A Reactive Approach to Underwater-Vehicle Control: the Mixed ORCCAD / PIRAT Programming of the VORTEX Vehicle

Daniel Simon,Eve Coste-Manière,Roger Pissard
INRIA BP 93
06902 Sophia-Antipolis Cedex – FRANCE

Vincent Rigaud,Michel Perrier,Alexis Peuch
IFREMER BP 330
83507 La Seyne sur Mer Cedex – FRANCE

Abstract

Robotic systems like autonomous underwater vehicles constantly interact in real time with their environment. ORCCAD and PIRAT are robot programming systems aimed to control reactive robots. Both rely on a three-level architecture and are based upon an object-oriented approach in order to enforce software reuse and reliability. Robot actions are modeled using the ROBOT-TASK concept, merging a control law and a logical reactive behaviour. At the mission management level, these elementary actions are scheduled using the synchronous language ESTEREL, allowing to perform formal verifications about the logical behaviour of the control programs. Experiments at the mission level were carried out using the VORTEX testbed vehicle developed at IFREMER.

Introduction

Robot control presents specific issues associated to automatic control but also classic concerns of real-time and high level programming. Robotic systems such as underwater vehicles are made up of several subsystems communicating in order to perform a set of demanding and complex tasks in structured or unstructured environments, within satisfactory levels of real-time performances. They constantly interact with the environment, so they must react in real-time to sensory information.

Several programming and control architectures have been developed to deal with these requirements. Traditional artificial intelligence approaches are encountered in the literature to deal with high level programming. Very often, real time and automatic control issues are not of first concern in those approaches. Therefore, implementations are not efficient and the control of the robotics system cannot be performed in an optimal way. Two other opposed class of approaches, closer to implementation, can be mentioned: a *hierarchical architecture* (NASREM [1]) and a *layered reactive approach* introduced in [9]. The first one is very rigid and promotes the supremacy of high level control, restricting low level communications. The latter, lacks for a top-level supervisor and is organized as a set of communication software modules corresponding to *levels of competence* called *behaviors*[1]. Obviously, distributed control is difficult to carry out for complex applications. A fruitful adaptation of this approach is known as the "State Configured Layered Control" [3].

As a matter of fact, it becomes useless to sophisticate the automatic control algorithms when they cannot be implemented in conditions close to those stated in theory. Therefore, mixed approaches, integrating high level and low level programming concerns in a coherent framework, are necessary to provide robot controllers with an intelligent control system. The Rational Behavior Model [10] is an example of such a hybrid approach. The described work also belongs to this category.

In this paper, we propose a solution to robotics programming where we take advantages of the best techniques available both in automatic control [18] and in real time programming which directly benefit from the emerging *synchronous paradigm* [4]. Two robot controllers ORCCAD and PIRAT are used to design and implement the different control laws encapsulated in ROBOT-TASKS. ROBOT-TASKS constitute the elementary entity handled at the mission programming level where the ESTEREL synchronous language is successfully used.

The paper is organized as follows: we briefly recall the main characteristics of ORCCAD and PIRAT. The keystone concept of ROBOT-TASK adopted in both systems is then described. Our approach is demonstrated on the VORTEX (Versatile and Open subsea Robot for Technical EXperiments) underwater-vehicle developed in the Subsea Robotics Laboratory of IFREMER through the description of a simple mission encoded

[1] "behavior" refers here to the school of psychology

in ESTEREL and effectively carried out in a pool with the PIRAT controller. Trends for the future are drawn in conclusion.

An Overview of ORCCAD and PIRAT

The approach mixes various features of two sets of software tools: ORCCAD developed at INRIA and PIRAT developed at IFREMER. Both systems combine design and simulation tools to a real-time computing environment and are developed using Object Oriented technologies. At run time, both systems use the VXWORKS real-time operating system.

The ORCCAD control system: ORCCAD (Open Robot Controller Computer Aided Design) is a set of CAD tools aimed to design generic robot controllers [19]. Although it has primarily been designed to cope with manufacturing robotic applications, its adequacy to program mobile robots [16] and underwater-vehicles is currently proven. Through a graphical Human-Machine Interface, it provides verification, simulation and code generation tools in order to bridge the gap between control laws as understood by the control systems community, and real-time computing as understood by the computer science community. The associated controller is *open* since qualified users have access to every control level.

The *mission* level is designed for the end-user. A mission is specified by sequencing high level actions and by defining a set of exception handlers around the nominal mission. Using a synchronous language like ESTEREL [5] allows to handle reactivity in a rigorous and theoretically well founded framework. It also allows to perform some formal verifications on the resulting mission program.

Those high level actions called ROBOT-TASKS are conceptually described in the next section. They are designed by a control scientist at the *control* level. Roughly speaking, the ROBOT-TASK is a multitasks program which combines a control law and its reactive behaviour [12]. The ROBOT-TASK designer is provided with a graphical Human-Machine Interface, and the SIMPARC simulation software is used to evaluate the performances of the ROBOT-TASK before real time implementation [2]. SIMPARC is a hybrid simulator able to process in a coherent way the discrete event driven execution of the control code running on a model of the target architecture and the numerical integration of the dynamics of the model of the controlled system.

At the *system* layer, the ROBOT-TASKS are implemented as a set of communicating periodic real-time tasks, the MODULE-TASKS. The MODULE-TASKS are synchronized by communication ports. Formal methods are currently developed to check for temporal properties of real-time closed-loop robotic tasks [20].

The Human-Machine Interface allows to specify the features of the MODULE-TASKS: source files, computation period, synchronizations and connections between ports (figure 1). A ROBOT-TASK is instantiated when all these features are defined. Simulation code can be automatically generated for the SIMPARC simulator. Finally, downloadable code is generated for the VXWORKS operating system.

Let us point out that, in the current state of the system, because of the complex structure of ROBOT-TASKS together with intrinsic stability considerations of the various control laws to be implemented, the mechanisms which allow to switch between ROBOT-TASKS are still to be refined and are not yet implemented. This is a current limitation to mission implementation in the ORCCAD system. In our mixed approach it has been overcome through the use of the PIRAT system at the execution level and the ORCCAD concepts for mission programming.

The PIRAT control system: PIRAT (PIloting a Robot in an Autonomous or Teleoperated way) is a run-time environment more specifically devoted to the control of underwater vehicles. It is currently developed and tested on the experimental VORTEX vehicle at IFREMER[13]. Similarly to the architecture of ORCCAD, the software is made of three levels. The lowest one interfaces the control software and the hardware (sensors and motors) through specific drivers and runs under the VxWorks operating system.

The servo-control level implements control laws as communicating elementary control modules. This software was designed using the Booch methodology [7] and is encoded with the C++ object-oriented language. For example, some classes are related to the vehicle equipment like sensors, actuators and accessories. Other classes are related to algorithms like control laws and filters. At run time, the control modules can be dynamically switched or parametrized using the LIVE graphical interface and the PILI message passing facility. LIVE provides the operator with all the information required to control the vehicle and monitor its state (figure 1). The development tools include the MATLAB/SIMULINK package for control laws design, simulation and automatic C++ code generation. Currently, a reactive behaviour is being added to the PIRAT control laws so that their concepts will be similar to the ROBOT-TASKS of ORCCAD.

At the mission level, PIRAT also uses ESTEREL to

switch between different control laws.
Although some run-time and development tools are different, both the ORCCAD and the PIRAT systems share a common idea: most of the time, a mission for a mobile robot can be achieved by sequencing reactive control laws, the ROBOT-TASKS. Thanks to the adoption of this formal concept, the two systems should be able to live together under control of a joint mission management level.

The Robot-Tasks concept

The ROBOT-TASK is a keystone concept introduced in the ORCCAD framework: it is the minimal granule to be handled by the *end user* at the mission level, while it is the object of maximum complexity to be considered by a *control designer*. It characterizes in a structured way the set of elements allowing the controller to actually work: control scheme in closed loop, temporal features related to implementation, management of associated internal and external events. In a formal way, a ROBOT-TASK is the entire parametrized specification of:

- an elementary servo-control task, i.e. the activation of a control scheme structurally invariant along the task duration;
- a logical behavior associated with a set of signals liable to occur previously to and during the task execution. These signals monitor the execution of the control laws.

An Object-Oriented approach was chosen to model the ROBOT-TASK. A given ROBOT-TASK is then fully specified by the instantiation of the concerned objects. This requires the definition of the elementary servoing task: in a first step, the formal specification in *continuous* time must be established. It characterizes the task from the automatic control point of view. Then, it has to be extended to integrate implementation issues: discretization, variable quantization, delays, computation times, periods... Finally, starting, stopping, killing the task and controlling it require to define adequate signals and to build an automaton managing the overall behavior.
These signals are emitted by some dedicated MODULE-TASKS, the *observers*, and are strongly typed:

- *preconditions* are required to start a servoing task. They can be pure synchronization flags (i.e. to check that all required MODULE-TASKs are ready to run) or measurement preconditions (such as the value of a sensor to be processed).
- *exceptions* are exclusively emitted by observers in case of failure detection during the control law execution.
- *postconditions* are emitted at the end of the servoing task and are used to synchronize with the next ROBOT-TASK and to report the final state of the task.

Exceptions processing belong to three categories:

- *type 1* exceptions can be handled inside the ROBOT-TASK itself, for example by tuning a parameter of the control law and can be reported at the mission management level for information only.
- *type 2* exceptions are emitted when the ROBOT-TASK can no longer be processed: it is reported at the mission level which activates a predefined exception handler, such as switching to another ROBOT-TASK.
- *type 3* exceptions denote a serious failure in the system. The mission can no more be achieved. The system must reach a safety state, for example by dropping the safety ballast of an AUV.

The reactive behaviour of the ROBOT-TASK is implemented by an automaton encoded using the ESTEREL synchronous language, a deterministic concurrent programming language especially designed for writing reactive real time systems [4]. It is based on a model of synchronous parallelism and communication which allows a high level modular programming style simpler and more rigorous than its asynchronous counterpart. ESTEREL has an efficient implementation based on well-defined mathematical semantics. ESTEREL programs are compiled into efficient equivalent sequential automata upon which analysis and proofs can be performed using verification systems dedicated to automata [17].
Since the signals awaited by the ROBOT-TASK automata are strongly typed, the corresponding ESTEREL code is rather straightforward to write. A menu available in the Human-Machine Interface of ORCCAD allows to automatically perform this code generation and therefore to produce the corresponding (and simple) automata.

Mission Programming with ESTEREL

The adoption of a ROBOT-TASK-like concept in the PIRAT framework should allow to smoothly mesh both systems at the mission level. The "global" behaviour of the system will be demonstrated also using the ESTEREL synchronous language as an Application Programming Language. Here, ROBOT-TASKS are the

elementary actions that are handled by the control structure in order to carry out a whole robot mission. The resulting *Application Automaton* deals with the temporal and logical organization of the ROBOT-TASKS. Exceptions and operator's calls handling allows to switch from a nominal mission to predefined recovery procedures.

A major advantage of the synchronous language is the existence in the programming environment of formal verification tools such as AUTO [17] for ESTEREL. Crucial properties for security concerns can be checked forbidding particular situations to occur. Therefore, the local behaviour of the robot tasks and the global behaviour of the general application can be *logically* verified. Using ESTEREL to encode both local and global behaviours allows to establish efficient links between the control and mission levels and to perform rigorous verification of some crucial safety properties. Once these proofs are performed, it is necessary to focus on implementation issues of logically correct programs. Even though actual hardware machines, for which the ideal synchronous model is realistic, do exist, the machines used to support the real-time applications are usually asynchronous. Thus, the underlying synchronous paradigm has to cope with the real asynchronous world, and a particular asynchronous executive architecture must be provided. The inherent difficulties induced by the asynchronous environment are localized in small interface monitors. The remaining (protocols, control automata synchronization and sequencing) can be conceived and proved separately. As stated in [11], to execute ESTEREL programs such as ROBOT-TASKS'automata and application automata, three main parts are necessary. A short description of these parts is recalled here:

1. An interface allows to transform events from various origin in logical events; the continuous asynchronous process controlled by the reactive automaton are event driven, and these events must be managed in a consistent way;
2. The computation tasks carry out all the complex data transformations, but never communicate. They constitute the only components, the execution of which can take "time". The `exec` statement introduced in the ESTEREL language handles these asynchronous tasks. It provides a gateway to asynchrony and therefore remedy the defect of the original language that assumed instantaneous execution of external procedures. `exec` is abundantly used in our robotics context since most of the tasks are endless control laws;
3. The reactive kernel manages the execution of the ESTEREL deterministic automaton produced at compile time. It concentrates all the reactive difficulties and the synchronous hypothesis is here perfectly valid, provided that the execution of a transition remains *atomic* [8, 11].

An example of a pool mission

At this step of the description, we provide a simple example to illustrate the feasibility of our approach. Although both ORCCAD and PIRAT are still under development, preliminary experiments were successfully conducted with the VORTEX vehicle, merging PIRAT tasks at the servo-control level and ESTEREL programs at the mission level. Let us recall the main characteristics of the VORTEX vehicle:

The VORTEX testbed vehicle

VORTEX is an experimental ROV-like vehicle designed to test the autonomous functions to be used by future AUVs [15]. Mechanically, it is made of a tubular structure on which up to eight propellers can be set up at any location (figure 2). Central to this structure resides the on board electronics package and the attitude and depth sensors. Modular foam blocks providing buoyancy are fixed on the top of the structure. The set of sensors includes attitude sensors (two inclinometers and three piezoelectric angular rate sensors coupled with a fluxgate compass), a depthmeter, an altitude sonar, an orientable video camera, a panoramic sonar and an eight devices lateral sonar belt. The camera will be used for target tracking tasks and the sonar belt for navigation tasks like wall following and obstacle avoidance.

Each sensor and actuator is associated with an on board microcontroller used for data collecting and transmitting. The microcontrollers are arranged in a modular network architecture so that it is very easy to add or remove components on the vehicle. At any time, the vehicle control can be switched to the joystick control mode for obvious safety reasons. The PIRAT real-time control software runs on a surface VME system. The vehicle and the surface computer are linked by a tether, used for energy and data transfer. Video and sonar images are transferred using a specific fiber optic link. Others data are sent on a FDDI fiber optic loop. The VME controller, the development workstation and the operator workstation running the LIVE interface are connected on an Ethernet link. The structure of the control system is given figure 3.

The mission

We have chosen to illustrate the execution of a mission specification encoded with ESTEREL in the asynchronous environment provided by PIRAT. This test-mission is carried out in a test-pool and allows to validate the various aspects of our combined approach. Here, only the use of the ESTEREL synchronous language as a mission programming language is illustrated.

At the highest level, the mission is modeled as a set of actions scheduled by an ESTEREL program. At the lowest level, the involved actions are described using PIRAT control tasks: teleoperated modes with the pilot in the loop and autonomous modes with many kinds of classic control laws. Some sensor-based control laws (for e.g. wall-following tasks) were specifically designed to achieve the mission. The mission automaton currently runs on a workstation and communicates bidirectionally with PIRAT through the use of PILI messages. Observation functions are implemented on the workstation to interpret information from the VORTEX. The results of these observations are inputs to the automaton while outputs from the automaton are translated into the corresponding PILI orders towards the VORTEX.

The nominal mission scenario is the following: VORTEX is driven in a teleoperated mode to any point in the test pool. Upon reception of an event provided by the supervision interface (a switch activation), it is set to its autonomous behaviour. Then the scenario runs automatically. First, VORTEX dives at a desired depth (a simple PD controller was used, but non linear PID [14] or robust controllers like $H-\infty$ or Sliding mode controllers can be used). When the depth is reached, the vehicle swims ahead while maintaining its depth in order to approach the wall in front of it.

When a front corner sounder detects an obstacle (any of the pool walls) at a predefined set point, the control is switched to the wall following mode. The flux-gate measurements are used to watch up the laps count. The mission is stopped when the desired laps number is reaches. Then, the vehicle ascent to the surface.

Two exceptions were defined: at any time, the operator can stop the mission by hitting a key on the LIVE interface: VORTEX will then surface and wait to be reset to teleoperated mode.

At any time during the mission, a water leak detection stops the mission and drives VORTEX to the surface.

Thus, four PIRAT control laws are used to achieve the mission: **Diving**, **GoAhead**, **WallFollowing** and **Surface**. Actually, we are currently in the process of implemented these elementary actions into ROBOTTASKS in the ORCCAD system. So far, only **Diving** and **Surface** have been successfully carried out on VORTEX. We hope to adapt other elementary actions by taking advantages of the skill acquired in mobile robots domain [16]. We will develop for example, wall following strategies using visual servoing technics. Signals issued from the various sensors are used for both control law computation and for mission management. As VORTEX does not have absolute position and velocity measurements, the wall following control law is a purely sensor-based control loop designed according to the approach described in a companion paper [21].

The ESTEREL code

The simple ESTEREL code (where declarations were discarded) is given down below:

```
module VORTEX:
await START;
trap STOP in
   do
      do
 exec DIVING()();
 emit DEPTH_REACHED;
 trap GOAHEAD_PHASE, GOAHEAD_EXCEPT in
 [
    exec GOAHEAD()(); exit GOAHEAD_PHASE
 ||
    await  WALL_SONAR; exit GOAHEAD_EXCEPT
 ]
 handle GOAHEAD_PHASE do
    emit GOAHEAD_OVER
handle GOAHEAD_EXCEPT do
    emit GOAHEAD_INTERRUPTED;
    exec WALL_FOLLOWING()();
         end;

 watching OPERATOR_INTERRUPT
timeout
     emit SURFACE_REQUESTED;
              exit STOP
 end;
   watching WATER_LEAKAGE
   timeout
      emit LEAKAGE;
      exit STOP
   end;
handle STOP do
emit MISSION_ABORTED;
exec SURFACE()();
end % trap STOP
```

Let us comment this typical piece of code. *Signals* are basic entities in an ESTEREL program. They are used for communication and synchronization. They can be emitted or awaited using the **emit** and **await**

statements. Signals are broadcasted inside their declaration scope.

As mentioned before, the `exec()()` statement was designed to manage asynchronous external tasks such as those of the PIRAT software. For each `exec` statement, the compiler provides three interface functions: the `start()` and `kill()` functions are used to activate or suspend asynchronous tasks. The `return()` function is received by the automaton and signals the completion of the external task. The user defines the body of these functions with calls to the PILI messages. However, in the next attempts, we will directly manipulate operating system calls in order to manage all the real-time tasks to be activated within the current control task.

The `trap...exit...handle` statement is an exit mechanism fully compatible with parallelism. It is extensively used in the ROBOT-TASKS structure to manage exception handling. Another preemptive statement of ESTEREL is the watchdog accessed through the use of `do...watching` statement. Since physical time has no particular meaning in ESTEREL, watchdogs can be triggered by *any* kind of signals, for example by a setpoint pool laps count. The preemption mechanisms provided by ESTEREL are here very useful to encode the fault recovery behaviors embedding the nominal mission.

Another important statement is the parallel construct (||) which renders parallelism explicit to the user. It is abundantly used to encode the ROBOT-TASKS behaviour since various observers signals must be expected in parallel. It can also be useful at the application level, for example to split the main program in various parts, some related to generic actions like safety functions and recovery behaviours, others associated to the nominal part of the mission execution.

At compile time, the semantics at the basis of ESTEREL implementation automatically translate parallel programs into the corresponding finite state automaton. The mission automaton corresponding to the above mentioned code can be visualized by AUTOGRAPH [17] (figure 4). For the understanding of this automaton, the transitions received signals are labeled with (?) and emitted signals correspond to (!). The simulation package provided with ESTEREL has also been used to debug and refine the mission.

A major advantage of synchronous languages is that they are deterministic. Therefore, formal verifications of programs can be performed. Behaviors which are crucial from the reliability point of view can be checked off line by interfacing the produced automata with tools dedicated to automata analysis. For example, looking at this automaton, one can easily check that in every state but the first one, the WATER_LEAKAGE signal is awaited and that its occurrence always drives the control system to the recovery configuration (surfacing).

Let us point out that even though the compiler translates the source file into an efficient automaton, the mission is not explicitly specified in terms of a finite state machine. Through the use of ESTEREL, the mission programmer can specify it's program in term of actions sequencing, actions parallelization and exceptions recovery.

Once proofs and simulation were carried out, the mission automaton was implemented following the requirements stated in the previous section. Actually, the execution scheme is symmetrically organized around the reactive entity which ensures the correct activation of the automaton. The results gained from this study now lead to a formalization of the features required to automatically generate execution machines of ESTEREL programs. Ideally, this generator of execution machine will be integrated in the programming architecture of our system.

Summary and Future Trends

Both ORCCAD and PIRAT provide clean frameworks to cope with robot programming, from mission specification to real-time tasks implementation. The controllers are split into several levels, leading to open architectures where CAD tools are provided to qualified users. In particular, the end-user of the system is provided with high level actions, the ROBOT-TASKS. This allows him to concentrate on application specification and verification rather than on low level programming tricks.

A new version of PIRAT called PIRRAT (PIloting a Reactive Robot in an Autonomous or Teleoperated way), is currently under development. It will provide a clean design and implementation of a generic low level controller for autonomous and teleoperated subsea vehicles. In particular, PIRRAT will also be used for the real-time control software of the ROV6000, a scientist ROV currently under development at IFREMER. The concept of ROBOT-ACTION (a ROBOT-TASK-like concept) will be defined and implemented. Also, a protocol for the handling of ROBOT-ACTIONS is being specified. This will directly benefit the merging of ORCCAD and PIRRAT at the mission level.

Powerful simulation tools were proven to be efficient for control law design and tuning. However, development remains to be done to fully integrate models of sensors and environment in the packages we use.

Automatic code generation from design software tools leads to fast and less error prone programming, with the help of object oriented technologies to enforce software reusing and reliability.
The experiments showed that the execution machine for ESTEREL, i.e. the asynchronous/synchronous interface, must be carefully designed to be sure that the Application Automaton always keeps a coherent view of the real world. The new Communicating Reactive Processes development of ESTEREL should be a safe solution [6].
Finally, another ongoing work concerns the development of high level application-oriented tools over ESTEREL allowing an easier mastering of synchronous programming and the establishment of links with automatic planners.

References

[1] J. Albus and R. Quintero. Toward a reference model architecture for real-time intelligent control systems (artics). *ASME Press Series*, 3, 1990.

[2] C. Astraudo and J.-J. Borrelly. Simulation of multiprocessor robot controllers. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 573–578, Nice, May 1992.

[3] J. G. Bellingham and T. R. Consi. State configured layered control. In *Proceedings 1st Worshop on Mobile Robots for Subsea Environments*, pages 75–80, Monterey, October 1990.

[4] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proc. of the IEEE, Special Issue*, 79(9):1270–1282, September 1991.

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

[6] G. Berry, S. Ramesh, and R. Shyamasundar. Communicating reactive processes. In *Annual Symposium on Principles of Programming Languages (POPL 93)*, Charlestown, 1993.

[7] G. Booch. *Object-Oriented Design with application.* Benjamin/Cummings, Menlo Park, Ca, 1991.

[8] F. Boussinot and R. de Simone. The ESTEREL language. *Proceedings of the IEEE, Special Issue*, 79(9), September 1991.

[9] R. A. Brooks. A robust layered control system for mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[10] R. B. Byrnes. *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles.* PhD thesis, Naval Postgraduate School, Monterey, USA, March 1993.

[11] E. Coste-Manière. A synchronous/asynchronous approach to robot programming. In *Proceedings of the fifth Euromicro Workshop on Real-Time Systems*, pages 268–273, Oulu, Finland, 22-24 June 1993.

[12] E. Coste-Manière, B. Espiau, and D. Simon. Reactive objects in a task-level open controller. In *IEEE International Conference on Robotics and Automation*, pages 2732–2737, Nice, May 1992.

[13] M. Perrier, V. Rigaud, E. Coste-Manière, D. Simon, and A. Peuch. Vortex: a versatile testbed vehicle for control algorithms evaluation. In 8^th *International Symposium on Unmanned Untethered Submersible Technology (UUST'93)*, New Hampsire USA, 29-36 1993.

[14] M. Perrier, V. Rigaud, C. C. de Witt, and R. Bachmayer. Performance oriented robust nonlinear control for subsea robots: Experimental validation. In *IEEE International Conference on Robotics and Automation*, page to appear, San Diego, May 1994.

[15] V. Rigaud, E. Coste-Manière, M. Perrier, A. Peuch, and D. Simon. VORTEX: Versatile and open subsea robot for technical experiment. In *Proceedings of OCEANS'93*, volume 1, pages 198–203, Victoria, Canada, October 18-21 1993.

[16] P. Rives, R. Pissard-Gibollet, and K. Kapellos. Development of a reactive mobile robot using real time vision. In *Third International Symposium on Experimental Robotics*, pages 91–109, Kyoto, Japan, October 1993.

[17] V. Roy and R. de Simone. Auto and autograph. In R. Kurshan, editor, *Workshop on Computer Aided Verification*, New Brunswick, June 1990.

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

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

[20] D. Simon, P. Freedman, and E. Castillo. Analysing the temporal behavior of real-time closed-loop robotic tasks. In *IEEE International Conference on Robotics and Automation*, page To appear, San Diego, May 1994.

[21] D. Simon, A. Santos, and V. Rigaud. A new sensor-based control approach for autonomous underwater vehicles. In 2^{nd} *Workshop on Mobile Robots for Subsea Environment, International Advanced Robotics Programme*, page To appear, Monterey, May 1994.

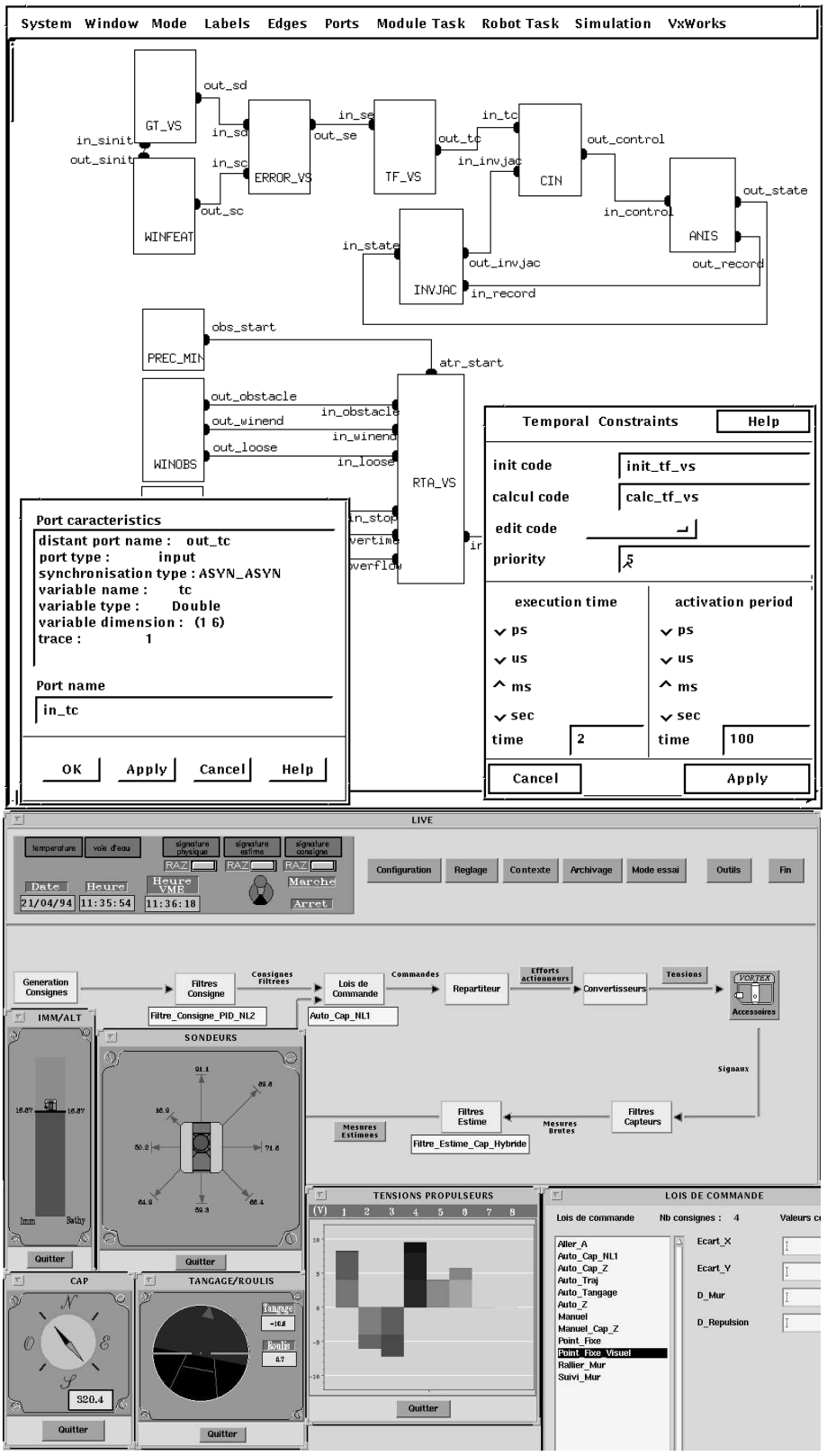

Figure 1: *Top:* The ORCCAD interface for ROBOT-TASK design – *Bottom:* The LIVE interface for VORTEX

Figure 2: The VORTEX testbed vehicle

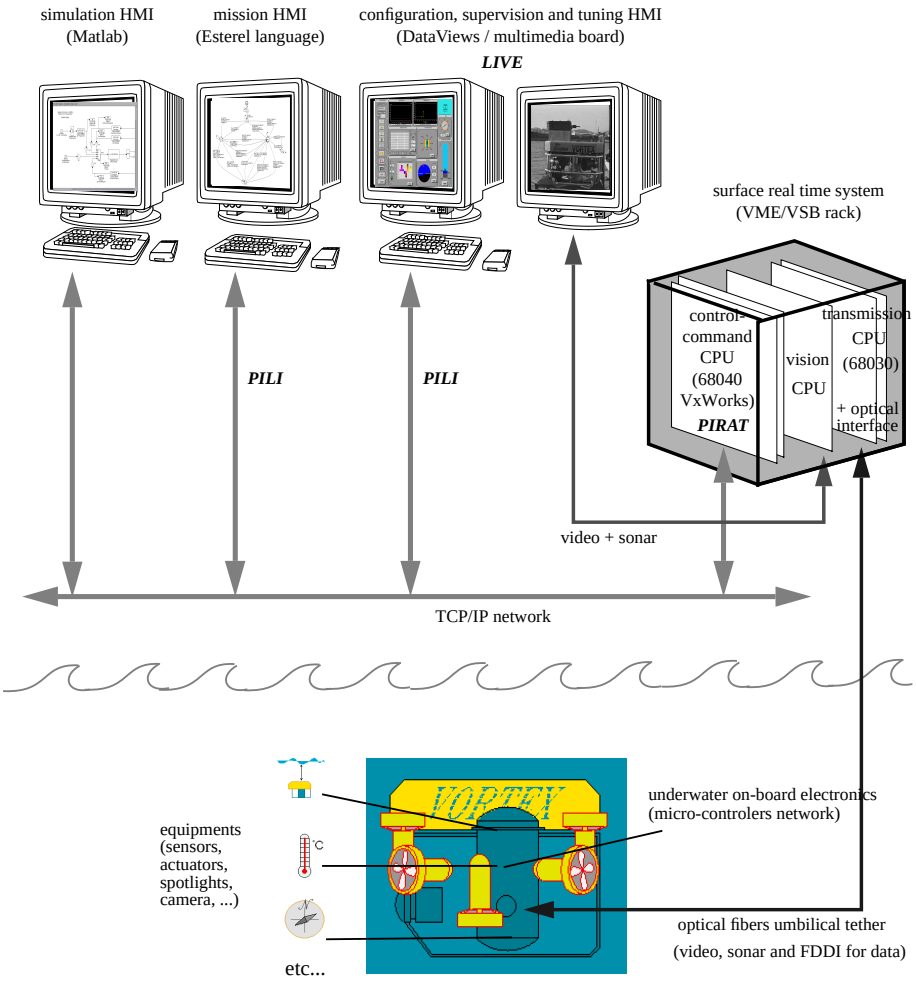

Figure 3: Structure of the current VORTEX control system

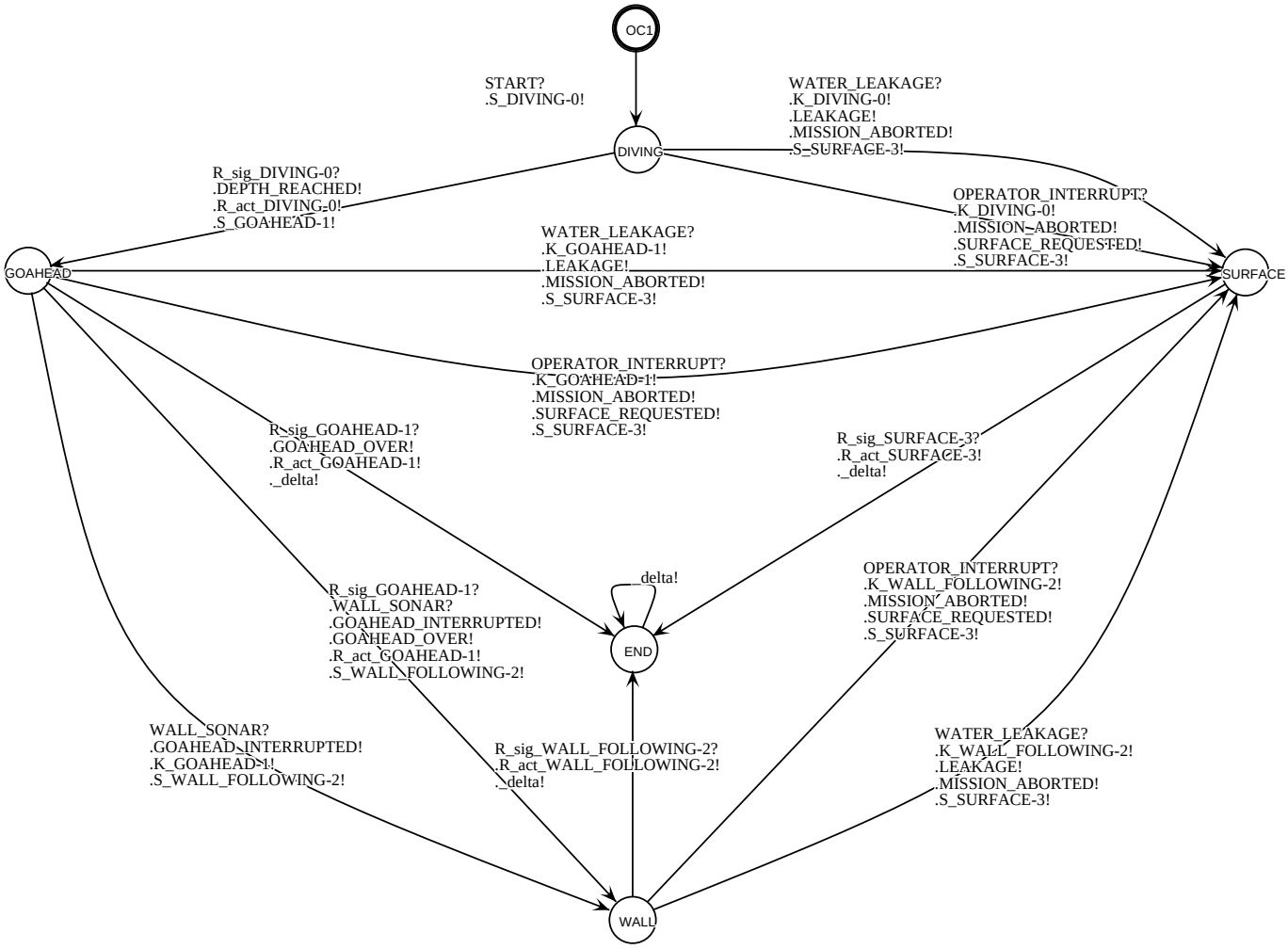

Figure 4: The mission management automaton