Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues

Daniel SIMON †, Bernard ESPIAU †, Eduardo CASTILLO †
Konstantinos KAPELLOS ‡

August 3, 1995

(†) INRIA
2004 Route des Lucioles
BP 109 06902 Sophia-Antipolis France

(‡) ISIA/ENSMP
rue Claude Daunesse
06565 Valbonne Cedex - France

Abstract

This paper describes an original system, ORCCAD, for the computer-aided design of robot controllers. Accessed by three different user levels (system, control and application), it proposes a coherent approach from a high level specification down to its implementation, and offers several tools for design, display and test.

Following a critical study of the main architectures reported through the literature, the paper presents the basic principles and underlying concepts of ORCCAD. The main entity considered is the *robot task*, an elementary control action associated with a local behavior controlled by a set of observers and modeled by a finite state automaton. It is made of a set of real-time communicating tasks, called *module tasks*, the full definition of which requires the specification of temporal and synchronization features. The module task which handles the behavior of the robot task is described using the synchronous language ESTEREL. The application level is defined as a set of synchronized robot tasks, also described using ESTEREL.

Two detailed examples are discussed. The first one consists in the specification and the test of a few versions of a joint control for a rigid robot; the other one is a mission specification for an autonomous underwater vehicle.

Dialogue and verification tools are integrated within ORCCAD: a graphical Human-Machine Interface used to build the Robot-Tasks and a simulation software dedicated to hybrid continuous/discrete time simulations. [1]

[0]Part of this work is made in cooperation with Aleph Technologies, Grenoble, France with support from Ministère de la Recherche et de l'Espace under grant #91 S 0097. A long version of this paper is available as Inria research report #1801 ([38])

1 Introduction

For the last ten years, many kinds of software architectures for the intelligent control of robotic systems have been designed. Different techniques emerging from cognitive science as well as automatic control domains were used. Although a complete robot control architecture includes various subsystems, developments generally focused only on few of them. Some approaches concentrate on system design, i.e. functional decomposition, while others mainly investigate implementation issues. Ad-hoc tools are then to be used in order to complete the system design, while other important aspects like verification of specifications and mapping of functional tasks on a hardware architecture fall in the gap ([26]). Let us examine the main approaches to the robot controller design problem.

Some existing implementation-oriented approaches rely on dedicated real-time operating systems like Chimera [40], using for example fast interprocess communication tools ([32]). Powerful distributed machines have been built ([28]), or layered software libraries have been designed ([5], [11]). However, although these approaches are efficient at the lowest level, they do not consider other levels in a coherent way and they generally provide the end-user neither with friendly interfaces nor with any programming methodology.

Hierarchical architectures like NASREM ([1]) have been proposed in order to enforce modularity and software development methodology. Within this approach a measurement data path flows from sensors to the highest levels and a control data path is fed back to actuators. Such a structure is very rigid as all "intelligent" actions like planning using sensors can only be defined at the high level while communications between low or intermediary levels are limited. This is a major drawback to build, for example, real-time efficient sensor-based control.

High level programming of robots traditionally falls in the field of Knowledge-Based Systems and mainly deals with planning and reasoning. Usually purely cognitive approaches resolutely ignore the basics of mechanics, control and real time computing. This results in techniques which are not time-efficient and have difficulties in taking into account some of the uncertainties involved in the real world where robots operate. The application of behaviorism to robots was proposed by Brooks as a counterpart to such classical A.I. approaches ([12]). Behaviorism was first introduced as a model of animal and human psychology ([41]). Within this theory, the activity of living beings may be modeled by a set of behaviors. Each elementary behavior is characterized by a reaction to a set of stimuli coming from the environment. The global behavior is considered as a simple juxtaposition of elementary ones using low level inhibition mechanisms without coordination from an upper mind level. This very mechanistic view of mental processes was not confirmed by experimentation and was fought by both psychologists and philosophers ([31]).

When applied to robots, behaviorism led to layered control based on the so-called *subsumption* architecture. The layered control system is organized as communicating software modules corresponding to *levels of competence*. New behaviors are added whenever more complex behavior is required. Conflicts due to competing reactions are solved by inhibiting the outputs of the lowest priority modules. A top-level supervisor therefore does not exist: control of the overall behavior is distributed along a set of modules and becomes difficult to analyze and validate, as soon as the set of elementary behaviors increases. Although small insect-like robots have been built according to this approach, unexpected responses may be generated, as mentioned in ([6]). Another drawback comes from the subsumption architecture: running permanently *all* the behaviors even when not necessary, and simply adding new processors when new behaviors are required, is clearly not cost-effective.

Nevertheless this reactive approach made a fruitful breach in the classical A.I. point of view and received further modifications. Let us quote for example the "State Configured Layered Control" ([6]) where subsets of behaviors are grouped into states under control of a state transition diagram. Using such a supervision level allows a more comprehensive control of applications and improves real-time efficiency.

Finally, hybrid architectures which gather the best features of the previously mentioned approaches are now emerging. The Rational Behavior Model ([13]) is an example of them: a mission is specified using a rule-based strategic level and concurrent repetitive behaviors are implemented in a tactical level. Their outputs are used by servo-loops at the execution level. Other interesting examples of hybrid approaches can be found in [9] and [19].

The work described in this paper falls in this last category and is based upon the following main considerations:

• most actions to be performed by robots can be stated as control problems which can be efficiently solved

in real-time by using adequate feedback control loops. We believe that control theory should be used as far as possible to specify complex actions. In this framework, the *Task-Function* approach, ([36]) specifically developed for robotic systems which may involve sensor-based control tasks, is the one used here;
• a robotic system should ideally be accessible to users with different competences. In particular the end-user of the system should be provided with high level functions, allowing him to concentrate on application specification and verification rather than on low level programming tricks. Besides, the control scientist must be provided with efficient design and programming tools. In the approach we propose, every type of user has his specific access to the system.
• since the overall performance of the system finally relies on the existence of efficient real-time mechanisms at the execution level, particular attention is paid to their specification and their verification.
• robotic control often requires the use of complex algorithms, the programming and test of which take a long time. This is why Object-Oriented Design and Programming are used here in order to improve software reliability and reusability ([15]). Automatic code generation is also used as much as possible.

This paper is organized as follows: the next section gives an overview of the *Open Robot Controller Computer Aided Design* system (ORCCAD). In section 3, the Robot-Task concept is analyzed in depth. Section 4 gives two examples of applications of ORCCAD, and section 5 presents a few issues about the ORCCAD software environment. Further developments of these issues are sketched in the conclusion and can be found in [22] and [38].

2 The ORCCAD Concept

2.1 General View

A robotic process may be defined as a set of robot actions organized such as to carry out a given application like an assembly operation or the mission control of a mobile robot. The design of a robotic process requires expertise in several domains: knowledge in mechanics is often required to properly define the task to perform, automatic control theory is involved in the design of control laws and tools from computer science are needed to produce efficient runtime software. Several key points in the design and the implementation of an application can thus be identified. In a first step, it is necessary to define all the elementary tasks involved. These are for example the atomic entities handled by a planning system. For each of them, issues from automatic control and implementation aspects have to be considered: definition of a regulation problem which may be significantly related to the elementary task, choice of a suitable control law, selection of the events liable to be considered during the task execution and definition of the associated reactions, decomposition of the task into synchronized real-time subtasks. Finally, all the defined real-time subtasks should be mapped on a target architecture ([37]).

Many of degrees of freedom are therefore given to the control designer in order to match an end-user specification. The aim of the ORCCAD system is to help the user to exploit these degrees of freedom in the more efficient way. The associated controller itself is naturally *open* since qualified users have access to every control level: the *application* layer is accessed to by the *end-user* of the system, the *control* layer is programmed by an *automatic control* expert and the lowest one, the *system* layer, is the charge of a system engineer.

Let us now take from [10] and [16] a few essential definitions for understanding the following:
• a *Robotic System* is a set of cooperating devices (such as robots and sensors) associated with a control system.
• a *Robot Controller* is the set of hardware and software resources involved in the *on-line* control of a robotic system.
• an *Application* is a set of actions performed by the system in order to reach a goal specified by an end-user.
• a *Robot-Task* ("RT") is a multitask program representing robotic actions. It gathers algorithmical and logical aspects and constitutes the elementary task previously evoked.
• a *Module-Task* ("MT") is a real-time task. Therefore, a RT is made of a connected set of MTs.

The ORCCAD system is a set of CAD tools allowing to design, test and implement applications, Robot-Tasks and Module-Tasks as defined above. The description of Robot-Tasks, down to the specification of its temporal parameters, is made through a specific Human-Machine Interface, while verification and simulation tools are available at the implementation level.

Because of the particular importance of the Robot-Task, we now present its *design methodology*, as it appears in the ORCCAD philosophy. The RT structure itself will be further detailed in section 3.

2.2 The ORCCAD Methodology for the Design of Robot-Tasks

Figure 1 illustrates the successive steps involved in the creation of a RT. Let us focus on the main items.

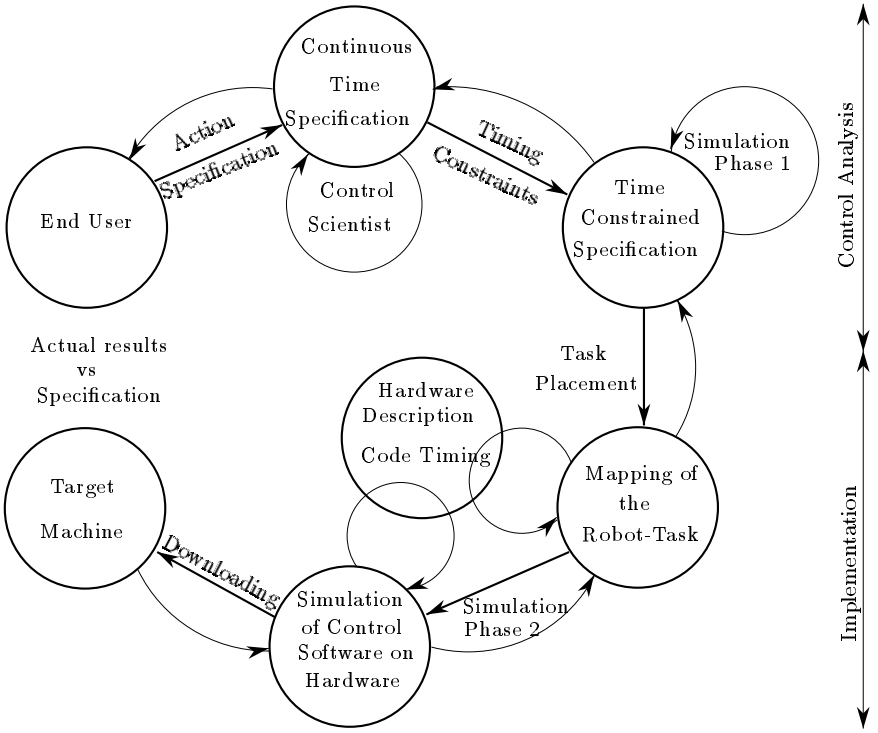

Figure 1: Robot-task creation process

2.2.1 Step 1: Specification of a Robot-Task

A new RT specification is provided by the end-user whenever a suitable RT cannot be found in the ORCCAD library. This description is done in a natural-like language. It includes a physical description of the robotics system to be used and a description of the task specifying in particular:

- the action to be performed during its execution, including the associated performances indices to be monitored (for example, the maximum allowed tracking error);
- the events to be taken into account during the task execution and the actions to be undertaken upon reception of the generated exceptions;
- the conditions authorizing the task to start and the ones generated at its completion.

From this description, the automatic control expert can then propose symbolic descriptions in continuous time of the control laws liable to carry out the specified action. This formal specification of control laws is given in terms of block-diagrams and/or equations. As said before, a systematic way to design control laws from the task specification is the "Task-Function approach". Owing to this approach, and at least for rigid robot arms, a specific control law can be derived from a generic control scheme. Inside this general structure the RT designer can select adequate items among a large choice of models and algorithms.

2.2.2 Step 2: Time Constrained Specification

The control algorithm may now be split into smaller tasks, the so-called Module-Tasks. The MTs communicate via message passing, using synchronization mechanisms and ports described in section 3.2.1.

A MT owns temporal properties like its computing duration and its activation period. These properties are attributes added at this step to the continuous time specification of the control law.

The set of MTs, of their temporal properties and of the chosen synchronization scheme defines the Time Constrained Specification (TCS) of the Robot-Task. Many properties would now be checked at the TCS level before implementation. From the control aspect the available analysis tools unfortunately do not give quantitative results on such non-linear systems driven by sampled, distributed and multi-clocked control loops. Besides, liveness properties must also be checked on the synchronization scheme to avoid, at least, dead-locks. Although using Timed Petri Nets and Synchronous Languages description are currently investigated for this purpose, formal tools do not exist yet to check the correctness of the temporal scheme of RTs. So far, the only solution is by simulation.

The simulation phase associated with this step is the first one proposed in ORCCAD. It takes into account not only the algorithmical and temporal attributes of the TCS, but also the physical behavior of the controlled system, modeled as a set of differential equations. Thus it may help the RT designer to tune performance-related parameters, such as the values of gains and the sampling periods of control loops.

2.2.3 Step 3: Implementation of the Robot-Task

Once the first simulation phase allowed the designer to conclude that an implementation was feasible, the MTs have to be mapped on a target architecture which is often distributed.

It is well known that optimal task scheduling is a NP-hard problem. Although many algorithms have been developed, only few of them take into account real time constraints like the respect of deadlines and sampling periods. In that framework, the facilities offered by ORCCAD simply consist in allowing the easy specification of a distribution and the validation by a dedicated simulation step. New attributes related to the operating system calls used to instantiate and activate real time tasks are thus added to the MTs in this step of task placement.

3 The Robot-Task in Depth

The Robot-Task is a keystone concept in the ORCCAD framework: it is the minimal granule to be handled by the *end user* at the application level, while it is the object of maximum complexity to be considered by a *control designer*. It characterizes in a structured way the control scheme in closed loop, the temporal features related to implementation and the management of associated events. It is defined in a formal way as follows:

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.*

An Object-Oriented approach was chosen for modeling the Robot-Task. A given Robot-Task is then fully specified by the instantiation of the concerned objects. As seen in the previous section, this requires the definition of the elementary servoing task: in a first step, the formal specification in *continuous* time has to be established. That characterizes the task from the automatic control point of view. Then, it has to be extended to take into account 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. Let us now describe the different aspects successively.

3.1 Continuous Time Control Specification

3.1.1 Introduction

In its present implementation, the system is dedicated to the design of control schemes and to the definition of tasks for a particular class of mechanical systems: the rigid robot manipulators. Nevertheless, a large variety of mechanical structures can be modeled owing to the connection of ORCCAD with ACT, a robotics-oriented CAD system ([29]). Moreover, considered tasks include the use of exteroceptive sensors: force, range, proximity, vision, which is one of the ORCCAD originalities. Let us now examine, briefly since it is not the aim of this paper, the proposed control scheme. Its full theoretical study and the discussion of various examples may be found in [20] and [36].

The basic principle consists in separating, in the *design* step, the specification of the task to be performed from the determination of the low-level control law, while properly melting both aspects in the *achievement* of the global control scheme. The first aspect deals with expressing the user's objective under the form of an adequate output function $e(q,t)$, called *task function*. The second consists in choosing the set of models to be implemented and in tuning the various parameters. That way, changing a robot for a given task, specifying various tasks for a given robot or refining a control law for a given couple {task, robot} are handled within a single framework.

3.1.2 A General Control Scheme

The dynamic (state) equation of a $n-$jointed robot, with joint coordinates denoted as q is:

$$\Gamma = M(q)\ddot{q} + N(q,\dot{q},t) \tag{1}$$

where Γ is the $n-$vector of joint actuator torques, M is the kinetic energy matrix, and N gathers all other dynamical terms. Let us consider a n-dimensional C^2 task-function, $e(q,t)$, to be regulated to zero during the time interval $[0,T]$, starting from an initial position q_0. The simplest example of such a function corresponds to trajectory tracking in q-space: $e(q,t) = q - q_d(t)$. Many more interesting choices, sometimes complex, are offered to the user in the ORCCAD system (see Appendix).

If there exists a C^2 solution to the equation $e(q,t) = 0$ and if some regularity conditions of the *task-jacobian* $\frac{\partial e}{\partial q}(q,t)$ are satisfied, the control Γ can be found. By writing:

$$\dot{e} = \frac{\partial e}{\partial q}(q,t)\dot{q} + \frac{\partial e}{\partial t}(q,t), \quad \ddot{e} = \frac{\partial e}{\partial q}(q,t)\ddot{q} + f(q,\dot{q},t), \tag{2}$$

it may be seen that a control which ideally decouples and ensures an asymptotically stable behavior of e is ([36]):

$$\Gamma = M\left(\frac{\partial e}{\partial q}\right)^{-1}[-kG(\mu De + \dot{e})] + N - M\left(\frac{\partial e}{\partial q}\right)^{-1} f \tag{3}$$

where G and D are positive matrices and k and μ positive scalars.

The ideal control scheme (3) is based on a perfect knowledge of all its components. In ORCCAD, this control is implemented under the more general form:

$$\Gamma = -k\hat{M}\left(\widehat{\frac{\partial e}{\partial q}}\right)^{-1} G\left(\mu De + \widehat{\frac{\partial e}{\partial q}}\dot{q} + \widehat{\frac{\partial e}{\partial t}}\right) + \hat{N} - \hat{M}\left(\widehat{\frac{\partial e}{\partial q}}\right)^{-1}\hat{f} \tag{4}$$

where, finally:
- $k(.)$ is a positive scalar gain, possibly variable. It has the intuitive meaning of a velocity gain, while the positive scalar μ may be interpreted as the ratio between velocity and position gains.
- G and D are constant positive adjusting matrices, generally diagonal.
- $\hat{M}(q,t)$ is a $n \times n$ symmetric positive matrix chosen as a model of the system kinetics energy matrix.
- $\hat{N}(q,\dot{q},t)$ is a $n-$dimensional vector function chosen as a model of the sum of the terms of gravity, friction, Coriolis and centrifugal forces.
- $\widehat{\frac{\partial e}{\partial q}}(q,t)$ is a $n-$dimensional vector function chosen as a model of the task-function jacobian matrix, while $\widehat{\frac{\partial e}{\partial q}}^{-1}(q,t)$ is the model chosen for its inverse.
- $\widehat{\frac{\partial e}{\partial t}}$ is a $n-$dimensional vector function chosen as a model of the time partial derivative of the task function.
- $\hat{f}(q,\dot{q},t)$ is a $n-$dimensional vector function chosen as a model of the terms coming from the second-order differentiation of e (cf eq. (2)).

In general, constraints on actuators and joint limits also exist, and an integral term can be added to equation (4).

Now, the elementary servoing task is fully specified in continuous-time when all functions, models and parameters involved in equation (4) are defined.

3.2 Implementation Issues

The passage from the above continuous-time specification to a description taking into account implementation aspects is done in a strongly structured way in the ORCCAD approach. In fact, we have at this level to handle *temporal properties*, i.e. discretization of the time, durations of computations, communication and synchronization between the involved processes. This is done by defining the basic entity called *Module-Task*, which is a real-time task used to implement an elementary process of the control law.

The MTs will be distributed over a multiprocessor target architecture: in order to improve programming modularity the MTs will therefore communicate using message passing and typed ports. Moreover, in order to ease the automatic code generation from the graphical HMI, the structure of the MTs is given by figure 2 (for a periodic task):

```
Initialization code;
   while(1){
      Reading all input ports;
      Data processing code;
      Writing all output ports;}
```

Figure 2: Structure of a Module-Task

Such a structure clearly separates calculations, related to control algorithms issues, and communications, related to implementation aspects and calls to the operating system.

When building the communication graph, the I/O *ports* are the only visible parts of the MTs. Each MT owns one input port for each required data and one output port for each produced data. The only operations that a MT may perform on a port are "Send" or "Receive". Implementation-related properties, like the name of the connected task and the type of synchronization to be used on the pair of ports, are given by the designer of the RT. This allows the reuse of the MTs as objects in different RT schemes with no need of modification of their internal data structure.

3.2.1 Message passing and synchronization

In general-purpose computers and networks systems, communication protocols emphasize data integrity, using large buffers and recovery procedures to avoid loss of data. On the other hand, in a closed-loop control system where most of the tasks are periodic, the exchanged data describe the state of the system: therefore the best data to be read for control purpose is the last produced. Moreover, it is often more efficient to occasionally lose data and to wait a while for the next one, or to reuse the last available data, than to start a long recovery process.

Often, in a rather complex structure like a RT, we may found loops running at different sampling periods. For example, the data related to the feedback part of the control law must be updated faster than the ones of the feedforward parts. It has been shown also that the performance of the control is influenced by the more or less tight coupling of the cooperative tasks, according to their respective durations ([2] [14]).

We therefore provide the RT designer with a set of 8 communication and synchronization mechanisms to be run by the pairs of MT ports, providing the following services (P means “Producer” and C means “Consumer”):

- P/asyn-C/asyn: Each task is running freely, and is never blocked by the communication.
- P/syn-C/syn: The first task to reach the rendez-vous is blocked until the second one is ready.
- P/asyn-C/syn: The producer is running freely. The consumer is blocked until the next data production except when a new data has been produced since the last consumption.
- P/syn-C/asyn: A symmetric protocol: the producer is blocked until a new reading if there was no pending demand; the consumer is running freely.
- P/asyn-C/synF: The producer is running freely, the consumer is always pending until a new data production.
- P/synF-C/asyn: The producer is synchronized with a new demand while the consumer is running freely.
- P/asyn-C/synD: The producer is requested to send data by the consumer.

• P/asyn-C/synDF: The producer sends its next producted data upon request of the consumer.
These protocols were encoded using the synchronous language ESTEREL [30] (see section 3.4.3).

3.3 The Related Data Structure: an Object-Oriented Model

An object-oriented approach was selected to model Robot-Tasks for the following reasons:
• the set of all possible choices of models and the set of all task functions to be reasonably proposed lead to a finite but large number of possibilities to be structured;
• the design of a control law, which is not trivial, is done through the use of a dedicated Human-Machine Interface (HMI) which is itself Object-Oriented;
• because of the complexity of the general control scheme, any modification should be done without disturbing the coherence of the overall scheme;
• most functional components of the system may be easily described by tree models with natural inheritance properties.

3.3.1 The Classes

The ORCCAD system presently handles the following classes, all detailed in [22]
• four classes related to the control scheme (4): task functions, trajectory generation, models, controls;
• two classes related to the physical entities involved by the system: physical resources, components;
• two classes related to the event-based behavior of the Robot-Task: observers of the elementary servoing tasks, automaton of the robot task.

As an example, figure 3 presents the class hierarchy of the class "task-function". Appendix 1 gives a few expressions of the considered task-functions.

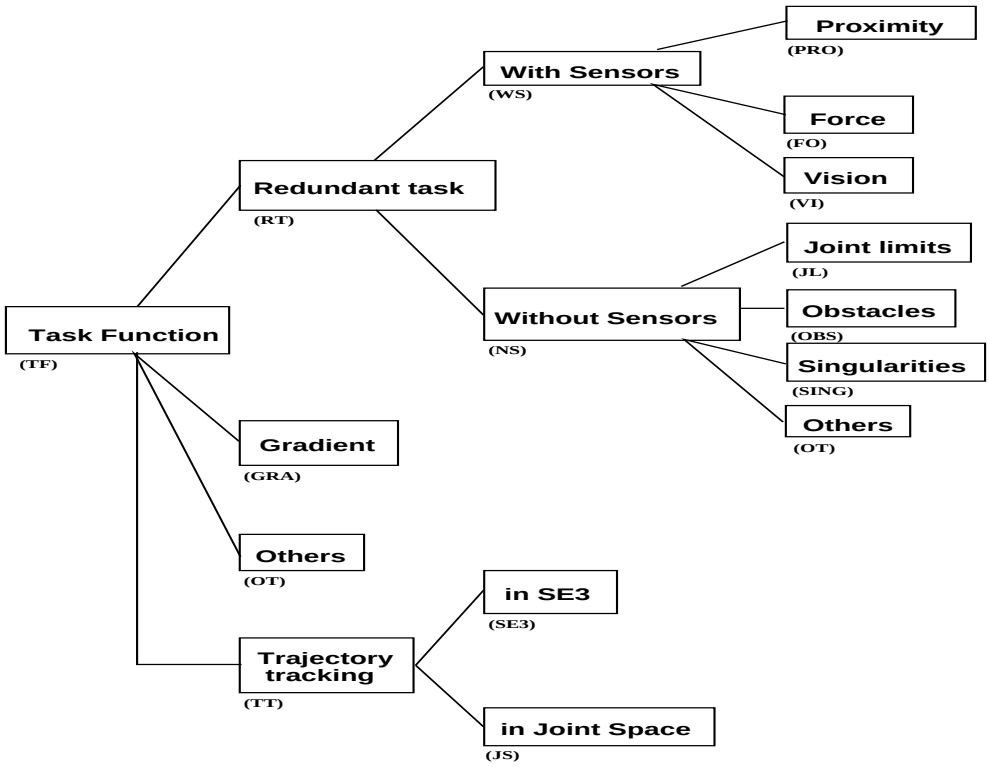

Figure 3: Hierarchy of the class Task-Function

3.3.2 Objects and Graphs

As detailed in the next section, a terminal leave in any of the class hierarchies is a single object with two kinds of attributes: the attributes relative to the *continuous-time specification* and the attributes associated with the *temporal properties*. When considering only the first ones, the object is partly instantiated, and called a functional object (*F-object*). When both types of attributes are considered, the instantiated object

is called a Module-Task object (*MT-object*). We may now define two important representations of a Robot-Task:

• *The* **functional graph** *of a Robot-Task is an oriented graph. Its nodes are the F-objects instantiated for the Robot-Task, and its edges correspond to the existence of formal data transfers between F-objects.*

• *The graph of time-constrained specification (***TCS graph***) of a Robot-Task is an oriented graph. Its nodes are the MT-objects instantiated for the Robot-Task, and each of its edges corresponds to the transfer of a variable from a Module-Task to another one.*

The functional graph is close to the classical representation called *block-diagram* in automatic control. The TCS graph allows the immediate visualization of the temporal properties associated to each MT by simply "clicking" into a port. Figure 15 gives an example of the TCS graphs attributes. The graphs, which may be easily edited by the user through the HMI, provides the user with a complete and synthetic representation of a Robot-Task.

3.3.3 Attributes and Methods

The objects in a Robot-Task include all information necessary to the generation of the functional and TCS graphs. Therefore, two sets of attributes are defined for every object, briefly:

i) the functional set (name and constant non-temporal parameters of the MT; names of the MTs to be connected) and

ii) the TCS set (period and duration of the MT; communication issues between MTs, i.e. port characteristics (name, type, synchronization, name and type of exchanged variables, etc...); localization of associated codes (initialization, computation) and name of the dedicated processor.)

Finally, three sets of methods operate on these attributes: test of coherence between parameters (for example, $\dim(e)=\dim(q)$ in equation (4)), automatic generation of the MT's C++ code and generation of the data required by the simulation system.

3.4 The Event-based Behavior

In a sense, a Robot-Task is atomic for the *application* designer. However it follows an internal sequencing which the designer has not to see in normal (failure-free) circumstances. Nevertheless the RT has to exchange information with other RTs, in order to synchronize and/or condition their activation. In the ORCCAD framework these two aspects are considered in a single way. Thus the RTs can be also considered as *reactive* systems ([25]) and the event processing can be programmed using the synchrony assumption ([7]): signals are emitted from and to an automaton which specifies the Robot-Task behavior. This automaton, called RTA, is encoded using the synchronous language ESTEREL ([8]).

3.4.1 Event Generation

Signals are emitted by objects from the "observers" class. Formally, an observer is: *as an F-object, an object belonging to the single class allowed to communicate with the RTA class* and *as an MT-object, any Module-Task provided with at least one output port handling an "event-type" variable.*

Note that, due to this definition, some objects may also become observers (i.e. belong to two different classes by multi-instantiation) by adding the adequate port. Let us give some examples of the observers defined in the class hierarchies: one may monitor the task function, in order to verify that the error remains small, the occurrence of singularities (in the task-jacobian or issued from the kinematics), the approach of a joint or actuator limit, the evolution of some selected variables, the output of an external sensor (part presence, obstacle detection, DC motor overheating...), etc. All the generated exceptions are handled by the RTA.

3.4.2 Signal Handling

In ORCCAD, *all* signals and associated processing must belong to well-defined categories.

Signals. We distinguish:

• *the pre-conditions.* Their occurrence is required for starting the servoing task. They may be:

_ pure synchronization signals: flags, signals associated with a rendez-vous.
_ signals related to the environment (also called measurement pre-conditions). An information issued from the environment is here required, usually through a sensor (part presence, physical resource availability...).
•*the exceptions.* They are exclusively emitted by observers in case of failure detection (see section 4.2).
•*the post-conditions.* Like pre-conditions they are of two kinds:
_ logical synchronization signals emitted by the RTA itself in case of correct termination.;
_ signals related to the environment, issued from a sensor: for example, final assembly force reporting.

Processing. We do not present treatments associated with pre- and post-conditions since they are quite simple. The exception processing is more specific of ORCCAD and is structured as follows:
• *type 1 processing*: the reaction to the received exception signal is limited to the modification of the value of at least one parameter in MT-objects. For example, when coming near to a task-jacobian singularity, the regularization parameter λ (see appendix 1) is progressively set from 0 to 1.
• *type 2 processing*: the exception requires the activation of a new Robot-Task. The current one is therefore killed. When the ending is correct, the nominal post-conditions are fulfilled. Otherwise, a specific signal is emitted towards the application, which *knows* the recovering process to activate. For example, if the robot reaches a joint limit while tracking the trajectory, a reconfiguration Robot-Task is started, then the previous one can be activated again.
• *type 3 processing*: the exception is considered as fatal. Then, everything is stopped. This occurs for instance when the error norm $\|e\|$ becomes quickly too large, inferring an actuator failure or a collision.

3.4.3 Implementation of the Robot Task Automaton

The logical behavior of the Robot-Task, specified in [38], is encoded using ESTEREL ([23]).

The use of ESTEREL Briefly, ESTEREL is a deterministic concurrent language dedicated to the programming of reactive real time systems. It is based on a model of synchronous parallelism and communication which allows a high level modular programming style which is simpler and more rigorous than in an asynchronous approach. ESTEREL has an efficient implementation based on well-defined mathematical semantics ([24]) . ESTEREL programs are compiled into equivalent labeled sequential automata upon which analysis and proofs can be performed using verification systems such as AUTO ([39]) and AUTOGRAPH ([34]). Some basic features of ESTEREL are briefly explained in examples provided in section 4.2.3 and a complete description of the language is given in [8].

The code of the Robot-Task automaton is distributed in modules: as a simple example, we give in figure 4a the ESTEREL code for a module awaiting the measurement post-condition POST_MES and protected by a watchdog triggered by WAIT_TIME. A more complex example of ESTEREL code is provided in section 4.2.3.

Automatic Code Generation In order to remain consistent with the whole ORCCAD philosophy, it is clear that the control designer should not have to write any ESTEREL code. Moreover, ESTEREL is an imperative language, sensitive to programming style, and requires a specific expertise. This is why the code involved in the RTA implementation is automatically generated in the ORCCAD system.

Therefore, to specify the event-based behavior of the Robot-Task, the user has only to instantiate the MT-object "RTA" using the HMI. For example, when defining an input port, the window of figure 4b must be filled up.

To demonstrate the efficiency of the system, let us indicate that a simple RTA specification using one measurement pre-condition, one exception of type-1, one exception of type-2 and one measurement post-condition leads to a 4 states and 13 transitions automaton (figure 5). Writing such an automaton directly would certainly have been less reliable and more tedious. Moreover, any modification is immediately taken into account by generating a new automaton.

4 Two Examples

We will now illustrate the ORCCAD approach through two examples. The first one deals with TCS design and simulation and the second one shows how ESTEREL can be used to program and verify RTs and Applications.

```
module POST−MES:
input
   POST−MES,
   TIME−GUARD;
output
   TIME−ELAPSED;
[   do
        await POST−MES;
    watching immediate TIME−GUARD
        timeout emit TIME−ELAPSED;
    end
] end module
```

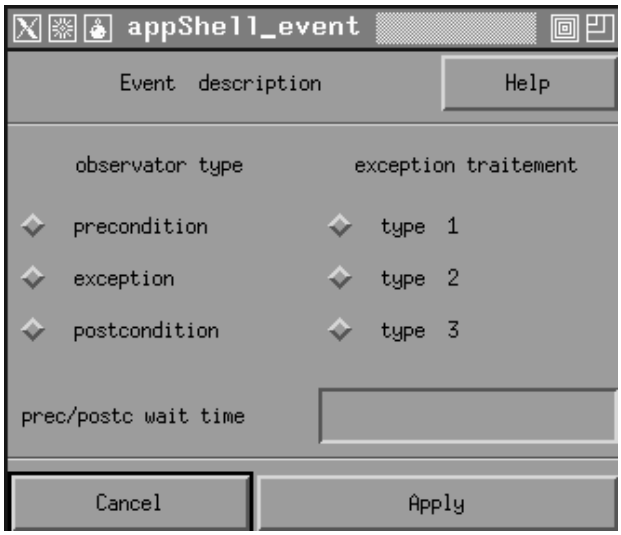

Figure 4: (a) Code of a Robot-Task Automaton module – (b) HMI panel for RTA specification

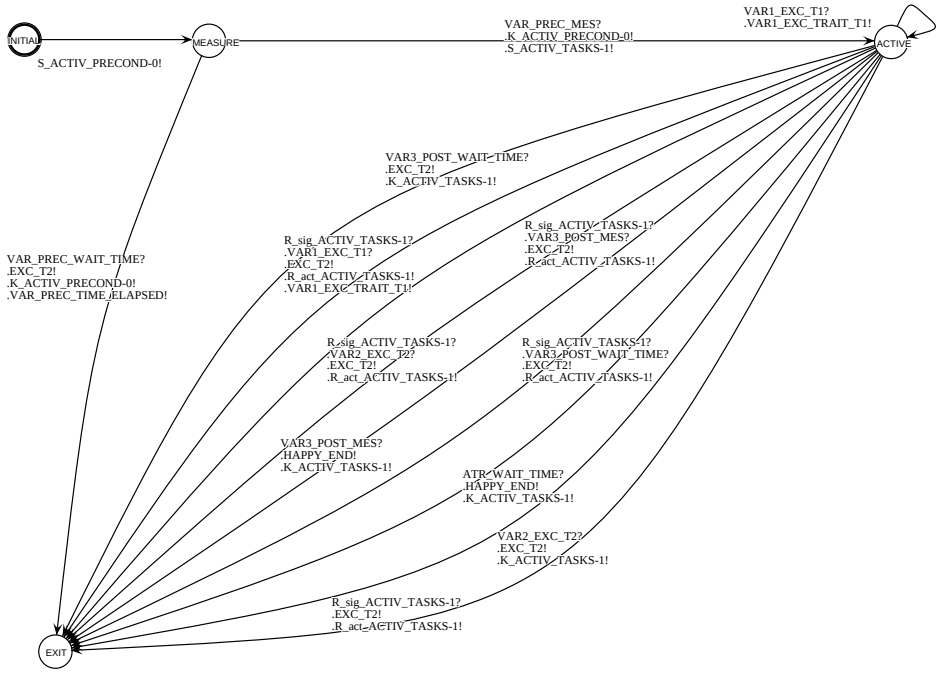

Figure 5: A Robot-Task Automaton

4.1 TCS Simulation of a Joint-Space Control

In this first example we consider a three degrees of freedom direct-drive robot. We present successively the implementation of a PID control law and of a Computed Torque control.

4.1.1 Robot-Task description

The PID control law has the advantage of requiring the knowledge of neither the model structure nor the model parameters to solve efficiently in most cases the regulation problem. The dynamic choices do be done in equation (4) are: $\hat{M} = I$ and $\hat{N} = 0$.
The task-function of a trajectory tracking in joint space can be easily written as: $e(q,t) = q - q_d(t)$ and therefore $\frac{\partial e}{\partial q}(q,t) = I$. The G and D tuning matrices can be fixed as identity matrices.

With the previous choices, the general form of the control (4) to which is added an integral action becomes:

$$\Gamma = k_p e(q,t) + k_v \dot{e}(q,t) + k_i \int e(q,t)dt \tag{5}$$

The trajectory to be tracked is specified as:

$$q_d = q_0(1 + cos(\pi t)), \quad \dot{q_d} = -q_0 \pi sin(\pi t) \tag{6}$$

with the initial position: ${q_0}^T = [.5,\ .5,\ .5]$ rad.
A single observer is associated to this RT. It raises a type 3 exception when a robot joint becomes near one of its limits.

4.1.2 The Control Block-Diagram

This control scheme is summarized figure 6a. The names of the variables to be transferred and the kinds of communication mechanisms are indicated for every port-to-port connection. The two parameters inside the boxes represent the activation period of the MTs and their assumed execution duration.

The **PR.MAN** box is only used for simulation purposes as a model of the physical system. It handles a state representation of the robot arm with the form:

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ M(q)^{-1}(N(q,\dot{q}) - \Gamma) \end{bmatrix} \tag{7}$$

to which are added "analog" functions like current loops, sensors non-linearities or torque limits.

4.1.3 Robot-Task Simulation and Control law refinement

The MTs durations of execution are estimated for a 16 MHz mc68020 processor. Several simulations were performed to find adequate MTs activation periods and to tune the PID parameters. To do this, the data processing code of the MTs was not modified; only the time constrained specification was updated using the HMI for a new simulation. Therefore, the refinement of the RT control law becomes an "easy" task and the MTs can be reused in another scheme using a different time constrained specification.
The PID simulation results were obtained with the following gain values:
$k_p = 2500 Nm\ rad^{-1}$, $k_v = 1000 Nm\ s\ rad^{-1}$ and $k_i = 40000 Nm\ rad^{-1}\ s^{-1}$.
Figure 7a shows the tracking error for the first joint, using different activation periods for the CO.JS control task. The tracking error decreases with the period, although the gains remain constant. As a counterpart, decreasing the period could allow decreasing the gains in order to achieve a given tracking error, allowing thus a reduction of control noise and avoiding excitations of flexible modes of the robot.

At the end of a simulation a timing diagram is given for each MT by the SIMPARC trace system. Figure 8 shows the diagram produced for the CO.JS MT, where dark areas denote active instants of the task. This diagram allows checking of the MT's timing constraints. It must be pointed out that, according to the type of synchronizations used, output data are not always sent synchronously with the completion of an active period.

4.1.4 A Computed Torque Control Law

Owing to the modularity of the ORCCAD system, it is easy to modify a RT. The PID control law was transformed in a Computed Torque Control law simply by adding a MT which computes more accurate models of the matrices M and N to the previous RT. Figure 6b shows the new block-diagram. The control law becomes:

$$\Gamma = M(q)(k_p e(q,t) + k_v \dot{e}(q,t)) + N(q,\dot{q}) \tag{8}$$

The gravity terms are taken into account in the computation of N so that we can drop the integral action.

Taking into account the main terms of the dynamics of the robot enables a smaller sampling rate of the control loop and allows a considerable reduction of the gains k_p and k_v. In the following simulations, the period of CO.FS has been set to 5 ms, k_p to $500 N m\ rad^{-1}$ and k_v to $120 N m\ s\ rad^{-1}$.

Figure 7b shows that the tracking error is not very sensitive to the sampling period of the MOD.DYN feedforward dynamics computation task. Although the values of control gains are moderate, the tracking error is smaller than the one obtained with the previous PID control, especially at high velocities. Figure 8a shows spikes on the control torque plot. These spikes are due to a kind of beating phenomenon between the control task and the dynamics computation task, which are not computed at the same rate. These spikes vanish when the sampling rate of MOD.DYN is decreased to the same value as the one of CO.JS.

4.2 ORCCAD-Oriented Specification of an Undersea Mission

4.2.1 Scheduling Robot-Tasks: the Application Level

The application level is for the end user. It relies on a set of parametrized Robot-Tasks, the logical and temporal organization of which defines its final behavior. From the end user's point of view, this level should remain far from the implementation issues. On the other hand, it has to cope with the user's concerns. In order to facilitate the communication with this user, these concerns have to be expressed using techniques, specific terminologies or description methods issued from the *application domain* and are necessarily different from one domain to another. Obviously, the ORCCAD system cannot handle by itself such a large set of possible application areas: the mission of a Mars Rover will never be described in the same way as the fitting of a windshield in automobile industry. Moreover, the application level would ideally have to be connected with various high level tools like planning or intelligent decision systems which for lie outside ORCCAD's domain.

Nevertheless, we may reasonably consider that an *invariant* representation scheme of an application exists somewhere when moving from the application to the implementation level. This is why the application features of ORCCAD are limited to such an *intermediary level*, aimed to describe in a precise way through an automaton the logical and temporal dependencies between the involved Robot-Tasks.

Again, in order to allow a dialogue with Robot-Task Automata, the intermediary level consists of an ESTEREL program. Just as ESTEREL code is automatically generated in RTs, it may be considered that the ESTEREL application program is the output of an higher level user-oriented language or interface depending on the application domain.

In the next section we present a detailed example of this upper level of ORCCAD.

As already mentioned, ORCCAD was primarily dedicated to manufacturing robotics. In particular, the HMI exploits the general control structure which has been exhibited for rigid robot arms. Such manufacturing applications may involve a rather large number of different actions, running in a structured and well known environment. Thus the application specification can be rather complex, while automatic recovery procedures might be simple since recovery actions may be performed by human operators. Programming mobile autonomous robots leads to a somewhat different situation. The set of possible actions of a mobile robot is generally small, while reliability is a major issue. Recovery procedures, using external sensors and state measurements, have to be very carefully designed in order to achieve the assigned mission, and to leave in all cases the robot in a safe recovery state. Simulating and/or proving the correctness of the mission before launching the robot in an uncertain environment is claimed to be necessary [27].

Although many differences exists between application specification in industrial plants and mobile robots, we will now demonstrate through a simple example that the ORCCAD approach may also be used in the design of control software for autonomous vehicles.

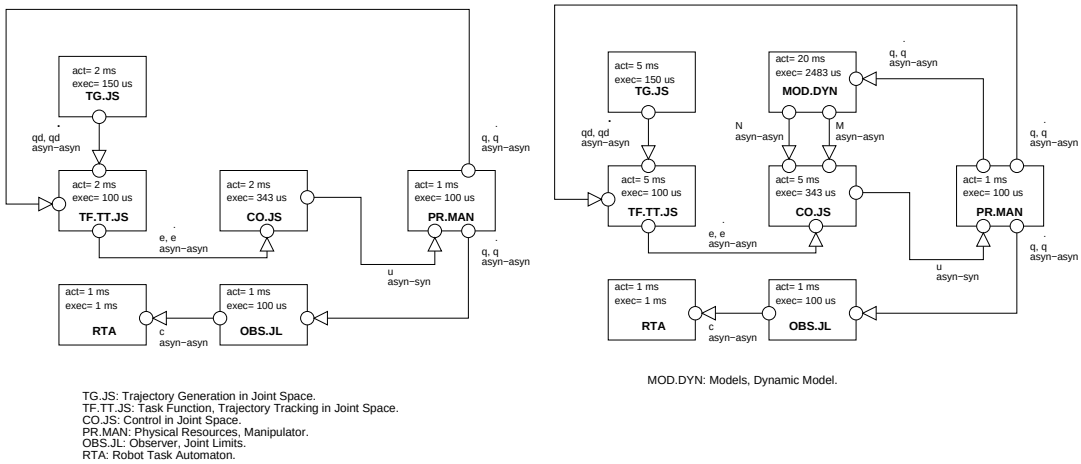

Figure 6: TCS bloc-diagrams (a) PID control – (b) Computed torque control

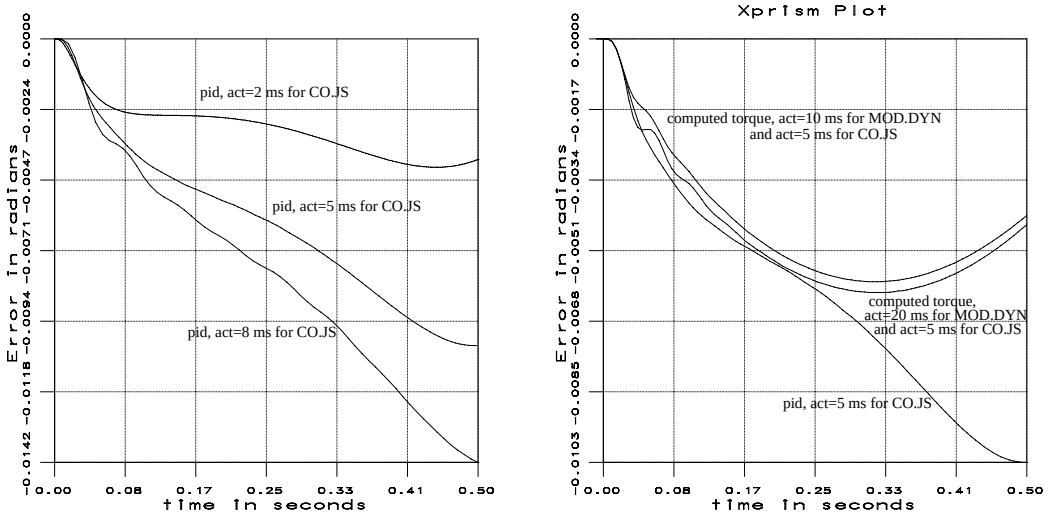

Figure 7: Tracking errors: (a) PID control – (b) Computed torque control

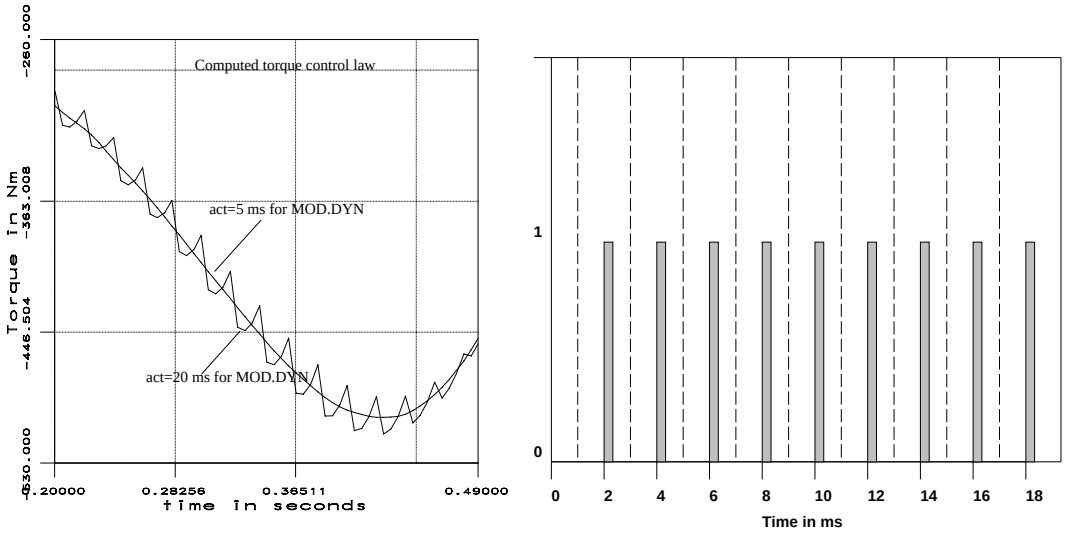

Figure 8: (a): Control torques – (b)Timing diagram

4.2.2 Mission Specification

This example was provided by J. Bellingham [6] to illustrate the "State Configured Layered Control approach". It is a mission specification for the Sea Squirt autonomous underwater vehicle developed at M.I.T.

The nominal mission is described by the following sequence of actions:

- transit to a user-defined waypoint using dead reckoning;
- dropping a transponder at the way point;
- spiral search around the transponder until the target is found;
- homing towards the home transponder;
- surface for recovery.

During the mission, the vehicle must avoid obstacles and shallow water. Several exception situations are specified:

- homing if a low-power situation is detected during transit or spiral search;
- homing if the target is found during transit;
- surface if a very-low-power alarm is triggered at any time.
- homing using dead reckoning if the transponder signal is lost. Go back to homing on transponder if the signal is recovered.

This specification is summarized by the state transition table of figure 9a, taken out from [6].

4.2.3 A Possible Specification Under ORCCAD

The control software should be organized around four RTs: TRANSIT to a way point using dead reckoning, SPIRAL search around a transponder, HOMING to the home transponder, HOMING_DR using dead reckoning. Here, the MTs correspond to the behaviors of the original layered approach and the RTs to the grouping of layers into states.

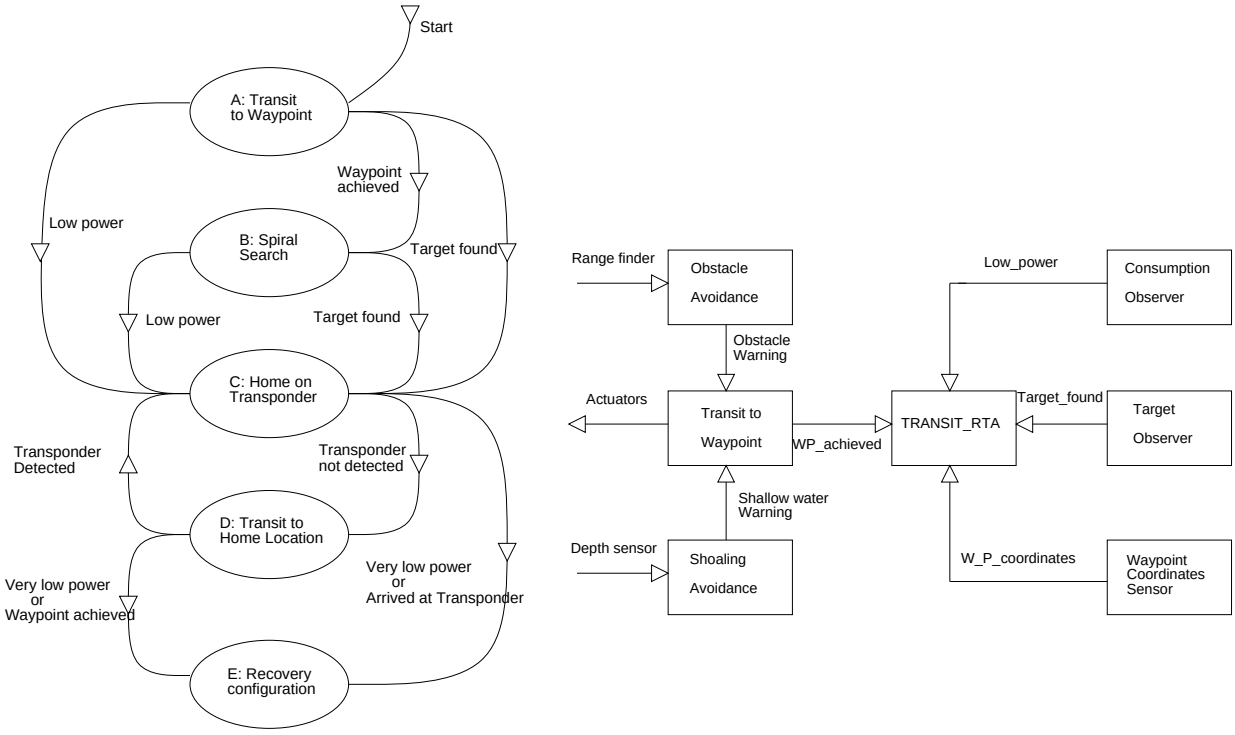

Figure 9: (a): Mission specification – (b)Structure of Transit

The internal structure of the proposed Transit RT is given figure 9b where:

- Transit_to_Waypoint is the main control task. In fact, it could itself be split into smaller MTs like autopilot or trajectory generation... Its output, WAY_POINT_ACHIEVED, is connected to the Automaton and triggers a *logical post-condition.*
- Obstacle-Avoidance and Shoaling-Avoidance are secondary periodic tasks. Their output are *Type 1 exceptions* altering the trajectory generator of Transit_to_Waypoint. They are connected to the main transit

task rather than the automaton in order to improve closed loop control efficiency. These two MTs are reused in the three others RTs with different connections on their ports.
• two observers check for LOW-POWER and TARGET-FOUND and trigger *Type 2 exceptions.*
• a valued signal is used to read the waypoint coordinates. This is a *measurement pre-condition.*
• Since Transit()() is a control task, it is assumed to be an endless loop and must be killed explicitly. A spontaneous stopping of the task is considered as a serious failure and raises a *Type 3 exception.*

The Automata of the RTs are triggered by the main mission program, and trigger themselves all the necessary MTs. Following the structure given in section 3.4.3, an example of ESTEREL code corresponding to the local behavior of the Transit RT is given figure 10 (declarations have been omitted for clarity).

```
await NEW_COORD; WP_COORD := ?NEW_COORD; %measurement pre-cond.
trap CRASH in %exit mechanism declarations
trap T1 in
  trap T2 in
       trap ARRIVED in
       [
       exec TRANSIT()(WP_COORD); exit CRASH; %triggers all the MTs
       ||
       await LOC_LOW_POWER; exit T1;
       ||
       await LOC_TARGET_FOUND; exit T2;
       ||
       await LOC_WP_ACHIEVED; exit ARRIVED;
       ]
       handle ARRIVED do emit WAY_POINT_ACHIEVED
       end %normal completion
  handle T2 do emit TARGET_FOUND
  end %type 2 exception
handle T1 do emit LOW_POWER
end; %type 2 exception
handle CRASH do emit FATAL  %type 3 exception
```

Figure 10: Source code for the automaton of transit

Let us comment this typical piece of code. Signals are basic entities in an ESTEREL program: they can be locals or used as the interface with the environment. They can be emitted or awaited using the **emit** and **await** statements, and are broadcasted inside their declaration scope.

The **exec()()** statement is aimed to manage asynchronous extern tasks ([33]). For each exec statement, the compiler provides three interface functions: the start() and kill() functions are used to activate or suspend asynchronous tasks while the return() function signals to the automaton the completion of the extern task. The body of these functions must be filled by the user (or by the HMI) with calls to the operating system in order to manage all the real-time MTs to be activated within the current RT.

The **trap...exit...handle** statement is an exit mechanism fully compatible with parallelism. It is extensively used in the RTs structure to manage exception handling. Another preemptive statement of ESTEREL is the **do...watching** statement, the basic construct in building watchdogs. Since physical time has no particular meaning in ESTEREL, watchdogs can be triggered by *any* kind of signals.

From the mission specification point of view, the end user is provided with a library of RTs. For example, the public declaration of Transit looks like figure 11. Using this library, the ESTEREL mission source program for the squirt mission can then be written straightforward (the declarations are discarded in the source program given figure 12).

In this example, the VERY_LOW_POWER alarm signal must be checked during the whole mission execution. Therefore, it is directly handled at the mission specification level and triggers a watchdog embedding the nominal mission. Signals which can be emitted to the hardware in order to perform very simple operations, like dropping a transponder, are also handled at the mission level. The **exec** statements in the

mission specification activate the Robot-Task Automata, and are also used to pass parameters to the RTAs.

The preemption mechanisms provided by ESTEREL are here very useful to encode the fault recovery behaviors embedding the nominal mission. Another very useful statement is the parallel construct (||). It is extensively used inside the RTs to wait for observers outputs. It can also be used at the application level, for example in splitting the main program in two parts, one related to generic actions like safety functions and recovery behaviors, the other being related to the specific part of the mission.

At compile time, this mission specification is translated into the Application Automaton displayed by AUTOGRAPH (figure 13) where the transitions are labeled with signals receptions (?) and emissions (!). The simulation package provided with ESTEREL may also be used for mission debugging (figure 14).

A major advantage of synchronous languages is that they are deterministic so that formal verification of programs can be performed: behaviors which are crucial from the reliability point of view can be checked off line. For example, looking at this automaton, one can easily check that in every state but the first one, the VERY_LOW_POWER signal is awaited and that its occurrence always drives the control system to the recovery configuration. Note that this property was not true on the original, *hand-made* transition diagram given by figure 9a. Obviously, building this Application Automaton through an high level language compiler is less tedious and error prone that writing it directly.

Such a programming approach, where the application is completely pre-defined and where on-line re-planning would be difficult (due to the compilation process of ESTEREL), can be compared to the Rational Behavior Model-Forward paradigm [13], where the strategy is determined *a priori* by the designers. However the RBM-F model may lead to conflicting transition paths, due to competing exception events. These conflicts must be explicitly solved, using for example Prioritized State Transitions Diagrams. Using ESTEREL, exceptions are handled by nested traps and watchdogs, the highest priority being assigned to the most external embedding exit mechanism. Thus, conflicts are solved at compile time in a *deterministic* way.

5 Software environment of ORCCAD

5.1 Human-Machine Interface

In all CAD systems, the efficiency of the Human-Machine Interface affects the overall performance of the design system. In ORCCAD, the HMI is also a key tool, presently oriented towards users with an automatic control background. Owing to its connection with the SIMPARC simulator (see section 5.2), it allows the easy design and test of a Robot-Task through dedicated interfaces (figure 15).

The basis of the HMI is the object-oriented structure presented in section 3.3. Any specific action makes dedicated windows to appear where the user, guided by the system, enters its specifications and instantiates the leaves of the hierarchies called *F-objects*, defined formally in section 3.3.2. Two *F-objects* can be connected if a set of tests of coherence is satisfied. The control law is specified in continuous-time.

The specification of the port characteristics, as well as the temporal attributes of the leaves of the hierarchies, transforms the continuous-time model in the TCS model. Once the algorithmic and temporal attributes and the physical behavior of the system to control have been defined, the user can proceed to a temporal simulation. The HMI generates automatically the data processing code for every *MT_ object* and creates the environment reported for this simulation.

```
task TRANSIT_RTA(R1,R2,R3,R4)(V1)
Function:  Transit to a way-point using dead reckoning, obstacle
avoidance and shallow water avoidance
Call parameters:  V1 (coord):  coordinates of the way point
typedef struct{float, float} coord;
Exit conditions and Return parameters:
R1 (boolean) true if LOW_POWER type 2 exception
R2 (boolean) true if TARGET_FOUND type 2 exception
R3 (boolean) true if WAY_POINT_ACHIEVED synchro.  post-cond.
R4 (boolean) true if FATAL_ERROR type 3 exception
```

Figure 11: Public declaration of the Transit RT

```
 await START; % init signal
 WP_COORD := ?WAY_POINT; %reading way-point coordinates
 trap CRASH in
 trap STOP in
        do %begin a watchdog
        exec
TRANSIT_RTA(LOW_POWER,TARGET_FOUND,WAY_POINT_ACHIEVED,FATAL)(WP_COORD);
        if (FATAL) then exit CRASH
        else if (WAY_POINT_ACHIEVED and (not TARGET_FOUND) and (not
LOW_POWER))
        then emit TRANSPONDER_DROPPED;
               exec SPIRAL_RTA(LOW_POWER,TARGET_FOUND)()
        end if; end if;
        %the following loop toggles between homing using a
        %transponder and homing using dead reckoning
        trap HOMING in
        [loop
                exec HOMING_RTA(AT_HOME, TRANSPONDER_LOST)();
                if (AT_HOME) then exit HOMING end;
                WP_COORD := ?HOME;
                exec
HOMING_DR_RTA(AT_HOME,TRANSPONDER_RECOVERED)(WP_COORD);
                if (AT_HOME) then exit HOMING end;
        end loop;]  %mission achieved
        watching VERY_LOW_POWER timeout exit STOP end; %watchdog
 end trap; %STOP
 end trap; %CRASH
 emit SHUT_DOWN; %hardware signal to drop safety ballast
 end module
```

Figure 12: Source code for the mission specification

Finally, for achieving the hardware simulation and the downloading of the code generated, further attributes related to the processing boards and to the operating system calls used to instantiate real-time tasks must be added to the *MT-objects*.

5.2 Simulation Software

Once a RT has been specified with ORCCAD, two simulation phases are required to validate the choices made by the designer: the first one deals with evaluation of the TCS properties while the second is very close to implementation and is the last step before downloading.

Inside the general control scheme (4), the RT designer has to make many choices about the models to be used and the temporal properties assigned to the MTs. Sometimes, simple but quickly computed models could lead to better overall performance than a slower sophisticated control law. As it does not appear that general answers exist, these choices will have to be checked for every new design.

Control systems, and more especially robotics systems, are *hybrid* systems from the time point of view. The controlled system belongs to the physical world, and can generally be described by a set of differential or partial derivative equations in continuous time. On the other side, the controller works basically in the frame of discrete time. SIMPARC (Simulator for Multiprocessors Advanced Robot Controllers) ([3], [4]) was originally designed from scratch to handle these two aspects within a *single* simulation system. This simulation tool actually runs the user's control programs on the simulated target architecture which is described as a set of communicating devices like processors, buses, converters and physical processes and sensors. Figure 16 shows the general organization of the software.

Simulation results are available from the two sides of the software: on one hand, the user may select

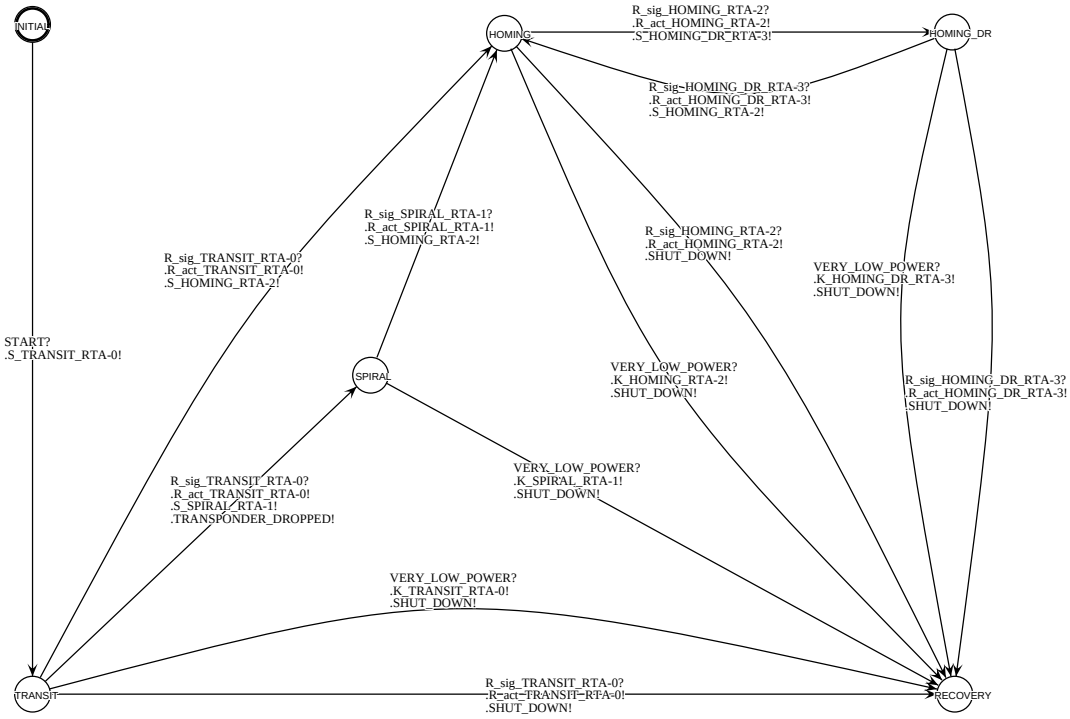

Figure 13: Application Automaton

events of interest to be monitored and stored by SIMPARC. The analysis of the generated file allows to trace relevant variables like CPU's load or bus contention and can help for task placement. Besides, data coming from the continuous system simulation can be displayed. The analysis of tracking errors or of control amplitude may for example help for tuning control loops gains.

The SIMPARC software allows the simulation of user's programs which are very close to actual implementation, thus further small effort will be necessary to produce code for downloading.

During the early steps of a RT design, only MTs and ports have to be considered by the TCS designer in order to make easier the *fast prototyping* of control laws. These two kinds of objects are handled through the HMI, and the simulation code is then automatically generated. Therefore, all the basic mechanisms of the SIMPARC kernel, which are somewhat tricky to use, are hidden. Examples of TCS simulations are provided in section 4.1.

6 Conclusion

In the area of robot control, the ORCCAD system is original since it proposes a *coherent approach* from the implementation level to the application specification level. In particular, the end-user of the system is provided with powerful and well defined primitives: the Robot-Task concept encapsulates in a single object both *reflex* actions related to control laws and *reactive* behaviors related to discrete time events. Encoding reactivity with the same synchronous language at both application and control levels allows the establishment of strong links between them and the performance of sound verifications of programs. Finally, real-time efficiency and reliability issues are considered at all the levels of the system.

The modularity of the approach, the control schemes it implements, the use of the ESTEREL language and its design and validation tools (HMI, simulators) make of ORCCAD an open and powerful system which can be successfully compared to other robot control design approaches. Nevertheless, the problem of designing a general robot controller (in the sense of the definition of section 2.1) is wide enough to ensure that relevant work remains to be done in order to extend the possibilities of the present version of ORCCAD.

A first point was already evoked: the ESTEREL language should be used as an intermediary level of the application specification, not generally open to the end-user, who would like to express his requirements in a more simple and natural way. Therefore it would be interesting to design, for a given application

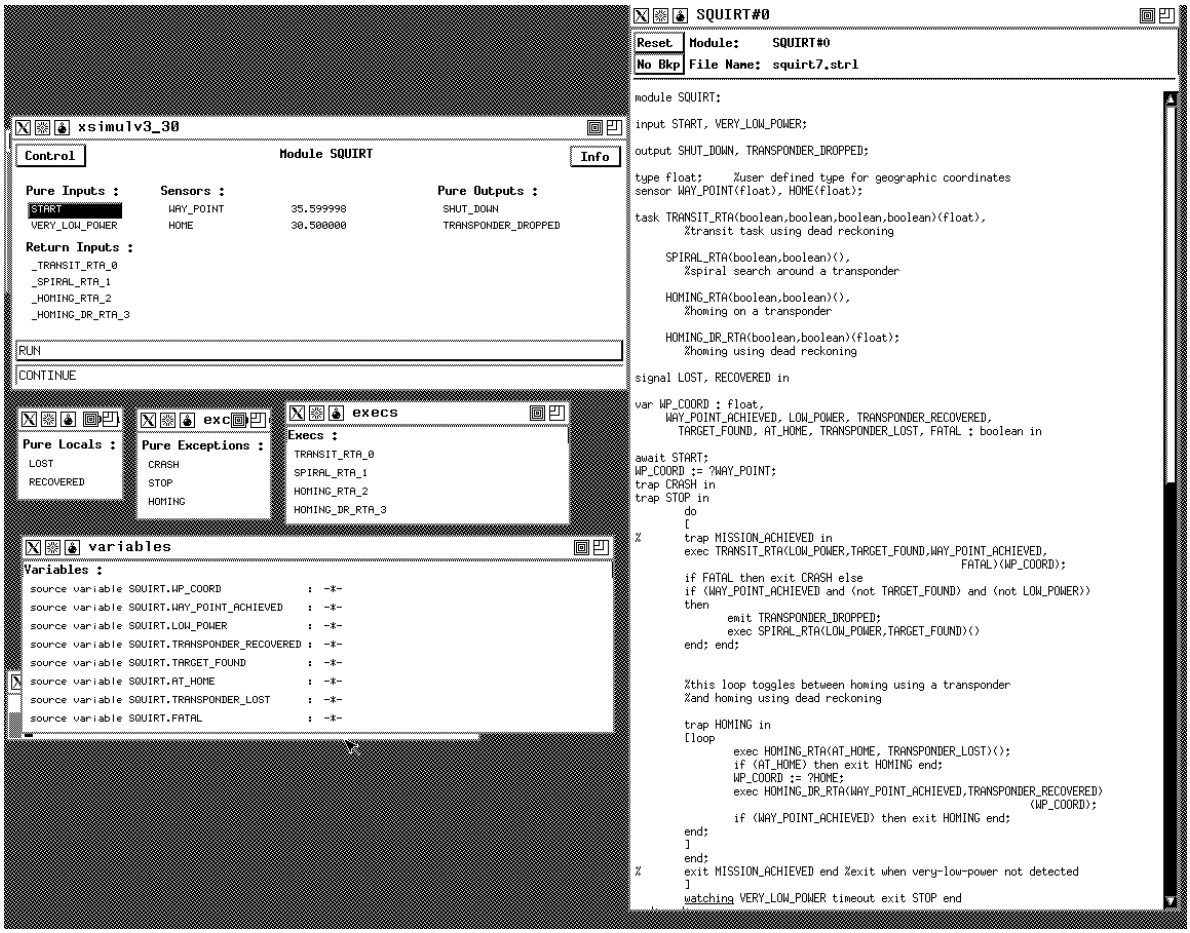

Figure 14: Simulation panel for ESTEREL

domain, a set of ESTEREL-encoded primitives expressing in a friendly way, even with some restrictions, the user's needs in terms of parallelism and synchronization. An application-specific interface would thus be realized. Moreover, the connection with a possible planning level would then be easier. As an example, let us mention that a first version of an application programming language is proposed in [17] and [18]. Aimed at the specification of plans of actions at the level of sets of RTs, this language is provided with imperative control structures and is defined in terms of temporal logic. A compiler translates plans written in the application language into ESTEREL code corresponding to their temporal definition. In this way the end user of ORCCAD is offered a language enabling him to express a sequencing of Robot-Tasks with no need to investigate the details of the signal exchanges, dialogues and implementation issues for each particular arrangement of tasks. However, the problem of *on-line* planning in the ORCCAD approach remains non-trivial and deserves to be investigated.

Let us emphasize again that ESTEREL allows the use of formal proof systems such as AUTO. Absence of deadlocks or properties more specific of the application may be checked. A debugger and an automata simulation system are also available. Although already used, these tools are not yet well integrated in the ORCCAD framework. For example, the information handled by ESTEREL may not directly be understandable by an end user in terms of natural variables. A non-trivial work to allow this matching remains to be done. To end with ESTEREL, let us mention that some recent works open still more interesting possibilities like links with the formalism of CSP and compiling ESTEREL programs into hardware circuits.

Let us now take a sight nearer to the implementation. Here also, formal proof tools for checking the synchronization graph of the Module-Tasks are necessary. This difficult problem is not yet solved and is presently under investigation. At the implementation level, the problem of downloading is in turn easy to address: now the simulation code can be automatically generated. By using similar methods, the generation of executable code does not seem to be very difficult, although this development remains to be done. Besides, let us note that the ORCCAD requirements in terms of real-time primitives are minimal. Only a small subset of primitives, existing in most of commercial Operating Systems, is therefore needed.

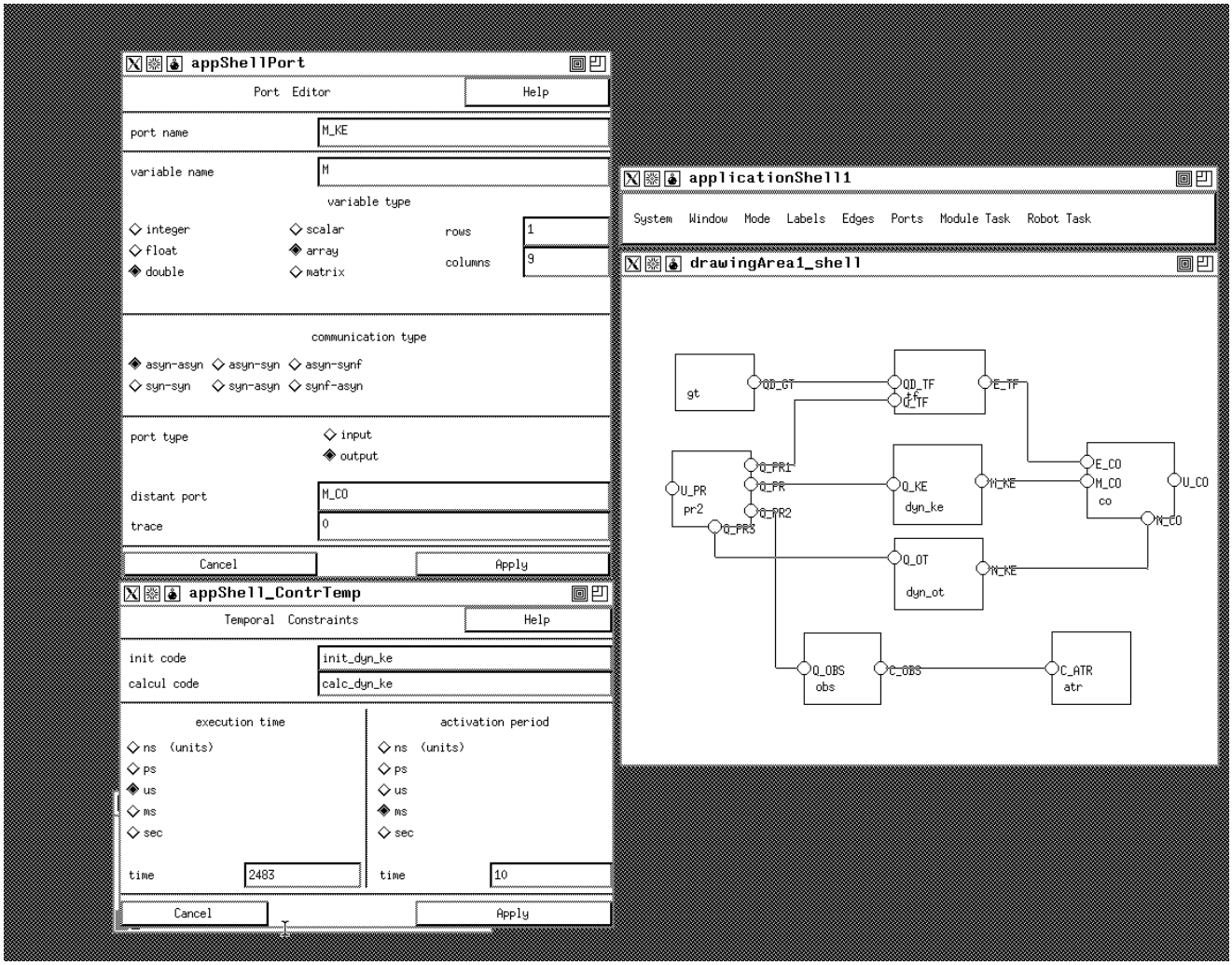

Figure 15: HMI panel for TCS specification

A final point, which concerns the extensions of ORCCAD, is the one of control algorithms associated with Robot-Tasks. The present control scheme covers the area of rigid robot manipulators. It is now necessary to propose other Robot-Task structures, dedicated for example to well-defined classes of autonomous mobile robots.

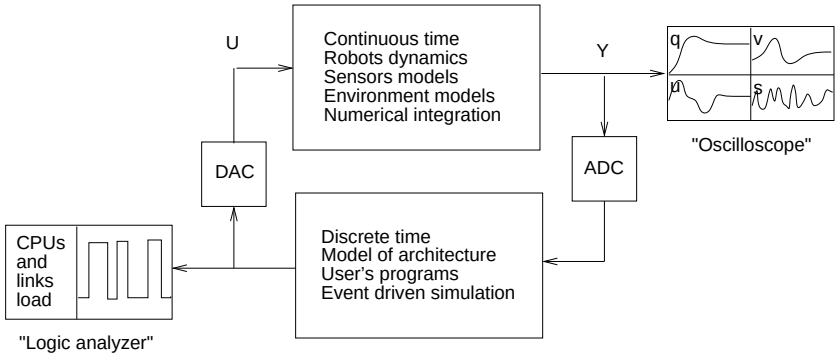

Figure 16: General Organization of SIMPARC

References

[1] J.S. Albus, R. Quintero, "Toward a Reference Model Architecture for Real-Time Intelligent Control Systems (ARTICS)", *Robotics and Manufacturing*, vol.3, Jamshidi and Saif Editors, ASME Press Series 1990

[2] B. Armstrong-Helouvry, "Latency Analysis of Asynchronous Distributed Control", Internal Report, Department of Electrical Engineering, University of Wisconsin, Milwaukee, 1992

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

[4] C. Astraudo and J.J. Borrelly, "Simparc reference manual", Esprit II ARMS project, public deliverable WP 4320 DELBL 6, November 1991.

[5] P. Backes, S. Hayati, V. Hayward and K. Tso, " The Kali Multi-Arm Robot Programming and Control Environment" in *Proc. of the NASA Conference on Space Telerobotics*, G.Rodriguez and H. Seraji editors, JPL Publication 89-7, January 1989.

[6] J.G. Bellingham, T.R. Consi, "State Configured Layered Control, *Proc. 1st Worshop on Mobile Robots for Subsea Environments*, pp 75-80, Monterey, October 1990.

[7] A. Benveniste and G. Berry, "The Synchronous Approach to Reactive and Real-Time Systems", *Proceedings of the IEEE*, vol. 79, no 9, September 1991.

[8] G. Berry and G. Gonthier, "The Synchronous Esterel Programming Language : Design, Semantics, Implementation",*Science of Computer Programming*, vol 19, no 2, pp 87-152, Nov. 1992.

[9] G.A. den Boer, E. Gaussens e.a., "An Exception Handling Model applied to an Autonomous Mobile Robot." *IAS3 conference*, Pittsburgh, 1993.

[10] J.J. Borrelly and D. Simon, "Propositions d'Architecture de Contrôleur Ouvert pour la Robotique", INRIA Research Report no 1304, October 1990.

[11] M. Boyer, L. Daneshmend, V. Hayward, A. Foisy: An Object-Oriented Paradigm for the Design and Implementation of Robot Planning and Programming Systems, *IEEE Int. Conf. on Robotics and Automation*, Sacramento, 1991.

[12] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, vol 2, no 1, pp 14-23, March 1986.

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

[14] J.B. Chen, B.S.R. Armstrong, R.S. Fearing and J.W. Burdick, "Satyr and the Nymph: Software Archetype for Real Time Robotics", *Proc. IEEE-ACM Joint Computer Conference*, Dallas, November 1988.

[15] E. Coste-Manière, "Synchronisme et Asynchronisme dans la programmation des systèmes robotiques: apport du langage ESTEREL et de concepts objets", PhD Dissertation, Ecole des Mines de Paris, France, July 1991.

[16] E. Coste-Manière, B. Espiau and D. Simon, "Reactive Objects in a Task-Level Open Controller, *Proc. IEEE Conf. on Robotics and Automation*, pp 2732-2737, Nice, France, May 1992.

[17] E. Coste-Manière, B. Espiau and E. Rutten, "Task-Level Robot Programming Combining Object-Oriented Design and Synchronous Approach: a Tentative Study", INRIA Research Report no 1441, May 1991.

[18] E. Coste-Manière, B. Espiau and E. Rutten, "A Task-Level Robot Programming Language and its Reactive Execution", *Proc. IEEE Conf. on Robotics and Automation*, pp 2751-2756, Nice, France, May 1992

[19] J.L. Crowley, "Layered Control for Intelligent Robotic Devices", *IEEE Conf. Rob. and Aut., Workshop on Architectures for Intelligent Control Systems*, pp 55-60, Nice, France, May 1992.

[20] B. Espiau, F. Chaumette and P. Rives, "A New Approach to Visual Servoing in Robotics", *IEEE Trans. Rob. Auto.,* vol 8, no. 3, pp. 313-326, June 1992.

[21] B. Espiau, J.P Merlet and C. Samson, "Force-feedback Control and Non-contact Sensing: a Unified Approach", *Proc. 8th Symp. IFTOMM RoManSy*, Krakow, Poland, July 1990

[22] B. Espiau and K. Kapellos, "Robot Task Specification", Internal INRIA-Aleph Working Report, January 1992

[23] Esterel V3 Language Reference Manuel, CISI Ingéniérie, Valbonne, France, 1991

[24] G. Gonthier, "Modèles d'exécution des langages réactifs synchrones: application à Esterel", PhD dissertation, Université d'Orsay, France, 1988

[25] D. Harel, A. Pnueli, "On the development of Reactive Systems", Research report, the Weizmann Institute of Science, Israel, January 1985.

[26] S.Y. Harmon, "Architectures: Designers vs. Implementors", *IEEE Conf. Rob. and Aut., Workshop on Architectures for Intelligent Control Systems*, pp 1-6, Nice, France, May 1992.

[27] A.J. Healey, R.B. McGhee e.a., "Mission Planning, Execution and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle", *Proc. 1st Worshop on Mobile Robots for Subsea Environments*, pp 177-186, Monterey, October 1990

[28] J. Ish-Shalom, P. Kazanzides, " SPARTA: Multiple Signal Processors for High-Performance Robot Control", *IEEE Transactions on Robotics and Automation*, vol 5, no 5, October 1989.

[29] E. Mazer e.a., "ACT: A Robot Programming Environment", *Proc. IEEE Int. Conf. on Robotics and Automation*, Sacramento, April 1991.

[30] M. Mejia, D. Simon, P. Belmans and J.J. Borrelly, "Mécanismes de synchronisation dans un systeme robotique réparti", *Proc. Séminaire Franco-Brésilien sur les systèmes informatiques répartis*, Florianopolis, Brazil, September 89.

[31] M. Merleau-Ponty, "La Structure du Comportement", Presses Universitaires de France, Paris 1942.

[32] S. Narasimhan, D.M. Siegel and J.M. Hollerbach, "Condor: An Architecture for Controlling the Utah/MIT Dexterous Hand", *IEEE Trans. on Rob. and Aut.*, vol 5 no 5, October 1989.

[33] J.P. Paris, "Exécution de tâches asynchrones depuis Esterel", PhD dissertation, Université de Nice, France, 1992.

[34] V. Roy, "AUTOGRAPH, Un Outil de Visualisation pour les Calculs de Processus", PhD dissertation, Université de Nice, France, 1990.

[35] C. Samson and B. Espiau, "Application of the Task-Function Approach to Sensor-based Control of Robot Manipulators", *Proc. IFAC 1990*, Tallinn, CCCP, Aug. 1990

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

[37] D. Simon and A. Joubert," ORCCAD: Towards an Open Robot Controller Computer Aided Design System", INRIA Research Report no 1396, February 1991.

[38] D. Simon, B. Espiau, E. Castillo and K. Kapellos, "Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues", INRIA Research Report no 1801, November 1992.

[39] R. de Simone and D. Vergamini " Aboard AUTO", INRIA Research Report no 112, October 1989.

[40] D.B. Stewart, D.E. Schmitz and P.K. Khosla, "CHIMERA II: A Real Time Multiprocessing Environment for Sensor Based Control", *Proc. IEEE International Symposium on Intelligent Control*, Albany , September 1989.

[41] J.B. Watson, "Psychology as the Behaviorist Views It", *Psychological Review*, vol XX, 1913.

A Appendix: Main Task Functions

A.1 Introduction

We will not give many details on task functions in this brief appendix. We prefer to refer the reader to [36] for the whole theoretical approach, and to references cited at the end for sensor-based applications. In ORCCAD, the general form of a task function is

$$e(q,t) = e'(q,t) + \lambda(t)(q - y(t)) \tag{9}$$

where $\lambda(t)$ is a regularization parameter in the following sense. We have:

$$\frac{\partial e}{\partial q} = \frac{\partial e'}{\partial q} + \lambda(t) I_n \tag{10}$$

and choosing $\lambda(t)$ as zero when $\frac{\partial e'}{\partial q}$ is nonsingular, and positive enough elsewhere ensures that $det\ (\frac{\partial e}{\partial q})$ is nonzero everywhere. In order to avoid in the last case too much disturbance of the original task function, $y(t)$ may be given by the filter

$$\dot{y}(t) + \alpha(t) y(t) = \alpha(t) q(t) \tag{11}$$

or more simply by $y(t) = q(t - \Delta)$. These expressions tend to keep the joint velocities small when crossing a singularity ([35]).

Let us now give some examples of classical, or less classical, task functions.

A.2 Trajectory Tracking

A.2.1 In Joint Space

It is simply:

$$e'(q,t) = q - q_d(t) \tag{12}$$

where $q_d(t)$ is the desired trajectory.

A.2.2 In $^3 \times SO_3$

Let us consider the case where a frame linked to the end effector of a six jointed robot should track a given trajectory. We then have:

$$e'(q,t) = \begin{pmatrix} x(q) - x_d(t) \\ O(R(q), R_d(t)) \end{pmatrix} \tag{13}$$

where x is the cartesian position of the frame origin, and $O(R(q), R_d(t))$ is any 3-dimensional parametrization of the attitude error between the actual and desired frame.

A.3 Redundant Tasks

These task functions take advantage of a possible underdetermination of the specified task with respect to the number, n of joint coordinates. Their form is:

$$e' = W^{\dagger} e_1 + (I_n - W^{\dagger} W) g \tag{14}$$

where e_1 is a m-dimensional primary (main) task vector, $m < n$, W is a full rank $m \times n$ matrix such that its null space is equal to the one of $\frac{\partial e_1}{\partial x}$, x representing an adequate working space, and $g = \frac{\partial h}{\partial x}$ is the gradient of a secondary task to be minimized under the constraint $e_1 = 0$. Many kinds of secondary tasks may be considered, for example tending to remove the robot from joint limits, kinematical singularities, obstacles...

A.4 Sensor-based Tasks

They constitute a particular case of redundant tasks. The working space is $^3 \times SO_3$, with generic element $\bar{r}$. The goal is to set a sensor output s to a value $s_d(t)$, with dim $(s) = p$. We then have, in eq (14):

$$e_1 = D(s - s_d) \tag{15}$$

Where the $m \times p$ matrix D, $m \leq max(6, p)$ is shown ([36]) to be such that $D = W \frac{\partial s}{\partial \bar{r}}$. Usually, h expresses a trajectory tracking in the working space, and the task is then called *hybrid task*.
This approach covers the use of force ([21]), proximity ([36]) as well as vision ([20])sensors.

B Appendix: Glossary of abbreviations

HMI: Human-Machine Interface
F-object: Functional object
MT: Module-Task
ORCCAD: Open Robot Controller Computer Aided Design
PID: Proportional-Integral-Derivative

RT: Robot-Task
RTA: Robot-Task Automaton
SIMPARC: Simulator for Multi-Processors Advanced Robot Controllers
TCS: Time-Constrained Specification