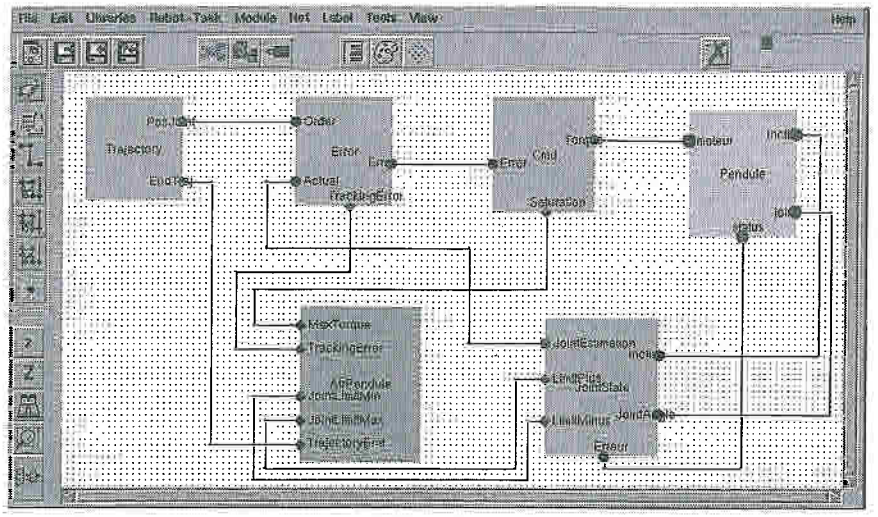

Fig. 2. The ROBOT-TASK structure for the control of a simple pendulum.

the torque with a PD control scheme. Meanwhile, CMD monitors excessive values of the torque to avoid saturation, and ERROR warns when the tracking error becomes too significant. In our environment, the types of modules and ports are graphically distinguished using different colors and forms (data are denoted by •, and events are symbolized by ◇).

3.1.2. *Real-Time Aspects*

So far, the specification of the ROBOT-TASK through modules has mainly been performed in continuous time. At this stage, *temporal properties* need to be added to the modules to take implementation aspects into account. These concern the discretization of time, computation duration, communication, and synchronization between the modules.

The implementation of a ROBOT-TASK is obtained by the translation of the overall specification (continuous and temporal properties) into a set of communicating real-time computing tasks, called MODULE-TASKs, each of which implements a part of the control law (one or more modules). Most of the MODULE-TASKs are periodic tasks. To ease the automatic code generation from the dedicated graphical user interface (see Section 5), the structure of a periodic MODULE-TASK is as shown in Figure 3. Such a structure clearly separates computations related to control-algorithms issues, communications related to implementation issues, and calls to the underlying operating system. The real-time attributes of a MODULE-TASK induced by the temporal properties of its modules are

- its (nominal or worst case) duration, which depends mainly on the algorithm, the programming language used to encode it (generally C or C++), and the target processor (the duration is estimated for simulation and schedulability analysis purposes);
- its activation period;
- the set of input and output ports of the MODULE-TASK and their associated communication protocols;
- the modules' priority, allowing a run-time scheduling by the operating system; and
- the assigned processor (in the multiprocessor case).

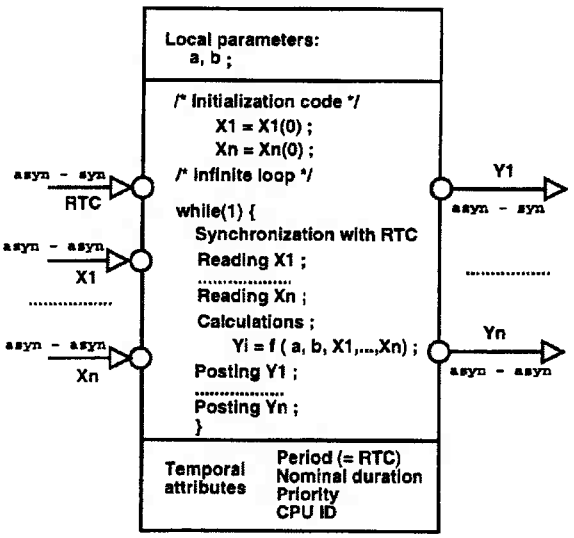

Fig. 3. The basic structure of a periodic MODULE-TASK.

In complex robotic control systems, there are often closed loops (in the sense of control laws) running at different sampling rates. For example, the data associated with feedback paths may be updated more frequently than the data associated with feedforward paths. Indeed, controller performance

may also be influenced by how tightly cooperative tasks are coupled with respect to their durations. As a consequence, ORCCAD makes different communication and synchronization mechanisms available to link pairs of MODULE-TASKs via their typed ports.

- ASYN-ASYN: Each task is running freely and the communication does not add synchronization;
- SYN-SYN: The first task to reach the rendezvous is blocked until the second one is ready;
- ASYN-SYN: The writer is running freely and posts messages onto its output ports at each period, and the reader either reads the message if a new one is available or is blocked until a new message is posted; and
- SYN-ASYN: The reader runs freely, and the writer is blocked up to the next request (except if a new one has been posted since the last reading).

Here, we are mainly concerned with closed-loop control; therefore, we shall often consider that the best data is either the last or the next available one. Thus, these protocols do not provide data queuing and generally allow loss of data, except for those that are used to send signals to the ROBOT-TASK and application automata to manage the logical behavior of control laws.

3.2. *Event-Based Part*

To build a complex robotic application, several control laws must be combined, each one achieving a given subobjective. The control laws are launched in sequence or in parallel, they might be aborted, and they must report their execution status. Thus, the control-law part of the ROBOT-TASK must be controlled. As a consequence, once specified, the continuous part of the ROBOT-TASK is encapsulated with event-based modeling. This encapsulation provides the cornerstone to building controllers for the ROBOT-TASKs (cf. Section 3.2.2.). Then, the whole application controller can in turn be specified as the logical composition of the ROBOT-TASK's discrete-event controllers (cf. Section 3.2.3).

3.2.1. *Requirements and Adopted Solution*

3.2.1.1. *Requirements*

Several formalisms can be envisioned to design a discrete-event controller (automaton-based description, general-purpose languages, etc). We chose to design our controller within the framework of *reactive systems* (Harel and Pnueli 1985) theory. A reactive system maintains a continuous interaction with its environment, at a speed that is determined by its environment, by emitting signals as responses to its stimuli. Consequently, the discrete-event part of the concept of behavior, which traditionally lacks models permitting formal composition, can be formally defined as the sets of allowed sequences of input-output signals that constitute the interface of a system with its environment.

A robotic system is a reactive system that is in continuous interaction with its environment and should satisfy the following requirements:

- Possibility of concurrency: The concurrency between the robotic system and its environment must be taken into account. Furthermore, it is often convenient and natural to consider such a system as a set of parallel components that cooperate to achieve a desired behavior.
- Real-time performances: The real-time constraints derive from the obligation to react at the rhythm imposed by the environment surrounding the robotics system. Also, the response time must be compatible with the dynamics of the controlled system.
- Determinism: A system is deterministic if the same set of (ordered) inputs produces the same set of (ordered) outputs whenever the control system is executed. With such an observational point of view, a robotic system's behavior becomes predictable and reproducible, a critical feature that improves the reliability of robotics systems. Clearly, their design, analysis, and debugging become much easier.

3.2.1.2. *The* ESTEREL *Language*

To contend with such requirements, it is natural to focus on *synchronous languages* that are especially designed to tackle reactive systems, and subsequently ease their specification, offer validation possibilities, and generate efficient executable code. Among them, we have chosen the ESTEREL language (Berry and Gonthier 1992; Berry 1997) because of its adequacy in handling control problems. Thus, the reactive controller of a robotic system can be expressed using ESTEREL once its logical behavior has been well defined.

3.2.2. *The Event-Based Part of the* ROBOT-TASK

Defining an elementary action with a continuous-time control law is not sufficient. It is also necessary to specify how and when this control can be activated and how and why it terminates; also, we must be able to take into account various types of errors that are liable to occur at run time.

Hence, we associate a transition system with the control law that locally controls the ROBOT-TASK. It handles three types of logical events.

- Preconditions—allow the ROBOT-TASK to actually start. These are synchronizing events or signals emitted by sensors. A temporal watchdog can be associated with each precondition;

- Exceptions—are emitted exclusively by observers (we will give more details of this concept below); and
- Postconditions—term not used in the usual sense for computer science. In this context, postconditions gather events required to terminate a ROBOT-TASK and events that are emitted at the end of the task. A watchdog can be associated with postcondition waiting.

To simplify the exception specification and make processing more reliable, the exceptions are further divided into three predefined classes, corresponding to different decision levels.

- Type 1—an exception belonging to this class has a range limited to the ROBOT-TASK itself. In other words, it can be handled inside an algorithmic module of the ROBOT-TASK. For instance, a reasonable but nevertheless too-large tracking error can lead to a demand of gain increase within two limits, as done by Kapellos et al. (1995). As another example, consider a biped robot in stationary walk. From one step to the next the control is the same, but the leg index switches from left to right. This problem can be solved using a single control scheme and having a Type-1 event emitted at each phase transition.
- Type 2—here, the detected problem is not fatal, but local handling is no longer possible. The ROBOT-TASK should be stopped and the control transferred to the upper level, which must know the decision to make. This is, for example, the case when a robot manipulator enters its joint-limits area or reaches a singularity. It also occurs if a nonholonomous vehicle appears unable to achieve some maneuver by feedback alone.
- Type 3—in this case, the failure is serious (actuator breakdown or water leakage, for example), and the system has to be driven to a safe position while emergency procedures can be undertaken. Again, the control is performed at a level higher than that of the ROBOT-TASK.

The reception of all these events rhythms the evolution of the action according to a predefined scheme: Roughly speaking, the satisfaction of the preconditions leads to the activation of the control law. If a specified exception occurs during an execution, it is handled according to its type; the reception of the postconditions implies the ending of the action.

Using this typology of events, a user can easily specify the event-based part of any ROBOT-TASK. Furthermore, because of the predefined behavior, the corresponding ESTEREL program can be automatically generated with no need for the user to effectively encode it "by hand." This automatic generation also hides the additional signals necessary to link this discrete controller with, first, the real-time surrounding software (cf. Section 5) and second, with the controller of the application in which this ROBOT-TASK is used (cf. Section 3.2.3, below). This behavior can be compiled into a finite automaton as illustrated in Figure 4 because of the mathematical semantics at the basis of the implementation of the ESTEREL language. For example, the `EndKill` signal is used for a connection with the real-time software, while the `Abort_Local` signal is connected to the application controller.

3.2.3. *The* ROBOT-PROCEDURE *and the* MAESTRO *Language*

In the ROBOT-TASK, the predominant concern is to address the continuous automatic control aspects relating to their realization. On the contrary, the ROBOT-PROCEDURE (Espiau, Kapellos, and Jourdan 1995) aims to coordinate the ROBOT-TASKs to complete a mission. This puts the stress on reactivity handling. Here also, reactivity is formally handled within the context of *reactive systems* theory, as described next.

As with the ROBOT-TASK scheme, a systematic and structured way of specification is adopted to formalize a ROBOT-PROCEDURE. However, unlike the ROBOT-TASK, the ROBOT-PROCEDURE is a stand-alone entity that propagates only unrecoverable errors outside its scope. It comprises:

- a set of preconditions that need to be fulfilled before the main program starts;
- a main program (nominal execution of the action) composed of ROBOT-TASKS, ROBOT-PROCEDURES and conditions;
- a set of postconditions that induce the end of the procedure;
- a set of reaction rules (and eventually assertions) to process every exception; and
- a predefined behavior for the logical coordination of the previous items.

The exception events are either reported by the participating ROBOT-PROCEDURE. These elements are coordinated as follows: the main program is activated after satisfaction of the preconditions and normally ends when the postconditions are satisfied. If an exception occurs, this nominal execution is aborted and replaced by the specified handling program.

To specify the ROBOT-PROCEDURE, we propose the MAESTRO language, a well-defined high-level specification formalism (Coste-Maniere and Turro 1997). Using this language, the user reasons in terms of actions (namely, ROBOT-TASKs and ROBOT-PROCEDURES) to be arranged through various types of intuitive, classical, and/or reactive operators. The specification is independent of the target model into which it can be translated and hides many programming tricks, such as additional signals required by a modular decomposition.

MAESTRO offers several kinds of constructs to specify the main program and conditions, as illustrated in Figure 5. They comprise:

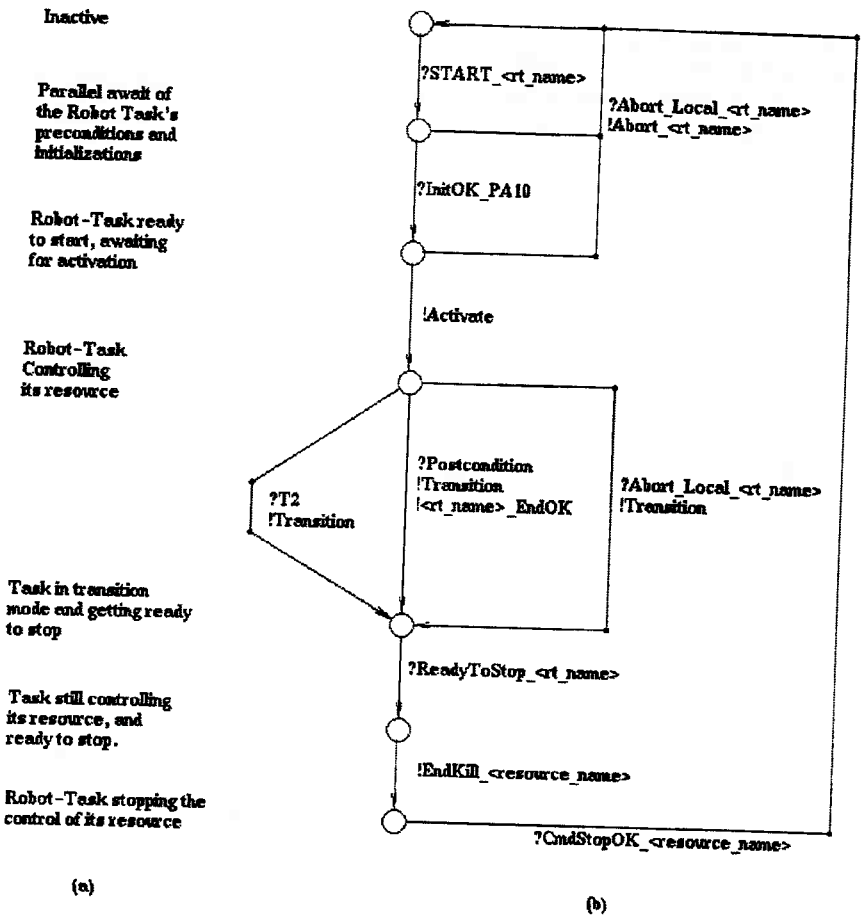

Fig. 4. The ROBOT-TASK automaton: left, comments: right, real signals.

- requests to launch ROBOT-TASKs or ROBOT-PROCEDUREs;
- classical control structures such as SEQUENCE, PARALLEL, and LOOP to compose the ROBOT-TASKs and ROBOT-PROCEDUREs; and
- a preemptive statement, DO/UNTIL, which aborts its body execution if a particular event occurs. This event may characterize the execution of a ROBOT-TASK (an exception or its end, for instance) or be produced by an external observation function. The DO/UNTIL construct may also include a reaction to be executed only if the condition actually holds.

Variables and types are provided: The ROBOT-TASKs can be parameterized at launch time or return data upon termination. Because the aim of ROBOT-PROCEDUREs is to specify a behavior, no computation is allowed on variables; their purpose is to store the value returned by a given ROBOT-TASK. In turn, this value may be used to parameterize another ROBOT-TASK.

Once the mission has been specified with MAESTRO, it is necessary to produce a controller that can be both verified and executed. As with the ROBOT-TASK, the controller is automatically generated into the ESTEREL language. Thus, the direct benefits of using ESTEREL are accessible with no need for the user to master its programming. Of course, the complexity of the ESTEREL controller is greater than the high-level MAESTRO specification written by the user because it requires merging the following building blocks:

- a generic skeleton that expands the interface of the controller with the real-time environment and composes, in parallel, all the controllers of the ROBOT-TASKs and ROBOT-PROCEDUREs involved in the mission. In addition, the sequencing protocols required to guarantee good dynamics properties of the system are added here (e.g., to guarantee that an actuator is not addressed by two control laws simultaneously or to minimize the time during which a physical resource is not controlled [Simon, Kapellos, and Espiau 1996]);
- the ROBOT-TASKs controllers, which are built by expanding abstract views that convey all the necessary information to individually control the execution of the

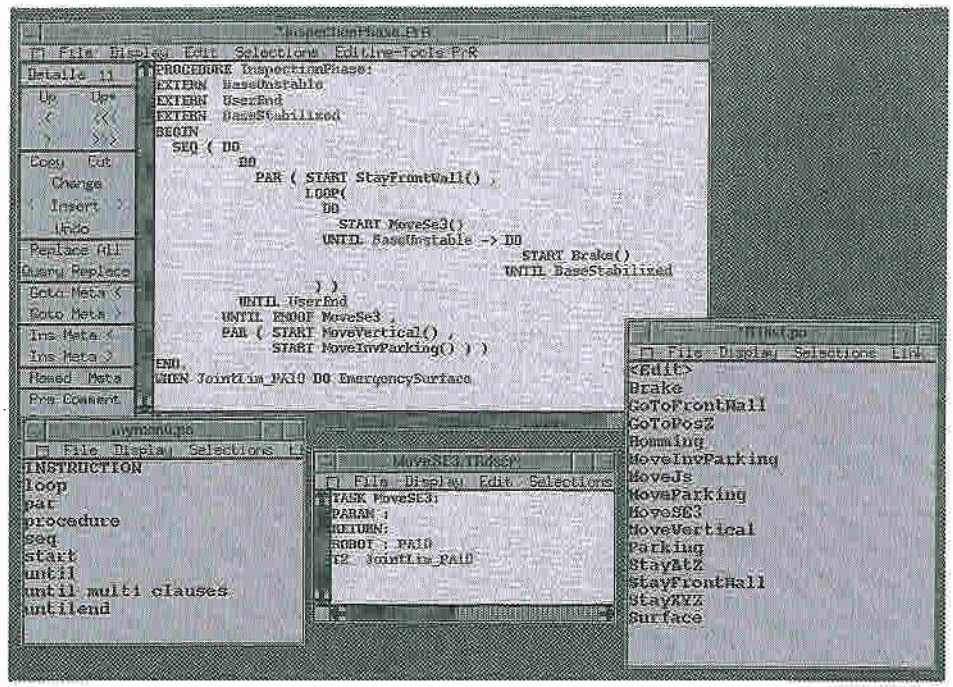

Fig. 5. The MAESTRO program, illustrating the programming of an underwater mission.

ROBOT-TASKs. These result from the user's specifications of the ROBOT-TASKs: preconditions, exceptions, postconditions, the type of data produced and consumed by the action, and the physical resource to which the ROBOT-TASK is dedicated; and

- the ROBOT-PROCEDUREs controllers, which are obtained by expanding the MAESTRO specification. Each operator has its own translation into ESTEREL, thus defining its operational semantics.

4. Formal Verification

Formal verifications and *hybrid simulations* are two complementary methods available to validate a robot controller.

With the formal verification, we can formally prove that the system will behave as expected. From the automatic control point of view, this requires, for example, obtaining theoretical results of stability and convergence. For instance, concerning the software aspect, it is necessary to formally verify that the critical parts of the system will not have deadlocks or that they will react to successive events exactly as specified (cf. Section 4.2). Furthermore, the introduction of discretization and synchronization issues between pieces of code can have important consequences for the resulting behavior, which again requires the use of formal analysis procedures (cf. Section 4.1).

There are still theoretical issues with formal methods that remain unresolved. Therefore, further investigation is necessary to study the behavior of the controller using simulations. Our specific approach to the simulation of hybrid systems is described in Section 5.4.

4.1. *Verification of the Synchronization Skeleton of a* ROBOT-TASK

We are interested here in examining the effects of some temporal issues, which appear inside the ROBOT-TASK specification, on its overall behavior (Simon, Castillo, and Freedman 1995). We consider the set of MODULE-TASKs that constitute the ROBOT-TASK, without taking into account scheduling aspects. Verifications can then be carried out using the *synchronization skeleton* of the ROBOT-TASK, i.e., a description of the temporal and synchronization constraints that exist inside the set of MODULE-TASKs. This description takes into account the MODULE-TASK period and duration, the ordered list of associated communication ports, and the kind of synchronization associated with connected ports.

Design inconsistencies may arise in several ways. *Structural* deadlocks are owing to the synchronization structure itself, independent of the numerical values of temporal attributes. In addition, badly chosen numerical values of temporal attributes, such as task period and duration, may lead to *temporal inconsistencies* and unsafe (e.g., nonperiodic) behavior of the ROBOT-TASK, even if it is free of structural deadlocks.

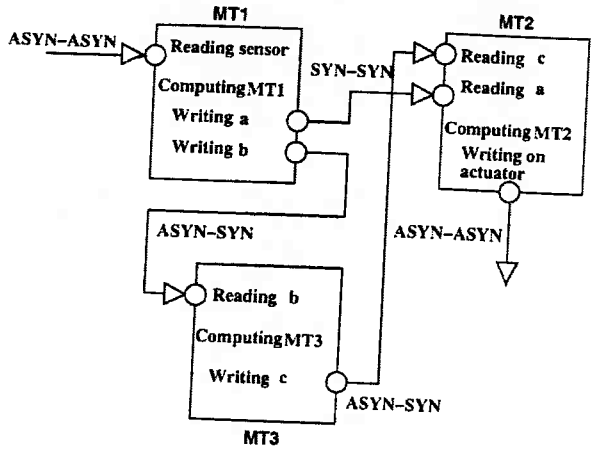

Fig. 6. A structural deadlock due to the ordering of the input ports of MODULE-TASK 2.

Structural deadlocks may be associated with circularity in inter-MODULE-TASK communication. Other structural deadlocks are more subtle and emerge when the order in which a MODULE-TASK communicates over multiple ports is incorrectly specified. In the example shown in Figure 6, MODULE-TASK 2 blocks while waiting to "read" from MODULE-TASK 3, which blocks MODULE-TASK 1 from "writing," which in turn blocks MODULE-TASK 3 from "reading." Clearly, this structural deadlock may be simply eliminated, but identifying such deadlock on the graphical user-interface screen is usually difficult because of the complexity of some control laws when many synchronizing links are interleaved.

Therefore, we need modeling and analysis tools to automatically check for deadlock avoidance in the network of synchronized MODULE-TASKS. Among various modeling tools, Petri nets theory (Murata 1989) provides a simple and efficient way to carry out this task. Besides modeling capabilities, a Petri net can be analyzed in a formal way to obtain information about the dynamic behavior of the modeled system.

As shown in Figure 7 on the left, the sequential behavior of the simplest periodic MODULE-TASK (reading an input port, performing a calculation, or writing to an output port) may be modeled by a condition/event Petri net with one transition associated with each input or output port and a timed transition associated with the MODULE-TASK computation. Periodic MODULE-TASKS are synchronized with a real-time clock that is also modeled by a Petri net. Using these Petri net models for the MODULE-TASKS and synchronization protocols, we are able to define the Petri net model of the set of synchronized MODULE-TASKS illustrated in Figure 6.

Because each place has just one input transition and one output transition, the MODULE-TASK behavior is deterministic, and the resulting Petri net is a so-called connected marked graph, which exhibits a useful structural property: In a marked graph, the token count in a directed circuit is invariant under any firing (Murata 1989). Therefore, the marked graph is live if and only if the initial marking places at least one token in any direct circuit of the Petri net.

A simple algorithm using linear programming with integers (Murata 1989) allows us to compute the circuits and therefore draw conclusions about the liveness of the Petri net model of the set of MODULE-TASKS. This algorithm has been encoded inside a C package, allowing us to automatically build the Petri net model from the graphical user interface of ORCCAD, to search for the directed circuits, and to perform the liveness analysis.

Using this package, the directed circuits of Figure 7 may be identified. If we consider the initial marking corresponding to all three MODULE-TASKS waiting for activation (places p1, p6, p11 marked), the reader may verify that one circuit (the dotted line) has no token and therefore that the marking is not live.

Temporal inconsistencies can occur in complex networks of MODULE-TASKS, in which the interleaving of computing paths can hide, for example, multiple synchronization. As a result, the beginning of the execution of a MODULE-TASK can be delayed at each activation of the control law, leading to missed deadlines.

Such a situation can be checked by observing the evolution of the reachability graph of the Petri net until either the normal completion of the ROBOT-TASK or a deadlock occurs. In fact, a safe timed Petri net will reach a steady-state (periodic) behavior after a finite time without being trapped in a sink place. Searching for this steady-state behavior consists of going through the reachability graph until a marking already visited is once again reached. This operation is usually faster than a systematic exploration of the reachability graph up to the completion of the ROBOT-TASK.

As an alternative, we note that algebraic methods for analyzing timed Petri nets are now emerging e.g., algebra on the (max, +) semi-ring (Baccelli et al. 1992). Indeed, since our basic Petri net models are timed-event graphs, they can be translated into a linear model in the (max, +) algebra, thus allowing us to exploit the (max, +) counterparts of classical concepts in system theory, such as state-space recursive equations, transfer functions, or feedback loops. In this way, it would be possible to analyze both the transient and asymptotic behavior of our multitasking controller implementations within the framework of discrete-event dynamic systems and to evaluate their performance, e.g., as expressed by token-production rate.

4.2. *Logical and Temporal Verifications*

Both logical and quantitative temporal analyses can be performed on the ROBOT-TASK and the ROBOT-PROCEDURE. Logical analysis is performed using behavioral methods (Boudol et al. 1990) and theorem-proving techniques (Kahn

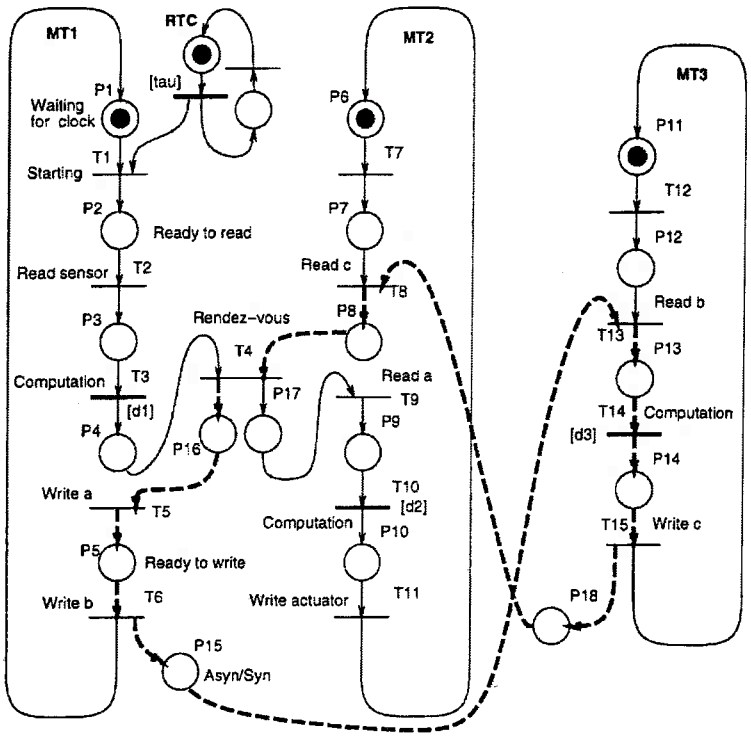

Fig. 7. A Petri net model of the three MODULE-TASKS with deadlock.

1987), while real-time model-checking methods (Henzinger et al. 1992) are used for temporal verification (Espiau, Kapellos, and Jourdan 1995).

The paradigm of specification-translation provides two representations, which are semantically equivalent. Thus, it becomes possible to progressively apply the verification process at every step of the design process, from the most abstract form (i.e., a MAESTRO program) to the most detailed form (i.e., an ESTEREL program and its corresponding automaton). As a consequence, the "correctness" of the controller is ensured at each step of the translation. Two classes of properties can be distinguished:

- "structural properties" that can be checked independently of a particular application. (Some are enforced by construction and others can be checked by the user of the ORCCAD software environment) (see Section 5); and
- properties related to the coherency of the specification with respect to application-dependent requirements that can be formulated interactively by the end user.

It is well known to practitioners of formal verification methods that the verification process is all the more well supported if it is integrated early in the development. This preoccupation led us to design the MAESTRO language (cf. Section 3.2.3) in a formal framework that enables one to apply theorem-proving methods to the corresponding programs.

Special emphasis was put on the structure of the generated ESTEREL code, for which the main elements are now recalled (a complete description can be found in [Espiau, Kapellos, and Jourdan 1995]). The generated controller is designed as the parallel composition of a set of ESTEREL modules (denoted by boxes in Figure 8), which are synchronized by systematic signal exchanges (arrows interconnecting boxes). Modules labeled *action_i* represent the controller of each action (ROBOT-TASK or ROBOT-PROCEDURE), while the module *coordinator* deals with their synchronization; the exchanged signals indicate the significant instants of their evolution (start, end, and abort). As explained below, the detection of these instants is relevant to the verification process, for example, when the property to verify is expressed in terms of actions (e.g., check that the execution of two actions cannot overlap).

In the sequel, we present the nature of the most significant results associated with the verification process as applied to the design of an application.

4.2.1. Structural Properties

4.2.1.1. Structural properties of the ROBOT-TASK

We proved the correctness of the predefined generic structure of the logical evolution of a ROBOT-TASK and, subsequently, that the behavior of a ROBOT-TASK is correct independent of

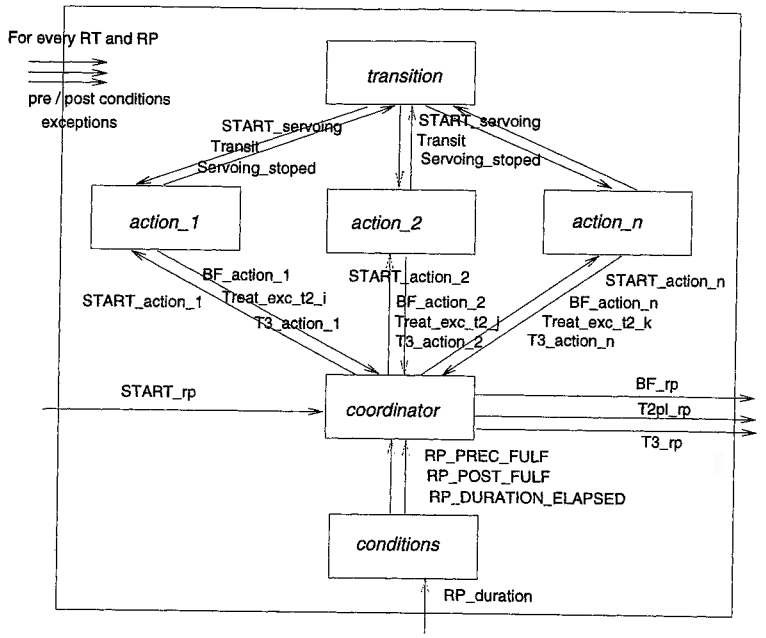

Fig. 8. The internal structure of a ROBOT-PROCEDURE.

a specific that (i.e., whatever the types and the number of events used in the ROBOT-TASK automaton) (Espiau, Kapellos, and Jourdan 1995). It is "nonblocking," and it adequately handles the different exceptions and abortion requests in any internal state. Therefore, the basic objects (i.e., the ROBOT-TASKs) composed for the design of complex robotic actions are guaranteed to have "good" properties.

4.2.1.2. *Structural properties of the* ROBOT-PROCEDURE

The user is more freely involved in the design process of the ROBOT-PROCEDURE than in ROBOT-TASK since it is at this step that one specifies the logic of the application. Hence, all structural properties cannot be enforced by construction.

However, critical parts of the behavior are hidden from the end user, and the corresponding structure of the ROBOT-PROCEDURE is constrained to be generic, thus allowing us to obtain partial results on structural properties. In particular, access to the actuators of the controlled physical resource is guaranteed to be nonconflicting when switching from one ROBOT-TASK to another and is as fast as possible to maintain the stability of the system (cf. the existence of a module labeled "transition," see Figure 8). As an example, consider the automaton shown in Figure 9. This illustrates the sequential execution of five different ROBOT-TASKs for which the start and stop commands are symbolically renamed `START_servoing` and `Servoing_stopped`. We observe that for each task switching, the stopping and starting signals are correctly ordered and the transition takes place in a single automaton transition. It thus minimizes the latency when switching from one control law to another.

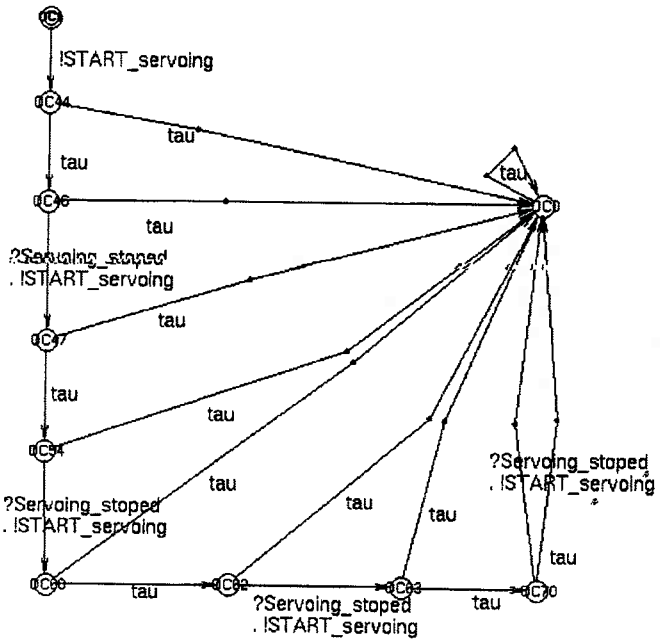

Fig. 9. An example of a transition between two ROBOT-TASKs.

Furthermore, incoherent specifications are detected during the specification of a ROBOT-PROCEDURE with MAESTRO. The generic structure of the actions and the semantics of the MAESTRO's operators are expressed with natural semantics rules. They enable us to check the user's statements with

the actions and their scope. For example, a program is only considered valid when

- all exceptions are resolved;
- possible deadlock or endless wait sources (e.g., a loop statement or ROBOT-TASK without postconditions) are encapsulated by a preemptive construct (i.e., an until statement); and
- variables are coherently used (type checking and the detection of variables not assigned).

The end user is thus provided with automatic and transparent preliminary verification of the coherence of the MAESTRO programs. The origins of the errors detected while performing verification of the MAESTRO specifications are directly expressed in terms of actions and conditions (e.g., "ROBOT-TASK_i not encapsulated in a pre-emptive statement"), and subsequently are directly understandable by the user. Once the MAESTRO specification has been checked according to the above properties, the same properties are subsequently ensured on the generated ESTEREL controller because the generated ESTEREL code exactly matches the semantics of each operator. Then, more extensive and detailed verifications can be performed on the ESTEREL formalism (as explained in the sequel).

4.2.2. *Coherence with the Application-Dependent Requirements*

The conformance of the ROBOT-PROCEDURE behavior with respect to the mission constraints can be verified interactively. These constraints have to be expressed in a generic way, as relationships between actions, between events and actions, or between events. In the verification process, the user must indicate the relevant set of events and/or actions related to the constraints to be checked. A global automaton is then abstracted and reduced by well-chosen signals among those considered in the translation of the ROBOT-PROCEDURE into its automaton model via ESTEREL. For example, the following property can be formally proven to be true during the execution of the underwater mission of pipeline inspection: "The arm will move to perform the inspection if and only if the base is stabilized." This process will be further detailed in Section 6 (cf. Figure 14).

5. Implementation: ORCCAD Tools

The complete design of a robot controller requires the integration of all the theoretical concepts introduced in the previous sections. The objective is to provide, in a coherent environment, a set of connected dedicated tools that fully benefit from these theories (efficiently applied to tackle all the specificities of the robotics domain) but hide their implementation details. The purposes are thus to enforce a separation of concerns, insulating the user from unnecessary detail and severing machine dependencies. Thus, the ORCCAD environment has been designed as a toolbox, where in-house and existing established tools are coherently organized around the ORCCAD object-oriented kernel (see Figure 10).

5.1. *The* ORCCAD *Kernel*

The main entities described in Section 3 constitute the ORCCAD kernel. It is implemented in an independent library that gives access to a C++ class hierarchy. The instantiation of all of these entities allows the user to describe a complete robotic application.

The module is the elementary unit at the basis of the implementation of the ROBOT-TASK. Once designed through the ORCCAD graphical user interface, modules are stored in a library according to their types, which can be *algorithmic, physical resource,* or ROBOT-TASK *automaton.*

A ROBOT-TASK in the ORCCAD kernel consists of a set of interconnected modules, to which temporal attributes have been associated (cf. Section 3.1). The synchronization mechanisms on the ports are also specified. Coherence tests are performed during the specification phase also carried out with the ORCCAD graphical user interface. When the specification is "correct," the abstract view of the ROBOT-TASK is produced. It includes the set of typed events involved in the reactive management of the ROBOT-TASK, the concerned physical resource, and its parameters.

A ROBOT-PROCEDURE is a collection of ROBOT-TASKS, and its logical composition is specified with the MAESTRO language. MAESTRO has been prototyped using the CENTAUR generic interactive environment (Borras et al. 1988) and handles the ROBOT-TASKS through their abstract views. A user-friendly structured editor is directly available to the end user to program the application with simple mouse selections (cf. Figure 5). Internal semantic validation is performed on the specified ROBOT-PROCEDURE within the CENTAUR environment before the ESTEREL code is generated for further verification or execution using theorem-proving techniques.

So far, the MAESTRO language is still in a prototype version. Thus, the logical composition of the ROBOT-TASKS must still be partially encoded directly in ESTEREL through a graphical user interface.

5.2. *The User Interface*

A graphical user interface provides a convenient way to specify a robot controller according to the ORCCAD methodology. The graphical part, written using the ILOG-VIEWS (Ilog 1995) class hierarchy, is the front end of the kernel; the objects required for the specification of a robotic application are instantiated through this interface. For example, modules and ROBOT-TASKS are instantiated as illustrated Figures 11 and

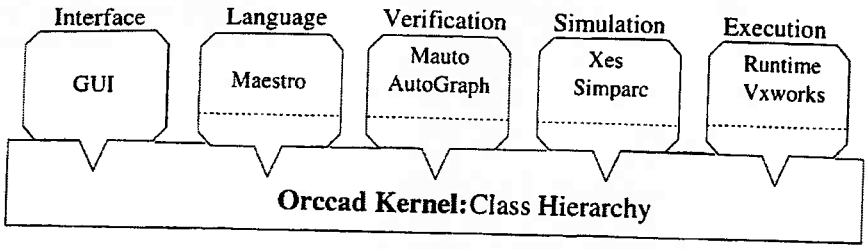

Fig. 10. The ORCCAD toolbox.

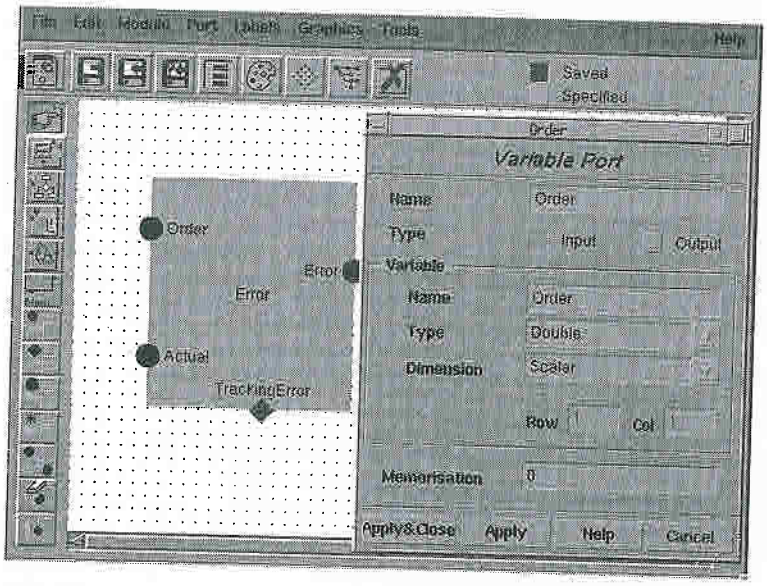

Fig. 11. The ORCCAD module editor.

13. Tools for verification (cf. Figure 12), simulation, or execution can also be reached from this interface.

5.3. *Verification of the* ESTEREL *Controller*

Once the application controller has been built, verifications must be performed to improve its reliability (cf. Section 4.2). Thus, the next step consists of providing the end user with a simplified interface to the adequate verification tools. The complexity of such a process is hidden, and only the relevant results are provided. This facility is offered in the ORCCAD environment (see Figure 12) to perform the observation of the controller automata, which results from the compilation of the associated ESTEREL program. The system transforms criteria expressed by the user in terms of conditions and actions into observation criteria that is understandable in the grammar of the MAUTO (Roy and de Simone 1990) and FC2TOOLS (Bouali et al. 1996) verification environments (Fernandez et al. 1996). These criteria, which appear obscure to nonspecialists of formal verification, can be automatically generated using the abstract views of each ROBOT-TASK and the generic structure of the ROBOT-PROCEDURES. Then, the observation of the controller automaton with respect to this criterion is carried out in MAUTO or FC2TOOLS. The resulting reduced automaton can be displayed in a graphical representation readily understandable to the user for interpretation. Afterward, it may be necessary to respecify parts of the application that do not meet the criteria to overcome defects.

5.4. *Simulation*

The ORCCAD environment facilitates the access to tools dedicated to simulations dealing with logical only and combined logical and continuous (i.e., hybrid) aspects of the ROBOT-TASKS and ROBOT-PROCEDURES.

5.4.1. *Logical Simulation*

With the XES simulation environment (Berry 1997), the user can interactively run the resulting ESTEREL controller through an intuitive graphic interface. Thus, one can manually trigger input events, such as preconditions, exceptions, and postconditions, and subsequently check that the sequence of output events generated (e.g., launch or stop the commands to the

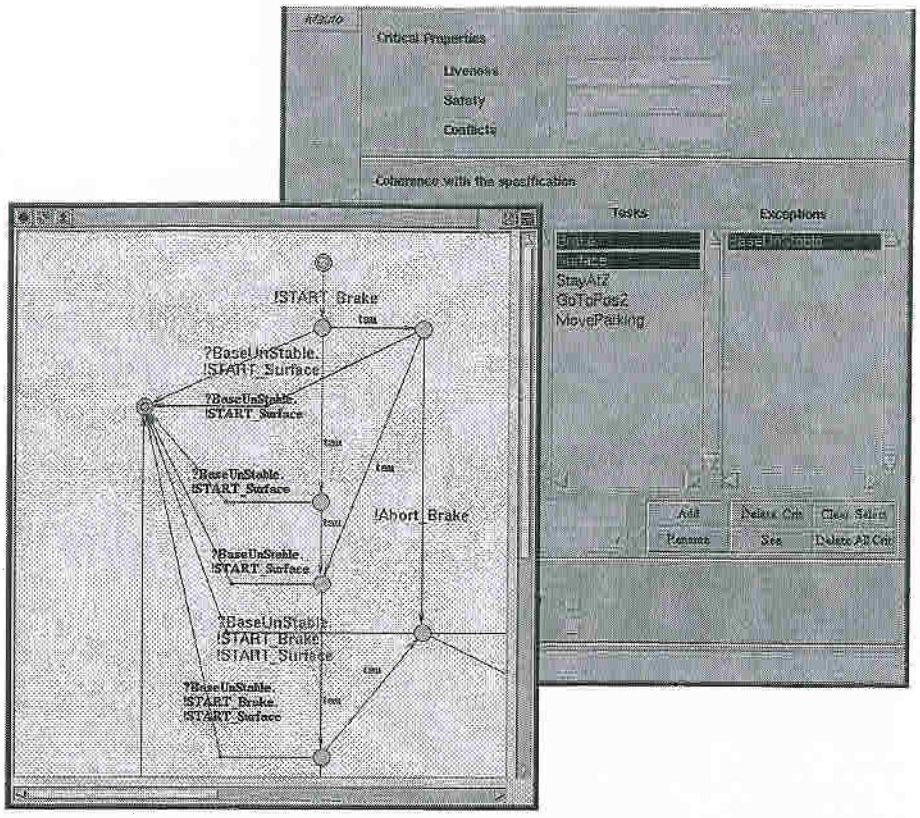

Fig. 12. The ORCCAD verification process.

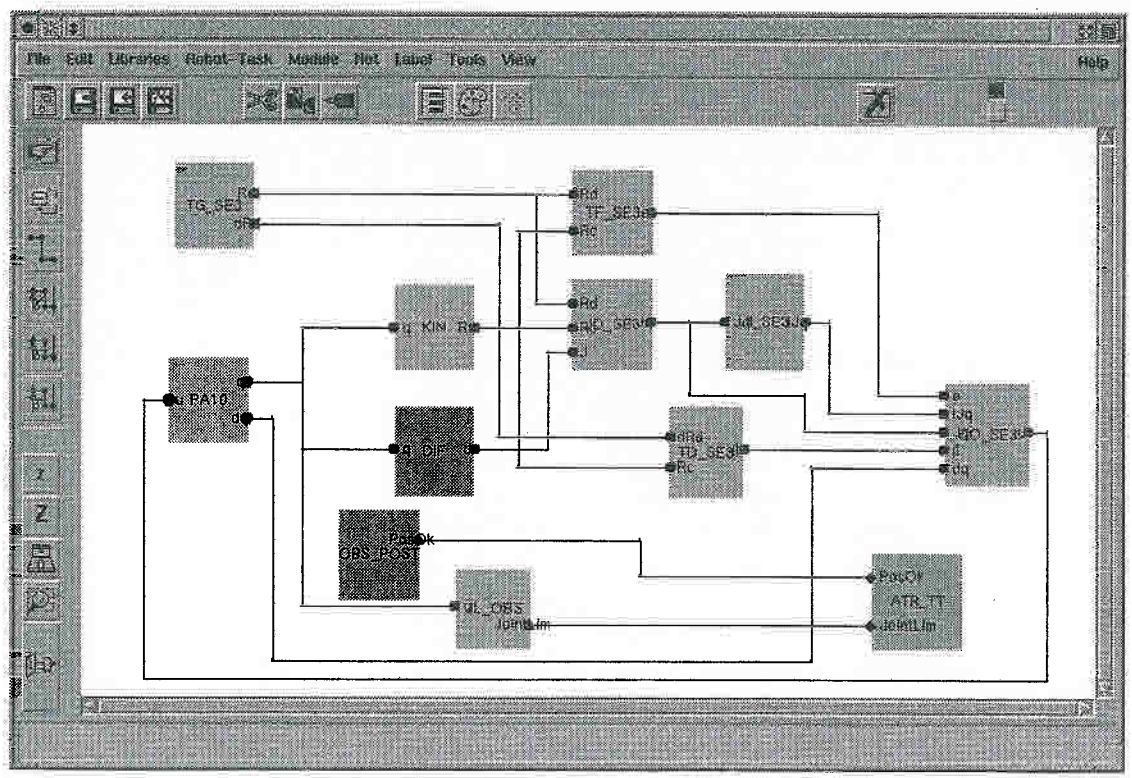

Fig. 13. The ROBOT-TASK structure for the control in the operational space of the PA10 arm.

physical device) effectively reflects the expected logical behavior.

5.4.2. *Hybrid Simulation*

Owing to the present lack of tools for fully predicting and formally analyzing the performance of a nonlinear system controlled by a multirate architecture, a global simulation plays a very important role in the design and test of hybrid systems and remains an essential component of an integrated environment. A main characteristic of the SIMPARC simulation package (Astraudo and Borrelly 1992) is that it allows one to simulate both the plant dynamics and important temporal characteristics of the controller (such as sampling rates and communication delays), including the basic features of real-time operating systems.

The main objects modeled by SIMPARC are:

- a description of the controller, using a library of hardware components including processors, memory, real-time clocks, converters, etc. Numerical attributes such as converter resolution or bus-communication delays, may be set in these components through the dedicated graphical user interface of SIMPARC (see Figure 15);
- a model of the physical system under control, which must be given as a set of ordinary differential equations. During a simulation, this model is numerically integrated coherently with the evolution of time in the controller;
- the basic features of a real-time operating system, such as real-time tasks, shared memory, semaphores, and message queues;
- some ray-tracing-based models of sensors, which allow us to simulate acoustic sounders, multibeam sonars, or video cameras; and
- the robot environment, which is modeled using facets.

Owing to the simulation model of the operating system, the control code generated by ORCCAD can be mapped on the SIMPARC simulated architecture in a structure similar to the one of the downloadable code.

As a result, simulations of the whole hybrid system can run under SIMPARC. Although such simulations have already been conducted, automatic generation of the simulation program is not yet available in the current version of the ORCCAD environment.

5.5. *Execution*

The execution component of the system translates a ROBOT-PROCEDURE specification into a real-time C++ code for a given robotic system. Achieving this requires two steps. First, ORCCAD produces code that is independent of the real-time targeted environment. Then, this code is further integrated with its run-time libraries. The whole robot-control software is made of three parts:

- a set of real-time tasks, usually periodic and synchronized according to consumer/producer protocols, dedicated to the computation of the continuous aspects of the system (mainly the control laws);
- the discrete-event automaton directly generated in C by the ESTEREL compiler; and
- the interface between the two previous parts. Its role is double: it transforms meaningful discontinuities occurring in the continuous aspects of the system into the discrete events that are fed to the automaton; vice versa, it links the appropriate requests made to the real-time layer with each output event produced by the automaton.

The ORCCAD execution subpart (see Figure 10) consists of an application-independent run-time library and the C++ code generator used to automatically implement the robot controller specified by the user. The target system supported by the run-time library is the real-time operating system VXWORKS 5.3 (WindRiver Systems 1995). So far, the generated code is mono-processor and mono-rate.

6. A Simple Example: An Underwater Mission

To illustrate the concepts described in the previous sections, we have chosen to guide the reader through the design of a complex underwater application. Further details on this application can be found in Simon, Kapellos, and Espiau (1996).

6.1. *Experimental Setup and Mission Scenario*

6.1.1. *The Experimental Setup*

VORTEX (cf. Figure 1a) is a remotely operated vehicle (ROV) designed by Ifremer as a testbed for control laws and control architectures. VORTEX is equipped with a set of screw propellers and traditional sensors, such as a compass, inclinometers, a depth meter, and gyrometers, allowing it to measure its internal state. A video camera is used for target-tracking tasks, and a belt with eight ultrasonic sensors allows us to perform positioning and wall-following tasks. A 7 degree of freedom Mitsubishi PA10 arm is mounted underneath the vehicle for manipulation tasks.

Control algorithms for the arm and the free-floating base are computed on two different external VME backplanes. Synchronization events and state-measurement data are exchanged between the controllers, which are each running ROBOT-TASKS and ROBOT-PROCEDURES related to both subsystems.

6.1.2. *The Mission Scenario*

The following mission scenario involving arm/base coordination of the VORTEX vehicle within its test pool has been chosen (preliminary results were obtained in the UNION project [Rigaud et al. 1998]). At a given depth, with the arm folded in its parking position and locked using the brakes, the vehicle swims ahead to a setpoint in front of a wall. There, VORTEX stabilizes using the acoustic sensors. This ends the first (initialization) phase of the mission.

During the second phase, the vehicle remains stabilized while the arm is in operation. The arm is unlocked and driven to several predefined positions while vehicle stability is monitored. In reaction to a loss of stability, the arm motion is frozen until stability is recovered. On completion of arm motions or when an operator decides, the arm is locked again in parking position and the vehicle returns to its initial position.

The system operator also defines several exceptions and recovery behaviors. At any time, if a water leak is detected, hardware fails, or the depth limit is crossed, then the mission is aborted and an emergency ascent is performed. Also, the vehicle must never swim with an unlocked arm. It is worth noting that these recovery behaviors are mission- and context-dependent (e.g., a loss of vehicle stability is considered to be fatal during the initialization phase, while it leads to synchronizations between arm and vehicle actions during the working phase).

6.2. *The Design Process*

To achieve this mission, various ROBOT-TASKs and ROBOT-PROCEDUREs have to be specified, verified, and implemented. Let us consider the design of one of each entity: the ROBOT-TASK dedicated to the control motion of the arm and the ROBOT-PROCEDURE associated with the second phase of the mission, which consumes the described ROBOT-TASK.

6.2.1. *A* ROBOT-TASK

6.2.1.1. *Control law design*

Controlling the PA10 arm during its motions requires us to specify a control law belonging to the class of decoupling/feedback linearizations in a dedicated task space. The goal assigned to the manipulator is defined as the regulation to zero of an n-dimensional output function $e(q,t)$ (where q is a vector of generalized coordinates), aimed at presenting the user's objective. In the present case, the output function includes the tracking by the arm tip of a trajectory in the six-dimensional space of frames SE(3). Because the robot has seven joints, one degree of freedom is available for simultaneously achieving a secondary task, which can be expressed as the minimization of a scalar cost function $h_s(q)$. Classical secondary goals of that kind are the avoidance of kinematic singularities or of joint limits, the minimization of the velocity norm, etc. The two tasks are finally gathered into a single task through the expression

$$e = J_1^{\dagger} e_1 + \alpha (I_6 - J_1^{\dagger} J_1) \frac{\partial h_s}{\partial q},$$

where e_1 expresses the trajectory tracking task

$$e_1 = \begin{pmatrix} P(q) - P^*(t) \\ A(q,t) \end{pmatrix},$$

where A is a parameterization of the attitude error, P is the position of the tip, and α is a positive weighting factor. ($J_1 = \frac{\partial e_1}{\partial q}$) is the Jacobian matrix of e_1.

The final control law is

$$\Gamma = -k\hat{M}\left(\frac{\hat{\partial} e}{\partial q}\right)^{-1} G\left(\mu De + \frac{\hat{\partial} e}{\partial q}\dot{q} + \frac{\hat{\partial} e}{\partial t}\right) + \hat{N} - \hat{M}\left(\frac{\hat{\partial} e}{\partial q}\right)^{-1} \hat{f}.$$

Here, Γ is the array of control actuator torques, M and N represent the Lagrangian dynamics, k, μ, G, and D are tuning parameters, and $\hat{f}$ comes from the second derivatives of e. The "hats" indicate that more or less complex models of the concerned terms can be used. In fact, it should be emphasized that the ROBOT-TASK designer may easily select the adequate models and tuning parameters in ORCCAD, since they belong to some predefined classes in an object-oriented description of the control.

6.2.1.2. *Specification of the* ROBOT-TASK

The method of Section 3.1 can now be used to create a ROBOT-TASK called PA10TTSE3. Figure 13 provides a graphical representation of this ROBOT-TASK. Most modules are algorithmic, periodic modules instantiated from a library. For example, the KIN module represents the model of the kinematics of the arm. The JL_OBS is an observer that sends an event of Type 2 to the ROBOT-TASK automaton (cf. Section 3.2) in case of reaching a joint limit to trigger a corresponding predefined external exception handling. The PA10 module provides a gateway to the physical system through user-defined drivers. Besides this scheme, which looks like a classical block diagram, numerical and temporal values are added to the modules in view of automatic real-time code generation.

6.2.2. *A* ROBOT-PROCEDURE

The complete description of the operational mission naturally requires the description of four main ROBOT-PROCEDURES:

the first procedure prepares the vehicle for cruising; the second procedure is used to navigate in the pool until the vehicle reaches the inspection place and is stabilized; the third ROBOT-PROCEDURE coordinates the actions of the platform and the arm to simulate the inspection of an underwater structure with the arm tip; and the final procedure drives the vehicle to its homing position.

The whole mission requires 11 ROBOT-TASKS. The PA10TTSE3 ROBOT-TASK described previously forms the basis of the third ROBOT-PROCEDURE. Its specification is illustrated in Figure 5 using the MAESTRO language (cf. Section 3.2). In conformity with the specification of the mission scenario, this ROBOT-TASK is launched in parallel to the ROBOT-TASK dedicated to the basis stabilization. Each time the basis becomes unstable, the brakes of the arm are activated until stability is recovered.

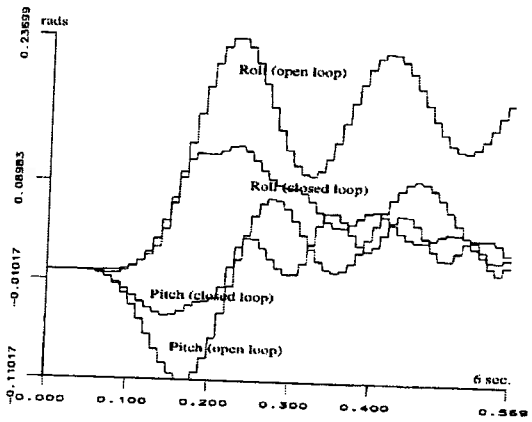

Fig. 16. Roll and pitch errors during arm motion.

6.2.3. *Formal Verification: Coherence with Requirements*

Once the ROBOT-PROCEDURE is specified using MAESTRO, the corresponding ESTEREL code can be automatically generated and the resulting automaton is ready for verification. Figure 14 illustrates the verification of one of the constraints expressed in the mission scenario. Here, we want to certify that the arm is motionless when the platform is recovering stabilization using the ultrasound sensors. The ROBOT-TASKS involved in this property are KEEPSTABLEUS and PA10TTSE3. Therefore, we observe and reduce the global automaton with respect to the signals associated with the start and end of both of these actions, namely, STARTKEEPSTABLEUS, BFKEEPSTABLEUS, STARTPA10TTSE3, BFPA10TTSE3, and ABORTPA10TTSE3.

The resulting automaton shows that no overlapping occurs and, subsequently, that both tasks always run in sequence: From the state OC3, either KEEPSTABLEUS is executed (state OC4) or PA10TTSE3 (state OC5). In particular, if KEEPSTABLEUS runs from state OC5, the PA10TTSE3 task is aborted, and is only reactivated after the end of the KEEPSTABLEUS task.

6.2.4. *Simulation: Use of* SIMPARC

Using the SIMPARC software package, we are able to perform realistic simulations of the VORTEX complex nonlinear dynamic system. Figure 15 shows the model of the VORTEX controller. The coupled dynamics of the robot, including drag, lift, and hydrostatic forces, are computed. According to the limits of the actual hardware, the sampling rates of the vehicle control and arm control are set to 100 ms and 10 ms, while the acoustic sensors are triggered every 360 ms.

Figure 16 shows the pitch and roll orientation errors of the vehicle stabilized in a pool corner using the ultrasonic sensors during a fast motion of the arm. The first and second joints are turned by 90° in 3 s. Owing to the rather large hydrostatic stability of VORTEX, the vehicle does not capsize even if it is not actively controlled. However, closed-loop stabilization with the ultrasound sensors reduces both transient and steady-state errors.

Tuning some parameters of the simulation file rapidly shows that the main limitation in the ROV stabilization performance comes from the low sampling rate of the acoustic sensors. Therefore, should an improvement in the stability of the underwater platform be needed, it would be more effectively obtained with an upgrade of the sensor electronics rather than with an increase of the computing power. The information is of direct benefit to the system designer.

6.2.5. *Experimental Results*

Figure 17 represents experimental results obtained when executing the aforementioned mission scenario (Kapellos et al. 1997). The top plot shows the front acoustic sensor signals, which are used in particular to stabilize the vehicle in front of the pool's wall while the arm is in motion. The bottom plot shows the pitch angle of the vehicle, which is the best way to detect the moments during which the arm is in motion, owing to the low accuracy of time profiling between the two backplanes.

7. Summary and Future Trends

The approach described in this paper has been successfully applied to various kind of robots, such as manipulators, wheeled mobile robots (Pissard-Gibollet et al. 1995), free-floating underwater manipulation systems (Coste-Manière, Peuch, and Perrier 1995; Simon, Kapellos, and Espiau 1996), and the automatic driving of electrical cars (Kapellos et al. 1995). A freeware "frozen" version of ORCCAD, described in Section 5, is now available (http://www.inrialpes.fr/iramr/pub/Orccad/). It is expected that it could also be used in other fields of automation and embedded systems, such as computerized railway systems

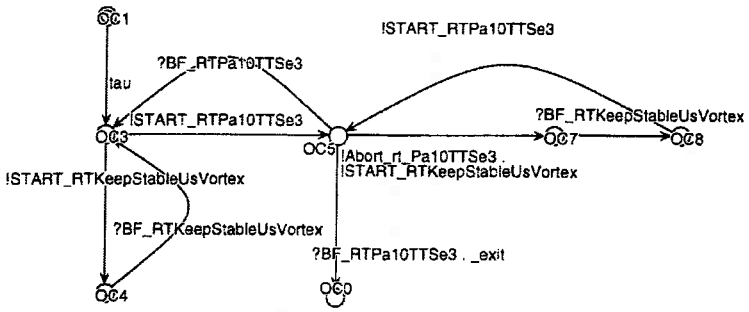

Fig. 14. Conformity with one of the specifications for the mission scenario.

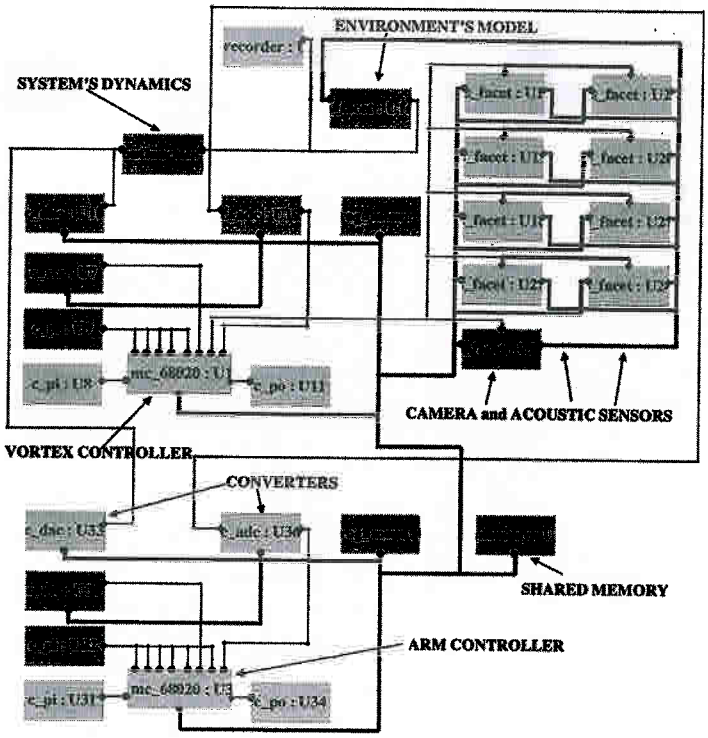

Fig. 15. The SIMPARC model of the underwater system hardware.

and the automotive industry. However, aside from its success, we consider that this work still leaves several questions open.

- Tools improvements—multirate control could provide optimization in the use of computing power. Currently, this is an unresolved problem for embedded systems. A connection between ORCCAD and SYNDEX (Lavarenne et al. 1991) is forecast to provide a gateway toward code optimization on distributed architectures, such as networks of microcontrollers. Concerning the high-level specification of the logical behavior of ROBOT-TASKS and ROBOT-PROCEDURES, which is currently done through a textual approach with ESTEREL and MAESTRO, we also envision to offer the user a StateCharts-like graphical approach.
- Logical behaviors distribution—while most previous applications used a centralized implementation of the logical behaviors in a single automaton, new problems, such as the control of the teleoperated system depicted in Figure 1, raised the problem of distributing the synchronous control code on an asynchronously distributed target. Distributing the ROBOT-PROCEDURE's logical behaviors may improve the system's reliability against communication faults by giving some autonomy to the subsystems. However, setting asynchronous links be-

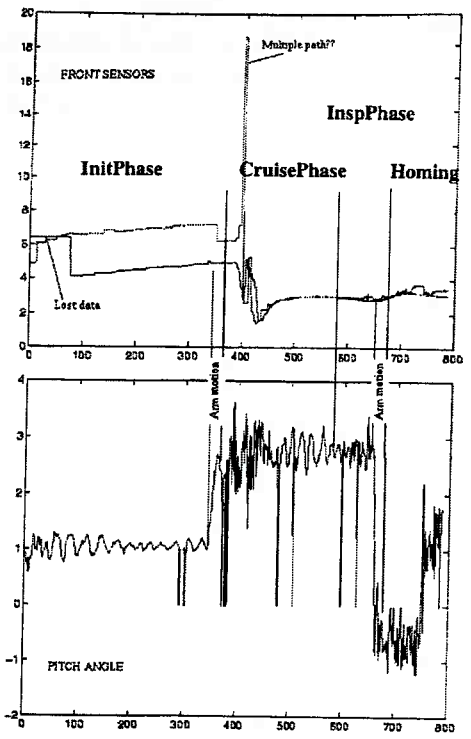

Fig. 17. Some experimental results of executing the mission scenario.

tween the local automata leads to loosing the capability of formal verification for the global system. The new techniques for distributing synchronous specifications on asynchronous distributed targets while preserving global verification capabilities that are emerging from the synchronous programming community (Caspi, Girault, and Pilaud 1994) deserve to be studied and integrated in our tools.

- Handling of symbolic information—in practice, logic is necessary but not sufficient. Clearly, an end user has to specify some issues in the form of symbolic (or linguistic) variables and understand messages from the system with the same semantics. This points out the necessity for a system like ORCCAD to handle symbolic aspects (inputs, outputs, and computations) with the same rigor as it does for events or continuous variables. Because the external views of the ROBOT-TASKS and ROBOT-PROCEDURES are pure discrete-event systems described by automata, it is expected that the application of the supervisory control theory for discrete event systems (Ramadge and Wonham 1989) may provide a solution to automatically generate a safe controller from an end user's constraints.

Acknowledgment

It is our pleasure to thank Alison Noble and Janet Bertot for their careful and patient rereading of our article.

References

Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. 1998. An architecture for autonomy. *Int. J. Robotics Res.* 17(4):315–337.

Albus, J., Lumia, R., and McCain, H. 1988. Hierarchical control of intelligent machines applied to space station telerobots. *Trans. Aerospace and Electronic Sys.* 24(September):535–541.

Astraudo, C., and Borrelly, J.-J. 1992. Simulation of multiprocessor robot controllers. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 573–578.

Baccelli, F., Cohen, G., Olsder, G., and Quadrat, J. 1992. *Synchronization and Linearity*. New York: John Wiley & Sons, Wiley Series in Probability and Mathematical Statistics.

Bellingham, J. G., and Consi, T. R. 1990. State configured layered control. *Proc. 1st Workshop on Mobile Robots for Subsea Environments*, pp. 75–80.

Benveniste, A., and Berry, G. 1991. The synchronous approach to reactive and real-time systems. *Proc. IEEE: Another Look at Real-Time Programming* (special issue) 79:1270–1282.

Berry, G. 1997. A quick guide to ESTEREL. Technical report, Ecole des Mines de Paris and INRIA. Also available from ESTEREL Web page, http://www.inria.fr/meije/esterel/.

Berry, G., and Gonthier, G. 1992. The synchronous ESTEREL programming language: Design, semantics, implementation. *Science of Computer Programming* 19(2):87–152.

Borras, P., Clement, D., Despeyroux, T., Incerpi, J., Kahn, G., Lang, B., and Pascual, V. 1988. Centaur: The system. *Proc. Third Symp. on Software Development Environments (SDE3), ACM SIGSOFT'88*, pp. 14–24.

Bouali, A., Ressouche, A., Roy, V., and de Simone, R. 1996. The FC2TOOLS set. *Computer-Aided Verification*, vol. 1102 of *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag. Available from FC2TOOL Web page, http://www.inria.fr/meije/verification/.

Boudol, G., Roy, V., de Simone, R., and Vergamini, D. 1990. Process calculi, from theory to practice: Verification tools. *Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems*. Berlin, Germany: Springer-Verlag.

Brooks, R. A. 1986. A robust layered control system for mobile robots. *IEEE J. Robotics and Automation* 2(1):14–23.

Byrnes, R. B. 1993. *The rational behavior model: A multi-paradigm, tri-level software architecture for the control of autonomous vehicles*. Ph.D. thesis, Naval Postgraduate School, Monterey, CA, USA.

Caspi, P., Girault, A., and Pilaud, D. 1994 (Las Vegas, NV). Distributing reactive systems. *Proc. 7th Int. Conf. on Parallel and Distributed Computing Systems.*

Coste-Manière, E., Peuch, A., and Perrier, M. 1995. Mission programming: Application to underwater robots. *Proc. Fourth Int. Symp. on Experimental Robotics: ISER'95*, pp. 252–256.

Coste-Manière, E., and Turro, N. 1997. The MAESTRO language and its environment: Specification, validation and control of robotic missions. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'97*, vol. 2, pp. 836–841. Available from MAESTRO Web page, http://www.inria.fr/icare/maestro/.

Espiau, B., Kapellos, K., and Jourdan, M. 1995. Verification in robotics: Why and how? *Robotics Research, the Seventh International Symposium*, Giralt and Hirzinger, eds. Berlin, Germany: Springer-Verlag.

Fernandez, J.-C., Kerbrat, H. G. A., Matescu, R., Mounier, L., and Sighireanu, M. 1996. CADP (caesar/aldebaran development package): A protocol validation and verification toolbox. *Proc. 8th Conf. on Computer-Aided Verification*, vol. 1102, eds. R. Alur and T. A. Henzinger. pp. 437–440.

Harel, D., and Pnueli, A. 1985. *On the Development of Reactive Systems, Logic and Models of Concurrent Systems.* Berlin, Germany: Springer-Verlag, pp. 477–498.

Henzinger, T., Nicollin, X., Sifakis, J., and Yovine, S. 1992. Symbolic model-checking for real-time systems, vol. 92 of Lecture Notes in Computer Science. *IEEE Computer Society Press.*

Ilog. 1995. *Ilog Views Reference Manual* 2.2. Mountain View, CA.

Kahn, G. 1987. *Natural Semantics*, vol. 247 of *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag.

Kapellos, K., Abdou, S., Espiau, B., and Jourdan, M. 1995. Specification, formal verification and implementation of tasks and missions for an autonomous vehicle. *Proc. Fourth Int. Symp. on Experimental Robotics*, pp. 257–262.

Kapellos, K., Simon, D., Granier, S., and Rigaud, V. 1997. Distributed control of a free-floating underwater manipulation system. *Proc. Int. Symp. on Experimental Robotics.*

Lavarenne, C., Seghrouchni, O., Sorel, Y., and Sorine, M. 1991. The Syndex software environment for real-time distributed systems design and implementation. *Proc. European Control Conf.*

MathWorks Inc. 1997. Mathworks Web page, http://www.mathworks.com.

Murata, T. 1989. Petri nets: Properties, analysis, and applications. *Proc. of the IEEE* 77(4):541–580.

Pissard-Gibollet, R., Kapellos, K., Rives, P., and Borrelly, J.-J. 1995. Real-time programming of mobile robot actions using advanced control techniques. *Proc. Fourth Int. Symp. on Experimental Robotics*, pp. 352–357.

Ramadge, P., and Wonham, W. 1989. The control of discrete event systems. *Proc. of the IEEE* 77(1):81–98.

Rigaud, V., Coste-Manière, E., and e.a. 1998. Union: Underwater intelligent operation and navigation. *IEEE Robotics and Automation Magazine* (special issue).

Roy, V., and de Simone, R. 1990. Auto and autograph.*Proc. Workshop on Computer-Aided Verification*, ed. R. Kurshan.

Samson, C., Borgne, M. L., and Espiau, B. 1991. *Robot Control: The Task-Function Approach.* Oxford, UK: Clarendon, Oxford Science Publications.

Schneider, S., Chen, V., Pardo-Castellote, G., and Wang, H. 1998. Controlshell: A software architecture for complex electro-mechanical systems. *Int. J. Robotics Res.* 17(4):360–380.

Simon, D., Castillo, E., and Freedman, P. 1995. Validation of robotics control systems. Part ii: Analysis of real-time closed-loop control tasks. *IEEE Trans. on Control System Technology.*

Simon, D., Espiau, B., Castillo, E., and Kapellos, K. 1993. Computer-aided design of a generic robot controller handling reactivity and real-time control issues. *IEEE Trans. on Control Systems Technology* 1(4):213–229.

Simon, D., Kapellos, K., and Espiau, B. 1998. Control laws, tasks and procedures with ORCCAD: Application to the control of an underwater arm. *Int. J. Systems Science* (special issue) 29(6).

Wind River Systems. 1995. *VxWorks reference manual 5.3.* DOC-11307-ND-00.