Advanced Manufacturing and Automation
amarc Research Centre
UNIVERSITY OF BRISTOL

PROCEEDINGS

EURISCON '94

European Robotics and Intelligent Systems Conference

Volume 2 (Stream B)

Malaga, Spain August 22 - 25 1994

A METHODOLOGY FOR THE DEVELOPMENT OF AN OBJECT-ORIENTED CONTROL ARCHITECTURE

Nicolas Delpech, Yann Plihon, Michel Llibre
CERT/DERA
2, avenue Edouard Belin.
31400 Toulouse, France
Phone: (33) 62 25 25 25
Fax: (33) 62 25 25 64
e-mail: delpech@saturne.cert.fr
plihon@saturne.cert.fr
llibre@saturne.cert.fr

ABSTRACT

A control system is often divided into two levels: the planning level and the executive level. We propose to define a generic software architecture of the executive level to make it the most independent possible of the application. We lean on a multi-agent real-time communication system (CESAM) defined at CERT-DERA. The selected agent model is the one of an executive agent considered as an extension of the usual object model. The definition of this architecture will be divided into two levels:

- a design level based on the usual objet-oriented paradigm which provides such mechanisms as inheritance and polymorphism allowing to satisfy the goals of software extensibility, reusability and portability.
- a distribution level which precises the type of activity and access to the methods of the objects achieved at the design level, according to the agent model defined by CESAM.

The application's programmer disposes of a programming method: his work is concentrated on the design of the application from the basic classes defined by our architecture, the distribution (communication, synchronization) of the new agent being made transparent by these same basic classes.

Once this environment of robotic application is defined, we describe how hybrid control have been developed on a parallel robot used as master device for an experimental teleoperation set-up from this architecture.

1. INTRODUCTION

A control system may be defined as a set of software and hardware entities handling real-time data coming from:

- the physical process it controls through the hardware,
- the human operator or a planning level which uses the control system to act on the process itself.

The control system has to ensure the data flow (data acquisition, sending the commands,...) with the process according to its dynamics ranging from a few microseconds to some milliseconds.

In our case, these physical systems are mainly robots and all devices going with them: sensors (camera, force sensors, encoders), effectors,...etc.

If the separation between the physical process and the control system is relatively easy to do, on the contrary, it is not so easy to do it between the planning system and the control system.

The software layers interface with the process to control is achieved through the hardware (input/output boards) which is more and more efficient and easy to use.

On the contrary, the planning system often considered as a decision level, deals with larger scopes with the increasing complexity of application. We usually consider that the decision level handles mainly symbolic data which are more adapted to define complex tasks and mission either automatically or by an operator. The interface between the decision level and the control level may be defined as the bidirectionnal translation from symbolic data into action-oriented data: it is the command interpreting level.

Through large researches in the field of Artificial Intelligence and specially in the field of Distributed Artificial Intelligence [1], the multi agents-oriented concept deals with the field of modelling and definition of decision and planning architectures. It allows to design the process behaviour and to organize the diverse decision layers involving action and perception so that an adequate reactivity at the lower level and an intelligent control at the higher level is kept.

By using this approach in order to structure a control system, we can achieve a unitary viewpoint between the structure of decision and control levels. At the same time, we mus satisfy the real-time constraints required by this last one which is the matter of our work.

As part of CERT-DERA research works [3][4], an agent-oriented model is proposed from which real-time applications can be developed. This executive agent-oriented model is defined as an extension of the standard object model which allows us to use the object-oriented paradigm.

We propose then an application programming methodology from robotic devices designed as standard objects or agents. The application programmer will structure his application from these objects.

At the end, we will present an hybrid control architecture for a robot used as a master device on a teleoperation set-up.

2. OBJECT-ORIENTED REAL-TIME PROGRAMMING

2.1. Introduction

The object-oriented paradigm [5][2] is based on an unifying idea including: the object, a data structure, and the methods to use them.

Unlike functional approaches which consider a program as a set of routines and separated data these routines act on, the object-oriented approach considers a program as a set of independent objects communicating thanks to messages.

If the object-oriented paradigm improves software structuring, it is not well-suited to develop real-time softwares. An extended object model [6][7][8] will allow us to develop such softwares.

2.2. Object-oriented language

The object-oriented paradigm deals with two main viewpoints:

- the **structural** viewpoint: an object is seen as a data structure joined to a class-named object. A class is the abstract data model of a set of entities sharing the same structure and having the same behaviour.
- the **activity-oriented** viewpoint: an object is seen as a little software entity both active and autonomous which interacts with other entities through messages. In this case, we focus on the behaviour of the object rather than its structure. So, we achieve an actor-oriented model [9][10].

2.2.1. Structural viewpoint: class-oriented language

The class-oriented language used in our work is the C++ language [11]. In this case, a class is composed of a static part which corresponds with its status and the connections with other objects, and a dynamic part which describes its behaviour, in other words, its whole functions and its event reacting way.

Three mechanisms characterized this class-oriented language: instantiation, inheritance and polymorphism.

* instantiation: a class is the conceptual entity which describes the object (encapsulation property). Its definition is used as a model to design its physical representatives as class instance.

* inheritance: it is possible by inheritance to define other more specific classes which precise the parent class scope. Two inheritance types exist:

- simple inheritance; a subclass possesses only a direct parent class,
- multiple inheritance; a subclass can possess several direct parent classes.

Once the whole classes are defined, a tree of classes is achieved. The inheritance notion induces a programming method which proceeds by successive derivations from the generic classes.

*polymorphism: it represents literally an entity having several forms. We can distinguish three polymorphism types:

- inheritance polymorphism; a subclass instance can be used as a parent class instantiated object.
- function polymorphism; a same function can own arguments being able to have several types.
- ad hoc named polymorphism; methods belonging to different classes can have the same selector.

However, this model is unsuited to resolve the problems of parallelism, synchronization and distribution of real-time control systems.

2.2.2. Active-oriented viewpoint: actor-oriented language

Usually, the actor-oriented model adds to the encapsulation and abstraction mechanisms, the notion of parallel activities and asynchronous communication. An actor is then an autonomous and independent activity which communicates through asynchronous messages. It is composed of:

- acquaintances which are the other actors that it directly knows,
- a script which describes its behaviour when it receives a message.

The proposed actor-oriented models usually involve a too much restrictive homogeneity to develop control systems; besides, the class-oriented language structural viewpoint is here away.

2.3. The extended object-oriented model.

In our CERT-DERA research work, we have proposed a non-uniformed object-oriented model [3,4] made up by three object types:

- the **standard objects** coming from the standard model. They are local, static and synchronous,
- the **exported objects** with remote access,
- the **agents** which are asynchronous, distributed and active objects.

* exported object

We add a remote interface to the two components of the standard object-oriented model: private status and local interface. The local interface consists in the methods usually named "public" and the remote interface in methods which are named "exported" and

remotely called (but also locally). This remote interface extends the object invocation mechanism to a distributed environment.

* agent [1]

The agent-oriented functional model (figure 1) provides four capacity ways:

- a specialization capacity representing its abilities.
- a self-control capacity representing its autonomy.
- an assignment capacity; its specialization does not allow it to deal with the whole ability scopes. So, it assigns some tasks to its acquaintances which are the agents whose abilities are needed to achieve its goals.
- a communication capacity which allows it to integrate a group (figure 2).

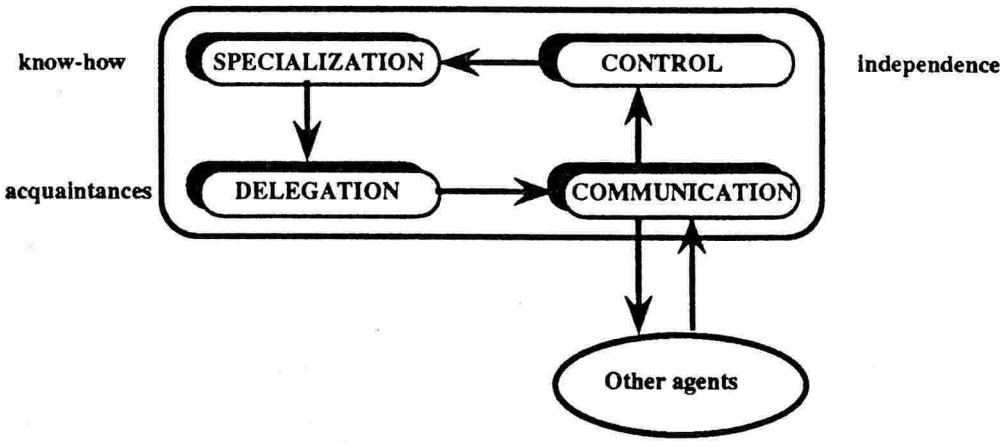

figure 1: executive agent functional model

The agent entity corresponds to the parallelism unity and is made of different objects. Among these objects, the exported objects which correspond to the distribution unity, allow the remote interface of the agent.

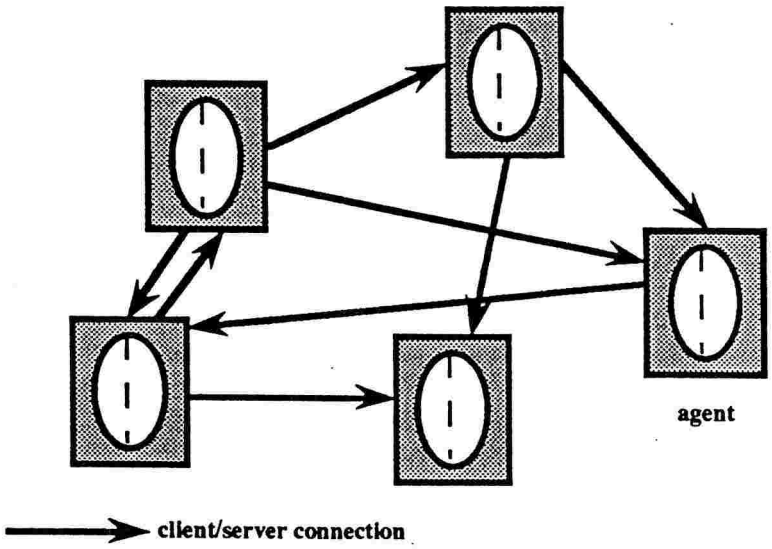

figure 2: relationship-oriented architecture

An agent takes part in activities during which its objects can communicate with the ones of other agents. This communication carries out according to two ways:

- message-passing,
- remote call.

These two ways are possible thanks to a communication system designed as a minimal kernel being able to equip any machine, whether real-time or not. It realizes the interface between the application on the one hand, and the operating system (whether real-time or not), the machine and the network on the other hand. The rules of this system are based mainly on two hierarchical layers, each one providing a certain abstraction level:

- shared object-based communication: it allows asynchronous message-passing between parallel agents on homogeneous machines.
- image object-based communication: an asynchronous communication protocol is provided by remote call of active objects on homogeneous or heterogeneous machines independently of the implementation locality. It is an asynchronous and unblocking client/server protocol.

2.4. CESAM programming environment [3]

2.4.1. C++a language

We have now a model-based programming language (figure 3) from which we can define objects with different abstraction levels: from the standard object to the executive agent with parallel activities, asynchronous remote communication and reactive behaviours. This language is defined as a C++ language's extension to the multi agents-oriented C++a language.

The communication between these objects is based on the shared object-based and image object-based communications whose principles will not be described in this paper [3].

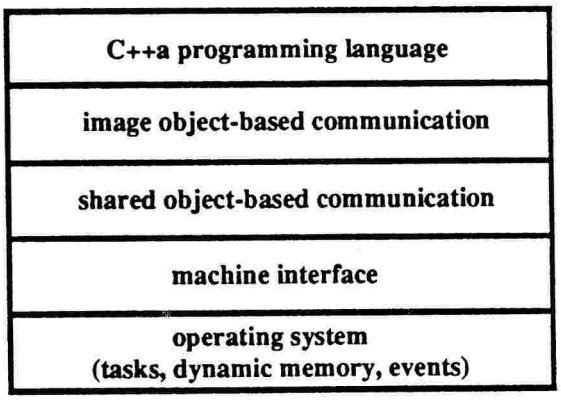

figure 3: communication system architecture

2.4.2. Implementation

At present, the previously defined notions are partly implemented. The achieved experimental version only allows the communication between agents belonging to a same computer whether multiprocessors or not. It also allows remote clients implemented on host machine (SUN workstation in our case) to request services of agents running on a real-time computer.

This experimental version has been developed on multiple VME-based 68000 family processors on a backplane network running the VxWorks operating system [12].

The extension of C++ language to a C++a multi agents-oriented language requires the development of a precompiler in realization. For the moment, macro instructions have been developed to make the application programming easier.

2.4.3. System agents

The CESAM environment software architecture (figure 4) is composed of four system agents:

- a **catalogue** resolving the remote access problems by memorising the services published by agents of its environment (in other words, of its running machine) in order to pass them to their clients.
- a **supervisor** agent to pass messages sent by clients of other computers, to send them to the assigned agents (servers) and to return reply messages.
- a **synchronizer** agent to control the activation and desactivation of agent's exported methods according to a determined script. Three synchronization ways are available: sequential, parallel and coordinated. More complex synchronization scripts can be achieved thanks to the synchronizer's recursive property.
- a **performer** of application programs.

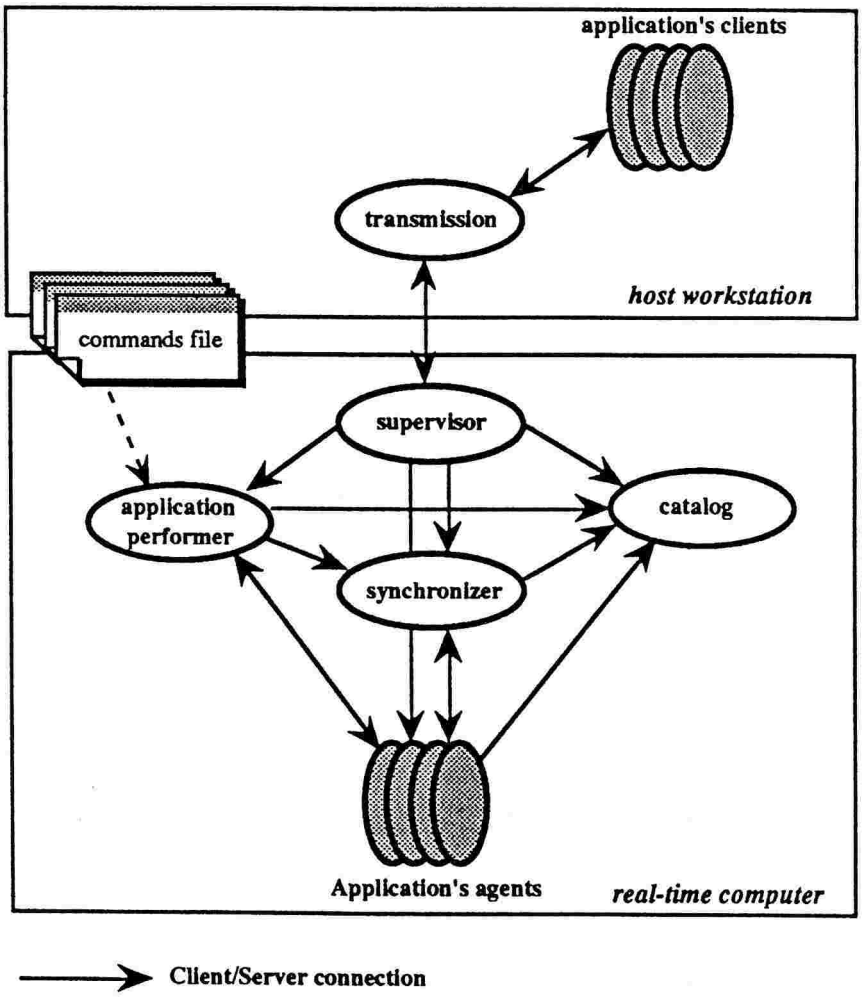

figure 4: CESAM software architecture

3. METHODOLOGY TO STRUCTURE ROBOT CONTROL SOFTWARE

3.1. Requirements for an application programmer

Usually, the robot control software development requires two competence levels:

- a **robotic programming level**: it focuses on diverse control laws like joint and cartesian servoloops, guiding, trajectory generator,...etc.
- a **real-time programming level**: it focuses on communication, synchronization between the diverse control system actors as well as their distribution. In the case of robotic application, we must face with the hardware control and the multiprocessors distribution.

The standard control programming according to a functional viewpoint induces couplings between these two levels. Therefore, it makes the robotician work more complex; indeed, the time spent to adjust the application real-time problems is usually more important than the time spent to adjust the control laws.

Besides, these softwares are confronted with maintenance problems. The permanent development of equipments leads to the possibility to replace some devices by identical ones, or more efficient ones, or to add new ones during the application. The control system must allow modifications without involving important modifications in the global architecture [13].

3.2. Why to use CESAM?

The real-time programming concepts introduced in the previous paragraph, must resolve several development constraints. As the standard object-oriented paradigm, this programming way is more suited for the following reasons:

- it's natural to model a system according to its physical reality as a set of objects (standard objects, exported objects, agents) in order to carry out a particular task.
- modular and generic softwares as well as their extensibility are satisfied.

Besides,

- a certain modelling homogeneity can be respected between the different control levels as well in the modelling level of the various control actors as in the communication level between these actors.
- it improves software portability and reusability. The properties of CESAM communication system enables to develop control softwares independently of the hardware and computers.

3.3. Programming methodology

An architecture and a programming methodology are proposed here to reduce coupling between the robotic programming level viewpoint and the real-time programming viewpoint [14]. Therefore, it is why the control software's structure is divided into two levels:

- **modelling** level,
- **distribution** level.

3.3.1. Modelling level

3.3.1.1. Definition

This level describes the application as a set of classes according to two object models: standard objects (standard object-oriented model with local interface) and exported objects (extended object-oriented model with local and remote interface). We do not precise the activity mode of these objects, in others words, in which real-time background they run.

The control level modelling as objects creates two class types:

- classes of physical objects which result from the modelling of the physical devices to control (input/output driver, effector, encoder, sensor, manipulator,...),
- classes of abstract objects which represent the various entities modelling the control laws developed in a control software (joint and cartesian servo, guiding, trajectory generator,...).

The first object type defines a convenient hardware's interface by means of objects seen as an image of this hardware. This image will be independent of the physical nature of the devices which are modelled. A control software independent of hardware (computer, electronic) could be developed making its evolution easier.

The second object type (abstract objects) defines robotic devices library modelling the law of the various control levels.

3.3.1.2. Principle

The object-oriented modelling of control system usually appears as just one tree designed from one or several generic classes (for instance, class robot) in order to achieve the definition of specific application objects thanks to successive derivations. The definition of one tree per application seems too much restrictive to realize a generic structure used for different applications. We prefer designing a tree per control level. So, a set of dedicated trees was achieved (figure 5). If there are structural ties between levels (trees), they are made by aggregation of generic classes. Each tree is based on some generic classes. Their specification is realized by simple or multiple inheritance while keeping the same interface.

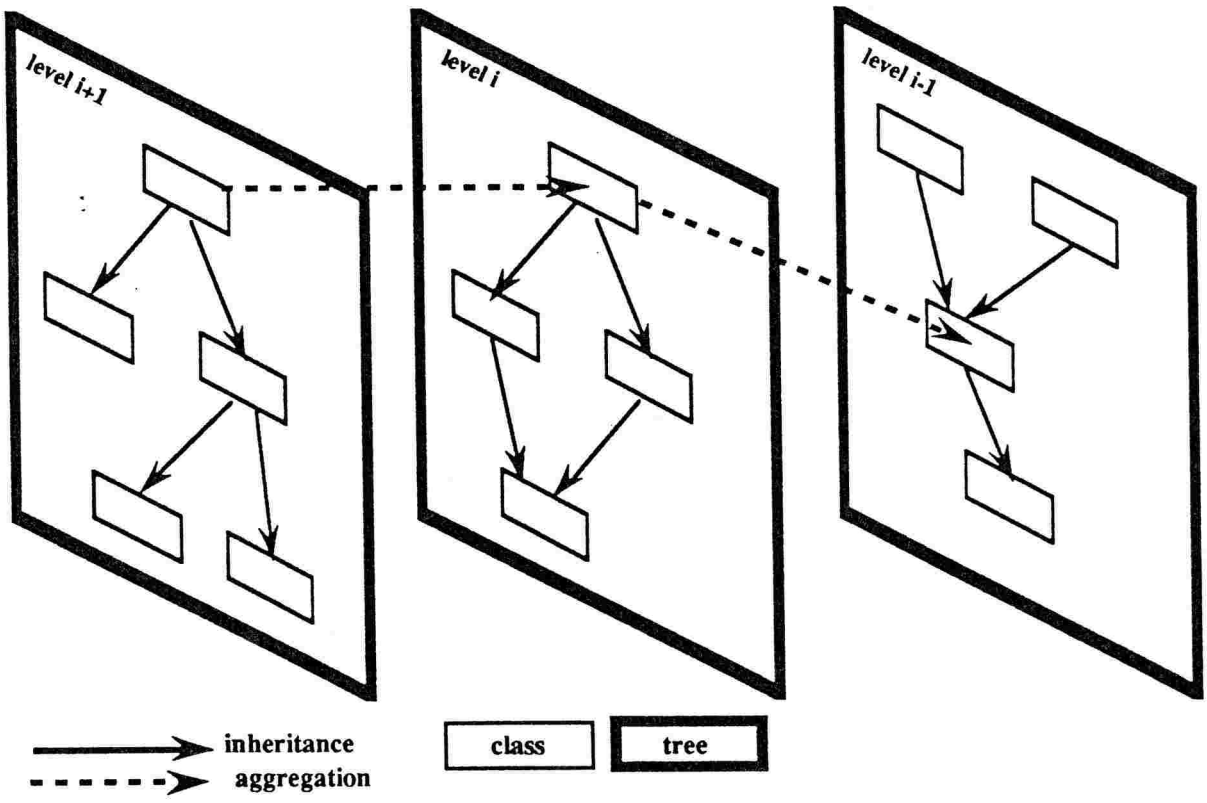

figure 5: basic structural scheme

Note: it is essential that the generic classes of each tree should be as general as possible in order that the available structure should be the less restrictive and as open as possible .

3.3.2. Distribution level

This level is based on:

- the definition of generic agents modelling the main activity ways which are usually required to the control structure: scheduled activity, data servers, client/server communication protocol,...),
- the distribution of objects, resulting from the modelling level, into executive agents which determine their activity way.

3.3.2.1. Generic agent

A generic agent designs a typical activity way of control systems. It encapsulates control structure and communications required to this activity, from the two communication levels provided by CESAM: shared object-based and image object-based communications.

At this point, the agent specialization abilities are not precised. The generic agents can be divided into two sets:

- the ones with a client/server activity. The agent behaves like a server waiting for requests to carry out. If there is no request, it is suspended. It concerns an asynchronous activity.
- the ones with server + local activity. The local activity is synchronous (in the robotic case, it concerns mainly a periodic-scheduled processes). Sequentially to this activity, the agents carry out the received requests (it only consults a request receiving without waiting for them).

3.3.2.2. Agent specialization

The generic agents are not directly feasible. In a second time, their ability field must be specified. So, thanks to mechanisms such as inheritance, polymorphism and aggregation, specific objects provided by modelling are joined to them. These objects can be either local or exported. We also specify the agent acquaintances which describe the ability field delegated to other agents.

This distribution into executive agents can be realized according to two contrary approaches:

- a distributed approach which consists in joining an independent executive agent to each object.
- a centralized approach which consists in joining as much objects as possible into an executive agent to limit the number of activities for an application.

We do not want to force a particular approach but to allow the programmer to choose an abstraction level specific to his application.

3.3.3. Conclusion

Control software development should lead to a direct application modelling without taking into account the distribution. The objects so achieved are then joined into predefined executive agents which interface the diverse activities required for such softwares. The programmer needs neither great abilities to real-time programming, nor ones to CESAM programming. Predefined objects and agents used by the programmer must be clearly specified. Their right running must be guaranted because the programmer does not know their software in most of cases. Nevertheless, the suggested structure must be open enough in order that the programmer could develop himself specific agents which are not provided. He will be able consequently to manipulate CESAM concepts.

4. A ROBOT HYBRID CONTROL ARCHITECTURE

Instead of giving an exhaustive list of the objects defined for the control systems, we are going to describe a specific control architecture: an hybrid position/force control. However, we will insist on the parting between the generic level of a robot hybrid control and the application level based on our experimental set-up.

4.1. Generic level

4.1.1. Hybrid position/force control definition

The position/force hybrid control approach [15] consists in dividing the controlled workspace of the robot in two complementary and orthogonal subspaces. In one of them, the robot is position controlled and in the other subspace it is force controlled. The choice of this space partition depends on the local interaction of the robot with its environment. The unconstrained directions (with no obstacle) are position controlled and the constrained directions (constrained by the environment) are force controlled. The space partition is described by S, a 6*6 diagonal matrix. The diagonal elements are either 0 (in that case, the corresponding axis is force controlled) or 1 (in that case, the corresponding axis is position controlled). The behavior of hybrid controlled robots depends on the situation of the robot's frame, the space partition S and a set of positions desired Xd and forces desired Fd. As the position loops and the force loops are independent, only the components of Xd (resp. Fd) corresponding to the position (resp. the force) controlled subspace are taken into account by the controller. We give a simplified scheme (figure 6) for hybrid position/force control.

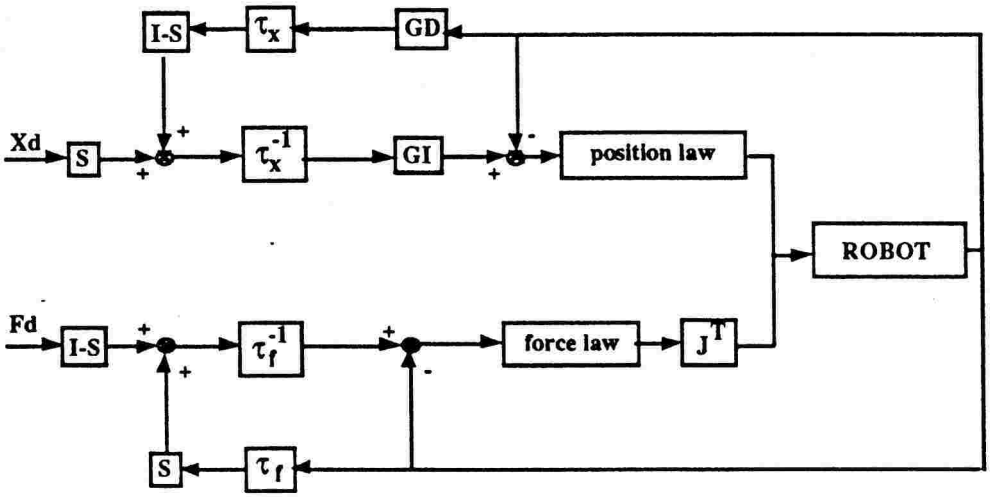

figure 6: a simplified hybrid control scheme

4.1.2. Software structuring

From a structural viewpoint [16], the robot hybrid control software is naturally organized according to three hierarchical layers:

- acquisition layer,
- joint servo layer,
- hybrid cartesian servo layer.

For each layer, a modelling phase and a distribution phase will be found.

4.1.2.1. Acquisition level

input/output modelling

A control software running on real-time computer uses input/output devices to have access to various sensors and effectors of the robot to control. It's easy to model these devices into generic objects in order to specify them according to the driver technical features.

Two main components for an input/output board are considered:

- a board component which encapsulates the physical board's features,
- a channel component which describes the board's interface with its users.

Two generic channel classes are then defined: an *input* class and an *output* class as well as their interface. On the contrary, it is not possible to define a generic board class because the physical features are very different according to the hardware used. It is only created at the application level.

This genericity facilitates an input/output board substitution easier without modifying its users.

distribution

Input data (measure) have timing constraints. In the case of the hybrid control, it concerns in particular, the integrity of the measure/date pair ensuring the right performance of servoloops which run periodically. So, we define an acquisition agent as following:

- it ensures the measure/date integrity (periodical acquisition of encoders measures),
- it manages the input/output shared device.

Indeed, two activities can not simultaneously have access to a board because a definitive error of the backplane network occurs generally. Consequently, a data acquisition agent is joined to each physical device. One agent can be joined to several boards but not several agents to the same board.

Several parallel activities could be connected to an acquisition agent to be synchronized with sensor measures. They could to wait for measures provided by a same physical device without causing access problems. Moreover, the connection to this acquisition agent is independent of the multi-CPU architecture.

Note: in the case of the acquisition agent providing force/position measures through analogical boards, its activity must be run under IT timer. This agent may be considered as a generic interface of timer device, designing the data periodical scheduled acquisition protocol.

Other acquisition agents could be defined without connecting their activity to an IT timer (ultra-sound server, image acquisition server,...).

To make the programming of the activities scheduled on measure data easier, a generic agent designing the connection to an acquisition agent is defined. It is a *SCHEDULED_MODULE* agent class. Each agent with a scheduled activity (from measure or not) inherits from the *SCHEDULED_MODULE* class.

The *SCHEDULED_MODULE* class has these arguments:

- a scheduling period,
- an list of analogical inputs (this list can be empty),
- an agent activity's priority.

4.1.2.2. Servo level

joint controller modelling(figure 7)

A tree modelling the various joint controllers required to the hybrid position/force control is determined from a *jointservo* parent class whose two specific classes inherit, modelling a position joint servo and a force one: *P_jointservo* and *F_jointservo* classes.

An *effector* generic parent class encapsulating effectors features and an *input* channel object are aggregated with the *jointservo* parent class.

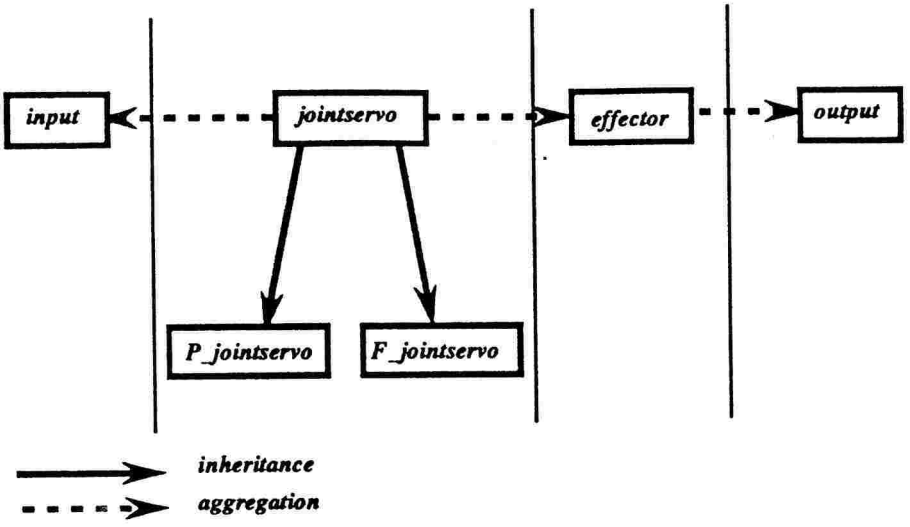

figure 7: joint servo level modelling

Distribution

The servo objects distribution into executive agents is realized according to different abstraction viewpoints, from the most distributed to the most centralized viewpoint:

- one executive agent per joint servo (ie 2n agents for a n axes robot),
- one agent per axis (ie one agent per position/force joint servo pair: n agents),
- one agent per joint servo type (ie one agent for the whole position joint servos and one agent for the whole force joint servo: 2 agents),
- one agent joining the joint servos.

The two first viewpoints are not the best solution for our application because we often try to reduce computing charge specially for an architecture with a limited number of processors. As regards the choice between the two last viewpoints, it is done in accordance with the application (see the application level chapter).

The two agents viewpoint allows to distribute them on two distinct CPU. So, we can reach lower scheduling periods. But this viewpoint leads to the problem of command fusion provided by position/force joint servos.

The one agent viewpoint facilitates the interface with higher level; it requires fewer processors. The scheduling periods are more limited.

This two last viewpoints (figure 8) are also more suited to an eventual kinematic coupling between the various axes.

In these two cases, the joint servo objects are local objects; the joint servo agents are exported objects with exported methods as interface: position/force command setting, position/force reading,...etc. These agents inherit from the *SCHEDULED_MODULE* agent class.

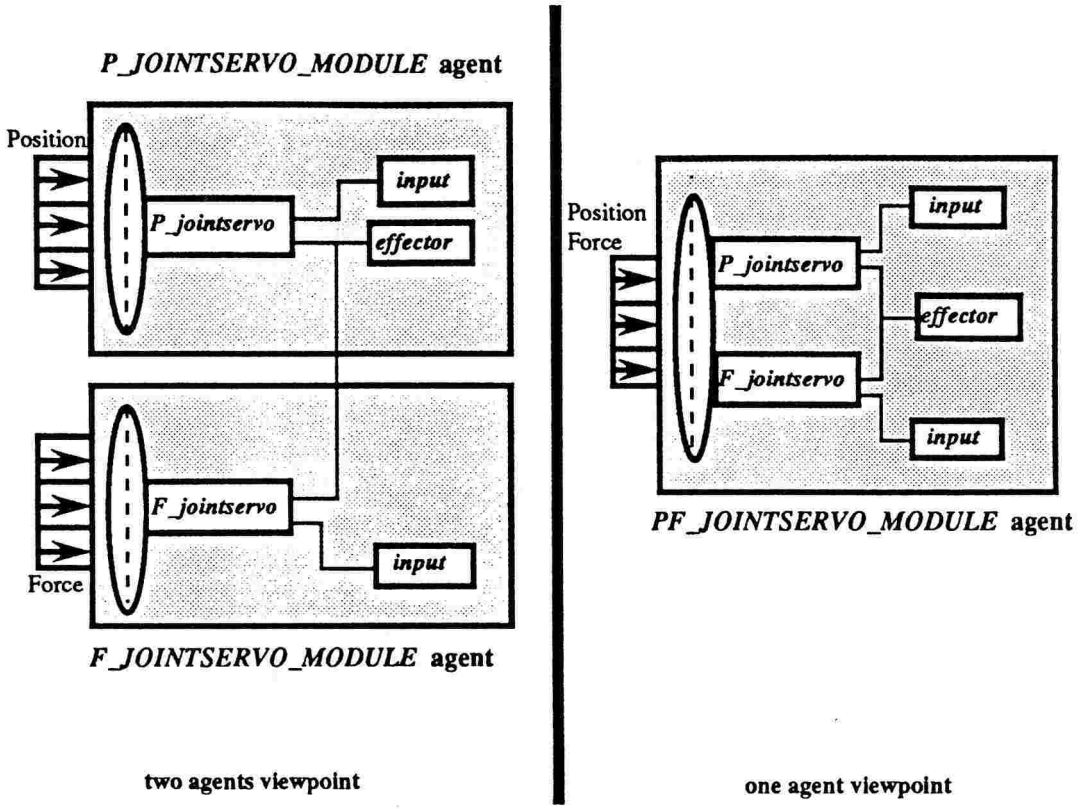

figure 8: joint servo agents

4.1.3. Hybrid cartesian level

modelling

Two classes are defined:

- an *hybrid_direction* class designing the hybrid servo control of a cartesian direction,
- an *hybrid_controller* class designing a 6 axes hybrid control scheme and its functions: interface methods (exported methods), definition of specific hybrid control frames, geometric transformations between the frames. The methods designing the robot mechanical structure (kinematic models, Jacobians,...) are specified at the application level.

An *hybrid_direction* object is a local object. An *hybrid_controller* object is an exported object aggregated with an *hybrid_direction* object list at the construction.

Distribution

An *HYBRID_ROBOT* executive agent class is defined from an *hybrid_controller* exported object. This agent, as the joint servo module, inherits from the *SCHEDULED_MODULE* agent class: it consists in a local scheduled activity (hybrid controller) joined to its interface (exported methods). These acquaintances are joint servo agents to which it sends the position/force joint commands.

4.2. Application level

4.2.1. Robot presentation: a master robot [18]

This robot is a fully parallel device having six degrees of freedom. The moving part, equipped with a handle that can be gripped by the operator, is connected to the fixed part

through six pneumatic linear actuators. Hybrid control requires both position and force control, and pneumatic control requires a high sampling rate: this control design is real time costly. The real-time operating system, VxWorks, is supported by two processors running in parallel (one 68040 MVME167 and one 68030 MVME147) and mounted on a VME rack. The reading of the position and force sensors as well as the command writing for the effectors are made by means of a CELOCIC board .

This robot is used as a master device in a teleoperation experimental set-up which is also composed of another hybrid controlled robot with six degrees of freedom: the slave robot.

4.2.2. Software architecture

4.2.2.1. Joint servos

The input/output classes must be specified in accordance with the used boards. So, a board class is created to encapsulate the CELOGIC input/output board's features: a *CELOGIC_board* class. In the same way, two specific channel classes are defined: *CELOGIC_input* class and *CELOGIC_output* class (figure 9).

figure 9: input/output modelling

Regarding effectors, a specific class is defined for the used pneumatic effectors which are carried out by two complementary servovalves (ie two analogical outputs): *pneumatic_effector* class.

Regarding the joint servo controller, an only executive agent viewpoint is chosen as a *PF_JOINTSERVO_MODULE* object implemented on a 68040 CPU with a 3ms scheduling period (figures 11,12).

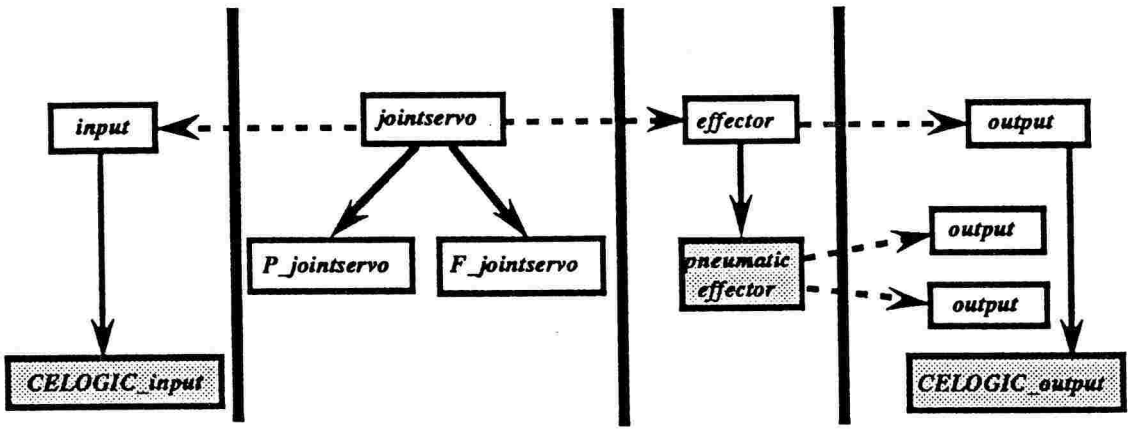

figure 10: master robot joint level modelling

4.2.2.2. Cartesian servocontrol

We use here the *HYBRID_ROBOT* executive agent by specifying the robot kinematic models; a *MASTER_ROBOT* class is defined. This agent is implemented on the second CPU (68030) with a 9ms scheduling period (figures 11,12).

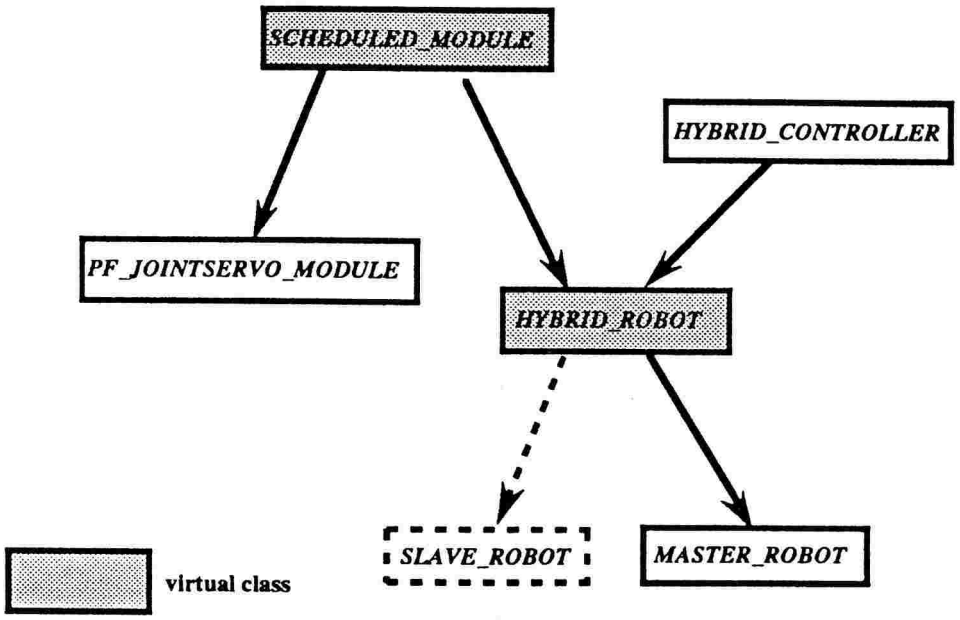

figure 11: robot hybrid control structuring

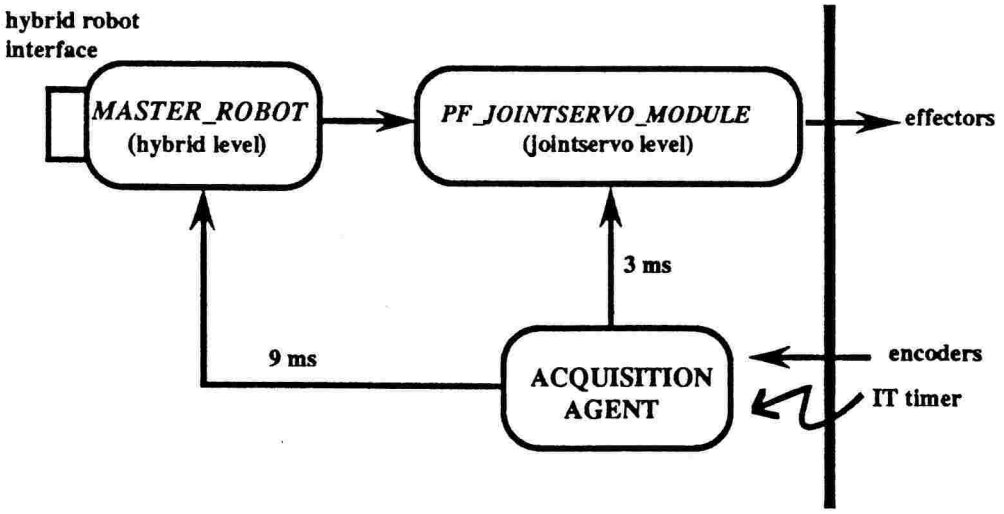

figure 12: functional architecture

5. CONCLUSION

As part of the hybrid teleoperation set-up, in the same way we have defined a *MASTER_ROBOT* class from the *HYBRID_ROBOT* class, we can define a specific class for the slave robot of the experimental set-up, also with an hybrid control law. Afterwards, we could design a generic teleoperation kernel independent of the used master and slave robots. The kernel software will only handle virtual *HYBRID_ROBOT* agents.

More generally, we have proposed a control system structuring and programming methodology in order to facilitate the robotician's work, rather than a definitive architecture that could not be completely generic. It must allow to lead his work to an application modelling from libraries of generic objects (standard and exported objects), whereas the whole real-time functions are previously modelled by executive agents.

To make this modelling work easier, it would be interesting, from the diverse generic objects, to propose a graphic software [17] allowing to design trees to model the application.

This architecture type could also facilitates the simulation of the control software as well as the definition of a graphic interface of the diverse objects in order to visualize their status during their execution.

REFERENCES

[1] J. Ferber, "Objets et agents: une étude des structures de représentation et de communication en Intelligence Artificielle", Thèse d'Etat-Université Paris VI, Juin 1989.

[2] J. Ferber, "Conception et Programmation par Objets", Hermes,1990.

[3] F. Mefthou, "Système de commande temps-réel multi-agents", Thèse d'Etat ENSAE, Décembre 1993.

[4] F. Mefthou and P. Carton, "Modèle d'un agent exécutif temps-réel", Premières journées francophones IAD-SMA, Toulouse, France, April 1993.

[5] G. Booch, "Object-Oriented Design with Applications", Benjamin Commings, 1986.

[6] P. Gautron, J.P. Briot, H. Saleh, S. Lemarie and L. Lescaudron, "Development of an environment for specification and execution of active objects on parallel machines", European Workshop on Parallel Computing (EWPC'92), Barcelona, Spain, March 1992.

[7] A. Yonezawa, J.P. Briot, E. Shibayama, "Object-oriented concurrent programming in ABCL/1", Proceeding of the 1st OOPSLA, pp 258-268, Portland, Oregon, 1986.

[8] M. Fabrian and B. Lennartson, "Object-oriented structuring of real-time control systems", Control Engineering Laboratory, Chalmers University of Technology, Gothenburg, Sweden, 1992.

[9] C.E. Hewit, P. Bishop and R. Steiger, "A universal modular ACTOR formalism for Artificial Intelligence", Proceeding of the 3rd IJCAI, pp 235-245, Stanford, California, 1973.

[10] G. Agha, "Actors: a model of concurrent computation in distributed systems", MIT Press, Cambridge, Masschusetts, 1986.

[11] B. Stroustrup, "Langage C++", InterEditions, Paris, 1989.

[12] Wind River Systems, "VxWorks Programmer's guide".

[13] D.J. Miller and R.C. Lennox, "An object-oriented environment for robot systems architecture", IEEE, Control Systems, Vol. 11, Number 2, pp 14-23, February 1991.

[14] E. Coste-Maniere, B. Espiau and D. Simon, "Reactive objects in task level open controller", Proceeding of the 1992 IEEE, International Conference on Robotics and Automation, pp 2732-2737, Nice, France, May 1992.

[15] M.H. Raiberg, J.J. Craig, "Hybrid position/force control of manipulators", ASME Journal of Dynamics Systems, Measurement and Control, pp 262-268, June, 1981.

[16] R. Lumia, J. Fiala and A. Warering, "The NASREM robot control system and testbed", IEEE Journal of Robotics and Automation, Vol. 5 (N°1), pp 20-26, 1990.
[17] D. Simon, B.Espiau, E. Castillo, K. Kapellos, "Computer-aided design of a generic robot controller handling reactivity and real-time control issues", Rapport de recherche INRIA N°1801, Nov 1992.
[18] Y. Briere, Y. Plihon, C. Reboulet, "The dual hybrid position-force concept for teleoperation", Euriscon 94, Malaga, Spain, 21-26 August, 1994.