MULTIAGENT-BASED CONTROL SYSTEMS: AN HYBRID APPROACH TO DISTRIBUTED PROCESS CONTROL[†]

Juan R. Velasco, José C. González, Carlos A. Iglesias and Luis Magdalena

ETSI Telecomunicación, Universidad Politécnica de Madrid
Ciudad Universitaria s/n., E-28040 Madrid, Spain.
email: jvelasco | jcg | cif@dit.upm.es -- llayos@mat.upm.es

Abstract: In this paper a general architecture and a platform developed to implement distributed applications, as a set of cooperating intelligent agents, is presented. Second, it will be shown how this architecture has been used to implement a distributed control system for a complex process: the economic control of a fossil power plant.

Agents in this application encapsulate different distributed hardware/software entities: neural and fuzzy controllers, data acquisition system, presentation manager, etc. These agents are defined in ADL, a high level specification language, and interchange data/knowledge through service requests using a common knowledge representation language.

Keywords: Agents, Distributed control, Fuzzy expert systems, Machine learning, Power generation

1. INTRODUCTION

This paper presents a way to undertake the distributed control problem from a multiagent systems point of view. To summarize, agents are autonomous entities capable of carrying out specific tasks by themselves or through cooperation with other agents. Multiagent systems offer a decentralized model of control, use the mechanisms of message-passing for communication purposes and are usually implemented from an object-oriented perspective.

[†]This research is funded in part by the Commission of the European Communities under the ESPRIT Basic Research Project *MIX: Modular Integration of Connectionist and Symbolic Processing in Knowledge Based Systems*, ESPRIT-9119, and by CDTI, Spanish Agency for Research and Development *CORAGE: Control mediante Razonamiento Aproximado y Algoritmos Genéticos*, PASO-PC095.

The MIX consortium is formed by the following institutions and companies: Institute National de Recherche en Informatique et en Automatique (INRIA--Lorraine/CRIN--CNRS, France), Centre Universitaire d'Informatique (Université de Genéve, Switzerland), Institute d'Informatique et de Mathématiques Appliquées de Grenoble (France), Kratzer Automatisierung (Germany), Fakultät für Informatik (Technische Universität München, Germany) and Dept. Ingeniería de Sistemas Telemáticos (Univérsidad Politécnica de Madrid, Spain).

The CORAGE Consortium is formed by UITESA, Dept. Ingeniería de Sistemas Telemáticos (Universidad Politécnica de Madrid), IBERDROLA and Grupo APEX.

The multiagent architecture that has been developed can be used to implement any kind of distributed application, not only distributed control system. In this general framework, several software elements (the agents) cooperate to reach their own goals. System designer has to decide the set of agents that will be involved in the task, specifying their particular capabilities. At a high level, this part of the design work is carried out by describing the agents in ADL (Agent Description Language) (see below and González, J.C et al. (1.995)). The problem of how to intercommunicate data between agents is solved by using a common knowledge representation language.

As an example of how to apply this architecture for distributed control, a real system is going to be shown: a fossil power plant. In particular, the goal is to achieve strategic (not tactic) control: the system has to reduce the heat rate (the ratio *combustible/generated power*), suggesting appropriate set points for automatic controllers or human operators.

At this moment, two versions (distributed and non-distributed) of a control system for a real power plant sited in Palencia (Spain) (García, J.A. et al , 1.993) are being implemented. This paper is focussed mainly on the distributed one.

2. AGENTS DESCRIPTION

The proposed architecture has been designed according to the following lines:

- Use of mechanisms of encapsulation, isolation and local control: each agent is an autonomous, independent entity.
- No assumptions are made regarding agents' knowledge or their problem-solving methods.
- Flexible and dynamic organization is allowed.

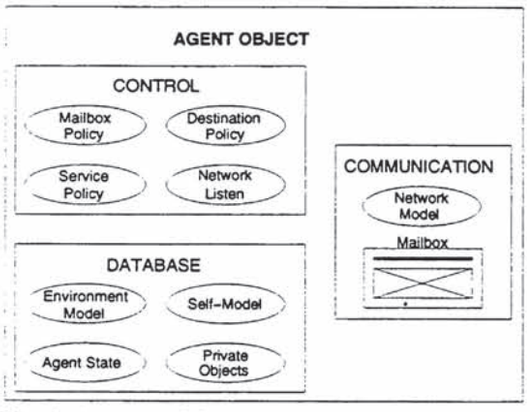

Fig. 1. Agent model

Every agent is composed of a control block, a database (including an agent model, an environment model, the agent state, some private objects and global data), and a communications block (the network communications model and a mailbox).

Any agent may include some *agent goals* (processes which start when the agent is born), *offered services* (the agent offers to the rest of agents a set of services, and these services may be executed in concurrent --as an independent process--, or non-concurrent mode), *offered primitives* (a set of internal services which may modify some of the agent's private objects) and *required services* (a list with the names of the services that this agent requires).

One of the mayor features of these agents is that their services (if concurrent) are executed as separated processes, so the agent control loop can continue its job. In this way, the same (concurrent) service can be executed several times, each one called from a different agent.

3. MIX MULTIAGENT PLATFORM

At the network level, coordination among agents is carried out through specialized agents (called "yellow pages" or YP). Whenever an agent is launched, it registers first to YP, informing about its net address, its offered services and the services it will request from other agents. In the same way, agents can subscribe to "groups". Groups refer to dynamic sets of agents, and can be used as aliases in service petitions. So, service petitions can be addressed to particular agents, to every agent in a group or to all the agents offering a service.

YP agents update continuously the information needed by their registered agents. Therefore, these are able to establish direct links among them, so avoiding collapse due to YP saturation or (some) network failures.

Regarding agent communication, several primitives are offered, including different synchronization mechanisms (synchronous, asynchronous or deferred) and higher level protocols, as Contract Net.

At this moment, the MIX platform (González, J.C. et al., 1.995) is made up of four elements:

- MSM (Multiagent System Model) C++ library, with the low level functionality of the platform. It is a modified version of the work carried out by Domínguez (1.992).
- ADL translator. ADL (Agent Description Language) is the language designed to specify agents. ADL files gather agents descriptions, and the translator generates C++ files and the appropriate makefile to obtain executables.

- CKRL ToolBox. A restricted version of CKRL (Common Knowledge Representation Language), developed by the MLT ESPRIT consortium (Cause, K. et al., 1.993), has been implemented to interchange information between agents[1]. This toolbox includes static and dynamic translators from CKRL descriptions to C++ classes and objects and vice-versa.
- Standard ADL agent definitions and CKRL ontologies.

4. AN APPLICATION: ECONOMIC CONTROL OF A FOSSIL POWER PLANT

A fossil power plant is a very complex process with a large number of variables upon which operators can actuate. The objective of this control system is to reduce the combustible consumption while generated power is kept constant. The first problem is that there not exists a reliable model of the process; so the system needs to learn how the power plant works. The second problem is that the quality of combustible used –a mix of anthracite and soft coal in the particular case of the power plant where the control system is going to be installed– changes every 5 minutes (there is a small homogenization of the last hour combustible, so coal quality changes with a smooth curve). This coal quality is part of the *heat rate* calculation, that is the optimization variable.

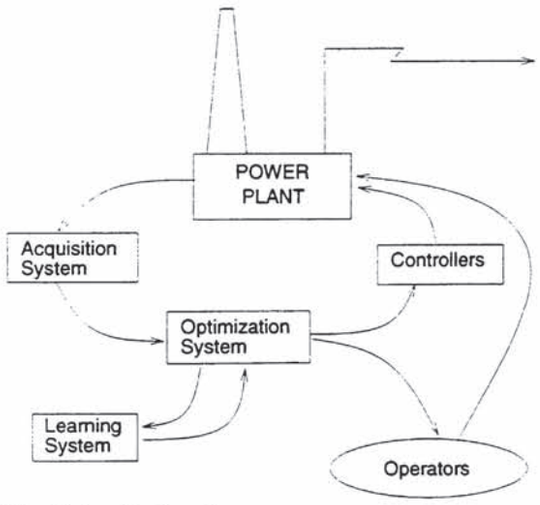

Fig. 2. Application diagram.

This last problem implies that the control system can have access only to an indirect estimation of the real heat rate. To solve it, a new performance criterion has to be determined. At design time, two variables are being analyzed to substitute the heat rate:

1.- *Principal air flow to the boiler*: This air flow carries the coal powder from mills to the boiler. So if this variable decreases, the combustible consumption decreases whichever the coal quality is.
2.- *Boiler output gas temperature*: A common sense analysis says that a lower temperature at the output of the boiler is better than a high one. In other case, heat is being wasted, so the plant is burning too much coal.

In both cases, the real optimization variable will be the ratio selected-variable/generated power, to obtain a relative consumption. After some performance tests in the power plant, one of both variables will be selected as objective.

In order to obtain good quality values for the control variables, a data acquisition system will filter the signals that reach the control system from sensors. The acquisition module gets 200 variables, and gives 23 to the optimization module. This 23 variables are known as the *context vector*. The optimization module will give 11 suggestions (over 11 operation variables) to controllers or operators –the so called *operation vector*–. The acquisition/filtering module is a very important part of the whole system: reliable inputs are even more needed that in the case of conventional control systems.

The control system (for some variables, a suggestion system) uses fuzzy logic to obtain the operation vector every 10 minutes. In order to make this fuzzy controller more accurate, the space of known states is divided in several big areas (called macrostates). These macrostates can be defined by experts (Velasco, J.R. et al., 1992), or computed using fuzzy clustering techniques (Velasco J.R. and Ventero, F.E., 1994) or a neural network. In this case, the second approach has been used.

To create the fuzzy knowledge bases, a modified version of the C4.5 algorithm (Quinlan, J.R., 1993) is used. This modification creates fuzzy rules from sample data files: to make the C4.5 function learn, the system must provide it a set of input vectors (context vectors) and the appropriate class for each vector. The system compares two consecutive vectors to determine when a cost reduction has been obtained and so, to classify the actions in the operation vector as bad, regular or good ones. After this classification, the algorithm creates fuzzy control rules.

The control system has as many rule bases as macrostates. When a new data vector is obtained, the control system asks the fuzzy clustering function about the appropriate macrostate. Since a given state may belong with different degrees to several macrostates, this

[1]The platform let us to use any other language for intercommunication between processes. In this way, KIF (Knowledge Interchange Format) (M. Genesereth, 1.992), another widely used language, is being considered as the second native language of the platform.

function selects the knowledge bases (KB in the following) to be used, along with their respective validity degree.

If the performance of the power plant is bad after several input vectors and several suggestions, the control system will ask the rule base generator for a new KB. This new KB will replace the old bad one.

Finally, suggestions made by the control system are used as set points by conventional controllers or human operators.

5. ADL AND CKRL SPECIFICATION

For the design of this application with the MIX platform, this distributed control system has to be seen as a set of agents with their respective goals and services, communicating them through exchanging messages. Figure 3 shows a graphical description of this system where each main action or group of actions may be seen as an agent with several goals/services.

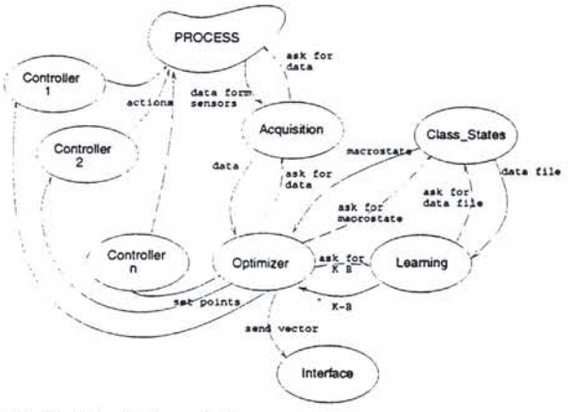

Fig. 3. Agents description

The *Acquisition* agent gets data from process sensors and gives context vectors to the optimizer upon demand. The *Optimizer* will ask the *Class_state* agent for the apropriate macrostate, and will use the correct(s) Knowledge Base(s) to obtain the operation vector. The values of the variables of this vector will be sent to specific *Controllers* as set point or will be shown to operators for a manual adjust. The optimizer agent will ask for a new KB to the *Learning* system if it sees that the cost value (the indirect heat rate) is growing up.

The MIX architecture uses ADL (Agent Description Language) as a specification/design language. From the ADL file, the MIX platform creates C++ agent files. After compiling and linking these files with the libraries, each agent will be an independent executable program which can run in a different computer. The complete ADL file for this application is shown in an appendix at the end of the paper. In this section just the agent definition process is going to be presented and it is going to be focused on the optimizer agent.

The Optimizer agent has as its proper goal the optimization of the heat rate. The pseudocode for this goal is as follows:

```
Repeat for ever
  Get context vector
  If heat rate is bad for n times
    Ask for new Knowledge Bases
  Ask for macrostate(s)
  Generate operation vector
  Set operation points to the controllers
  Tell operators manual actions
  Wait delay-time
```

In the code, bold face lines show service petitions that will be asked to different specialized agents: The *Acquisition* agent will give the context vector, the *Learning* agent will create new KBs, the *Class_states* agent will classify the context vector and each *Controller* will try to adjust the different set points.

However, at design level, the agent description only needs to know the name of required services (it does not have to know which agents will be available to perform them), the name of the functions that implement the services and goal, and the C++ file where this functions are described. The ADL description of the *Optimizer* agent is:

```
AGENT Optimizer -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "optimizer.C"
    REQ_SERVICES: Give_Last_Data;
                  Give_RB;
                  Classif_State;
                  Set_Point;
                  Send_Vector
  GOALS
    Optimize: CONCURRENT optimize
END Optimizer
```

When a service is specified, input and output types must be specified too. For instance:

```
AGENT Learning -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "learning.C"
    REQ_SERVICES: Give_Histo_Classified
  SERVICES
    Give_RB: CONCURRENT give_rb
        REQ_MSG_STRUCT powplant::Class
        ANS_MSG_STRUCT powplant::Rules
END Learning
```

In this case, Class and Rules are CKRL structures defined in the CKRL file. The MIX platform provides translation mechanisms to convert CKRL objects into C++ variables and vice-versa. The complete CKRL file is shown in the appendix.

6. CONCLUSIONS

Multiagent systems are proposed as an adequate approach for the design and implementation of distributed control systems. In particular, the multiagent platform developed for the MIX ESPRIT-9119 project is being used for the economic control of a fossil power plant. Although full evaluation of the system has not been yet finished, we can advance some preliminary conclusions. In comparison with the conventional (centralized) architecture previously used, the distributed solution shows evident advantages:

- Interfaces are more simple, so speeding up the development phase of the system life cycle.
- Control is more versatile, in the sense that this approach facilitates the simultaneous use of several controllers based on different techniques (with their own errors depending on the problem state).
- If error estimation is available as part of the output of the controllers, this information can be used to improve system accuracy.
- If a real time problem is faced, as the controllers have in general different response times, the system may decide upon the solutions at hand in any instant.
- Systems are more reliable in terms of fault tolerance and protection against noise.

7. REFERENCES

Causse, K, M. Csernel and J.U. Kietz (1.993). *Final Discussion of the Common Knowledge Representation Language (CKRL).* MLT Consortium, ESPRIT project 2154, Deliverable D2.3.

Domínguez, T. (1.992). *Definición de un modelo concurrente orientado a objetos para sistemas multiagente.* Ph.D. Thesis. E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid (in spanish).

García, J.A., J.R. Velasco, J.A. Castineira and J. Martín (1.993). *CORAGE: Control por Razonamiento Aproximado y Algoritmos Genéticos. Propuesta de Proyecto.* Project proposal for PASO-PC095 CORAGE Project (in Spanish)

Genesereth, M., R. Fikes and others (1.992). *Knowledge Interchange Format, version 3.0. Reference manual.* Computer Science Department, Stanford University.

González, J.C., J.R. Velasco, C.A. Iglesias, J. Alvarez and A. Escobero (1.995). *A Multiagent Architecture for Symbolic-Connectionist Integration.* MIX Consortium, ESPRIT project 9119, Deliverable D1

Quinlan, J.R (1.993), *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, USA.

Velasco, J.R., G. Fernández and L. Magdalena (1.992). Inductive Learning Applied to Fossil Power Plants Control Optimization, in *Symposium on Control on Power Plants and Power Systems*", IFAC, Munich, Germany

Velasco, J.R. and F.E. Ventero (1.994). Some Applications of Fuzzy Clustering to Fuzzy Control Systems in *3rd Int. Conf. on Fuzzy Theory and Technology*, (P. P. Wang (ed.)), 363-366 Durham, NC, USA.

APPENDIX: ADL AND CKRL FILES

A.1 ADL file

```
#DOMAIN "power_plant_domain"
#YP_SERVER "madrazo.gsi.dit.upm.es:6050"
              // Server of Yellow Pages Agent
#COMM_LANGUAGE CKRL
#MIX_LIBRARY
     "/export/home2/mix/tools/MIXcurrent"
#ONTOLOGY "powplant.ckrl"

// **** YP_Agent provides general services, as
// checkin, checkout, etc

AGENT YP_Agent -> YPAgent
END YP_Agent

// **** Process agent obtains data from
// sensors (this is its goal), filters
// them, and offer a service to give the
// last obtained to any other agents (in this
// example, to Optimizer)

AGENT Acquisition -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "acquisition.C"
  GOALS
    Collect_Data: CONCURRENT collect_data
  SERVICES
    Give_Last_Data: CONCURRENT give_last_data
        ANS_MSG_STRUCT powplant::Vector
END Process

// **** Optimizer agent asks for the last
// obtained data vector, asks to Class_States
// agent for the correct class of this vector,
// and obtain the rule-file-name of this class
// asking to Create_RB.
// Suggestions for control the process is
// given throw the standar output.

AGENT Optimizer -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "optimizer.C"
    REQ_SERVICES: Give_Last_Data;
          Give_RB; Classif_State;
          Set_Point; Send_Vector
  GOALS
    Optimize: CONCURRENT optimize
END Optimizer

// **** Class_State offers two services:
// Classif_State to Optimizer, in order
// to clasify vectors in the appropriate
// macrostate, and Give_Histo_Classified that
// suplies a file with data to learn rules.

AGENT Class_States -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "class_states.C"
  GOALS
    Create_States: CONCURRENT create_states
  SERVICES
    Classif_State: CONCURRENT classif_state
        REQ_MSG_STRUCT powplant::Vector
        ANS_MSG_STRUCT powplant::Class;
    Give_Histo_Classified:
                    CONCURRENT give_histo
        ANS_MSG_STRUCT powplant::Vector
END Class_States

//
// **** This agent creates rule bases, and
// give them to Optimize agent when
// they are needed
//

AGENT Learning -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "learning.C"
    REQ_SERVICES: Give_Histo_Classified
  SERVICES
    Give_RB: CONCURRENT give_rb
        REQ_MSG_STRUCT powplant::Class
        ANS_MSG_STRUCT powplant::Rules
END learning

// **** Interface shows operation vector
// to operator

AGENT Interface -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "interface.C"
  GOALS
    Show: CONCURRENT show_actions
  SERVICES
    Send_Vector: CONCURRENT send_vec
        REQ_MSG_STRUCT powplant::Vector
END Interface

// **** Controllers sets the specified value
// for the variable that it controls.

AGENT Controller_1 -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "controllers.C"
  GOALS
    Control: CONCURRENT control
  SERVICES
    Set_Point: CONCURRENT set_point
        REQ_MSG_STRUCT powplant::Point
END Controller_1

.....
.....

AGENT Controller_n -> BaseAgent
  RESOURCES
    REQ_LIBRARIES: "controllers.C"
  GOALS
    Control: CONCURRENT control
  SERVICES
    Set_Point: CONCURRENT set_point
        REQ_MSG_STRUCT powplant::Point
END Controller_n
```

A.2 CKRL

```
// Class of the process data.
// The class number is used to select the
// apropiate rule base for the process state
defsort intpos range (integer (1:*));
defproperty class_number sortref intpos;
defconcept Class
  relevant class_number;

// Process data is a list of values of
// variables obtained form sensors.
// Each variable has two values:
// the measure and a valid flag
defsort data list (real (0.0:1.0));
defsort valid list (integer (0:1));
defproperty vectordata sortref data;
defproperty vectorvalid sortref valid;
defconcept Vector
  relevant vectordata,vectorvalid;

// Set points are normalized real data
defsort point range (real (0.0:1.0));
defproperty pointdata sortref point;
defconcept Point
  relevant pointdata;

// Rules for rule-base communication
// rules are wrotten as strings, and
// decoded by optimizer
defsort regla_s range string;
defproperty regla_p sortref regla_s
defconcept Rules
  relevant regla_p;
```