IFAC

International Federation of Automatic Control

DISTRIBUTED COMPUTER CONTROL SYSTEMS 1995

A Postprint volume from the IFAC Workshop
Toulouse-Blagnac, France, 27-29 September 1995

Edited by

A.E.K. SAHRAOUI and J.A. DE LA PUENTE

PERGAMON

UK Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB, UK

USA Elsevier Science Inc., 660 White Plains Road, Tarrytown, New York 10591-5153, USA

JAPAN Elsevier Science Japan, Tsunashima Building Annex, 3-20-12 Yushima, Bunkyo-ku, Tokyo 113, Japan

First edition 1995

Library of Congress Cataloging in Publication Data

A catalogue record for this book is available from the Library of Congress

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0-08-042593 3

This volume was reproduced by means of the photo-offset process using the manuscripts supplied by the authors of the different papers. The manuscripts have been typed using different typewriters and typefaces. The lay-out, figures and tables of some papers did not agree completely with the standard requirements: consequently the reproduction does not display complete uniformity. To ensure rapid publication this discrepancy could not be changed: nor could the English be checked completely. Therefore, the readers are asked to excuse any deficiencies of this publication which may be due to the above mentioned reasons.

The Editors

Printed in Great Britain

WORKSHOP ON DISTRIBUTED COMPUTER CONTROL SYSTEMS 1995

Sponsored by:
IFAC - International Federation of Automatic Control
Technical Committee on Computers.

Organised by:
Association Française des Sciences et Technologies de l'Information et des Systèmes

International Committe

J.A. De La Puente, Chairman (Spain)
L. Boullart (Belgium)
A. Burns (United Kingdom))
A. Crespo (Spain)
F. Cristian (USA)
F. De Paoli (Italy)
J.C. Fabre (France)
M.A. Inamoto (Japan)
L. Ivanyoe (Hungary)
A. Keijzer (Netherlands)
H. Kopetz (Austria)
R. Lauber (Germany)
I. McLeod (USA)
J.J. Mercier (France)
A. Mok (USA)
S. Narita (Japan)
G. Qin (USA)
K. Ramamrithan (USA)
M.G. Rodd (United Kingdom)
G. Suski (USA)
J.P. Thomesse (France)
G. Zhao (Singapore)

National Organizing Committee

A.E.K. Sahraoui (Chairman)
E. Bernauer
J.C. Deschamps
E. Dufour
M.T. Ippolito
M. Tuffery
D. Vielle

CONTENTS

METHODOLOGIES FOR DCCS II

TEMPORAL PROPERTIES IN DCCS II

DEPENDABILITY ISSUES IN DCCS

SYSTEM ANALYSIS

REAL-TIME COMMUNICATION

APPLICATIONS

AN APPROACH DESIGNING PARALLEL SOFTWARE FOR DISTRIBUTED CONTROL SYSTEMS

H. Unger*, B. Däne and W. Fengler ****

**University of Rostock, Department of Informatics, D-18051 Rostock; Germany. E-mail: hunger@informatik.uni-rostock.de*

***Technical University of Ilmenau, Department of Informatics and Automation, D-98684 Ilmenau; Germany. E-mail: bdaene!wfengler@theoinf.tu-ilmenau.de*

Abstract. Petri Nets have been proved to be an effecient tool to represent complicated systems. Nevertheless, in general it is not easy to implement a technical system given as a Petri Net on a multiprocessor system. This contribution presents a new approach for this procedure. The main difference compared to other methods is the effective use of message passing communication during the implementation.

Key Words. Petri-nets; Distributed computer control systems; Parallel programs

1. INTRODUCTION

Progress in hardware design makes it possible to use multiprocessor architectures even in small automation systems. Parallel programming requires effective methods to find out parallel executable parts in a given algorithm (Boillat, *et al.*, 1991). Therefore it is necessary to solve a lot of problems in a transparent way for a wide group of users. That is why Petri Nets, a graphical language of description, became more and more important for modelling parallel software solutions (Reisig, *et al.*, 1987).

But there are only a few approaches for implementing Petri Net models on different multiprocessor architectures (Thomas, 1991; Unger, 1994). An overview is given in figure 1.

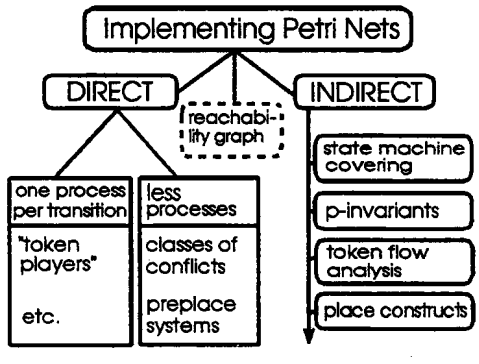

Fig. 1. An overview about existing implementation approaches

From the authors point of view all known methods fall into two basic types. The first one - the so called direct type - implements processes according to the transitions of the net. The second, indirect one is to decompose or to cover a given Petri Net by state machines, and then to implement one process for every state machine. Especially if a Petri Net has many transitions, the first method yields in each case a solution with plenty superfluous of processes and a large communication overhead. In general, the second group of methods generates a more efficient code, but, in contrast to the first one, it does not apply to all Petri Nets.

The main disadvantage of known approaches is the transformation of a subset of places into global data objects in a shared memory. These data objects normally contain integer values corresponding to the number of tokens in the places. Accessing the data objects by more than one process causes a lot of management problems and aggravates real parallel work of these processes. In the end a lot of technical systems like transputer systems or PVM implementations [1] require a client server relation instead of a shared memory for solving this problem and so the number of parallel working processes is increased.

The present paper shows a new approach for an implementation avoiding the disadvantage described above.

[1] Parallel Virtual Machine for UNIX clusters from the Oak Ridge National Laboratory (Sunderam, 1990)

2. BASIC CONCEPTS

Usually a Petri Net Φ is a 5-tupel (P, T, F, V, m_0) such that

(i) P, T are disjoint finite nonempty sets, the sets of places and transitions, respectively
(ii) $F \subseteq P \times T \cup T \times P$, the set of arcs
(iii) $V : F \rightarrow \mathbf{N}$, the multiplicity function
(iv) $m_0 : P \rightarrow \mathbf{N_0}$, the initial marking ($\mathbf{N}$ and $\mathbf{N_0}$ denote the sets of positive and nonnegative integers, respectively.)

A transition $t \in T$ is able to fire at a marking m if for every $p \in P$, $(p,t) \in F$

$$m(p) \geq V((p,t))$$

Firing $t \in T$ at m means to substitute m by m_{new} where

$$m_{new}(p) = \begin{cases} m(p) - V((p,t)) & : \ (p,t) \in F \\ m(p) + V((t,p)) & : \ (t,p) \in F \\ m(p) & : \ else \end{cases}$$

for any $p \in P$.

Additionally is defined: $pF = \{t|(p,t) \in F\}$, $Fp = \{t|(t,p) \in F\}$, $tF = \{p|(t,p) \in F\}$ and $Ft = \{p|(p,t) \in F\}$.

For modelling automation systems it is necessary to add some components to the standard Petri Net definition in order to describe the input and the output of data (Fengler and Philippow, 1991):

(1.) $\mathbf{w_X}$,
a set of boolean expressions associated to the transitions.
If $t \in T$, $w_x(t)$ is considered to be an additional condition to fire t.

(2.) $\mathbf{w_Y}$,
a set of boolean output variables associated to the places of the Petri Net.
$w_y(p) \in w_y$ is $TRUE$, if p is labeled.

(3.) $\mathbf{w_A}$,
a set of procedures associated to the places of P.
Procedures are started when a new token reaches the place.

Implementing a given Petri Net means to transform it into a program by interpreting sets of elements as structures of a parallel program. When doing so, the state of the program or a class of its states can be derived from an actual marking and vice versa.

3. TRANSFORMATION

In the following, a Petri Net transformation is shown resulting in a net with particular properties. It is based on separating conflict structures followed by a transformation of the remaining net. Afterwards, the net can be implemented in a message based manner.

3.1. *Conflict situations*

Conflicts directly influence the transformation of a Petri Net. Places with more than one posttransition are the reason for conflicts in a Petri Net. Such constructs are called static conflict situations. For the present contribution it is necessary to consider several static conflicts in a given Petri Net Φ in a more detailed way (see figure 2). All the structures consist of a set of transitions A and a set of preplaces S of the transitions of A in such a way that there is at least one transition to each other one which has a common preplace.

All non-free-choice conflict structures result in problems during the (basic) transformation and have to cut out in a first step described below.

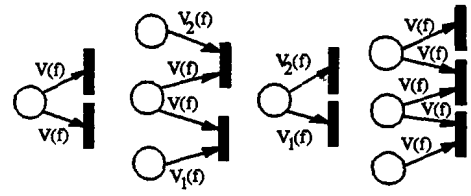

Fig. 2. Static Conflict Structures in a Petri Net

Let Π and Θ be set systems for all conflict structures of a given Petri Net with

$$\Pi = \{S_1, S_2, ..., S_h | h \in \mathbf{N}\}$$

and

$$\Theta = \{A_1, A_2, ..., A_h | h \in \mathbf{N}\}$$

The function $K(\Pi, \Theta)$ is defined as follows:

$$K(\Pi, \Theta) = \begin{cases} (\Pi', \Theta') : \exists i,j : A_i \cap A_j \neq 0 \\ \Pi' = (\Pi \setminus S_i \\ \setminus S_j) \cup \{S_i \cup S_j\} \\ \Theta' = (\Theta \setminus A_i \setminus A_j) \cup \{A_i \cup A_j\} \\ (\Pi, \Theta) : \forall i,j : A_i \cap A_j = 0 \end{cases}$$

Obviously, there is a $k \in \mathbf{N}$ such that $K^k(\Pi, \Theta) = K^{k+1}(\Pi, \Theta)$. In this case $K^k(\Pi_0, \Theta_0)$ is called a

maximal conflict set.

For $Q = \{q|q \in P, |pF(q)| > 1\}$, (Π_0, Θ_0) with

$$\Pi_0 = \{M_i|M_i = \{q_i\}, i = 1(1)|Q|\}$$

and

$$\Theta_0 = \{N_i|N_i = \{t|(q_i, t) \in F\},$$

$$i = 1(1)|Q|\}$$

is the set of places and their posttransitions which could be the source of a conflict. Furthermore, the connection between some of such sources via their transitions (figure (2d)) is represented in the maximal conflict set $K^k(\Pi, \Theta)$.
In order to get a set with all preplaces of $t \in \Theta$ in $(\Pi, \Theta) = K^k(\Pi_0, \Theta_0)$ the set system Π is modified by $\Pi' = \{p|\exists t \in \Theta : (p, t) \in F\}$.

For further transformation such structures (see figure 3) have to be cut out from a given Petri Net Φ. The main idea consists in a functional separation of the pre- and the postarea of a transition. The fireability of such a transition can completely be tested in the first subnet. The postarea of the transition located in the second subnet only sets tokens on places, when this transition has got a message from the prearea.

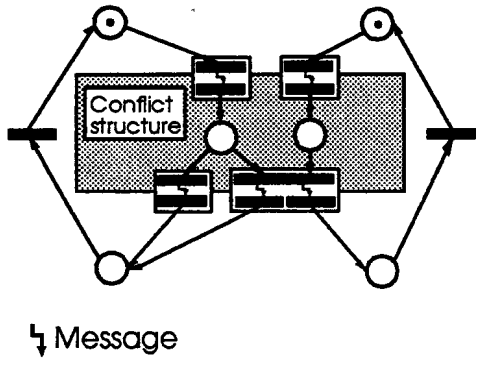

Fig. 3. Separation of Conflict Structures

A later discussion shows that only the more difficult conflict situation in figure (2c) must be cut out.

3.2. *Transformation of the remaining Petri Net*

The transformation of the modified Petri Net $\Phi = (P, T, F, V, m_0)$ (a net without static conflict structures) described in this section is carried out in three steps. At first, an unmarked place construct $(P'(p), T'(p), F'(p), V'(p))$ is defined for each $p \in P$ of a given Petri Net Φ. After doing so, these constructs will be joined by arcs, and a corresponding marking m' is defined. Thus, one gets a corresponding Petri Net $\Phi' = (P', T', F', V', m')$ with $P' = \bigcup P'(p)$, $T' = \bigcup T'(p)$ and $F' \supset \bigcup F'(p)$.

(1.)
Let $p \in P$, $t_{out} \in T$ the only transition with $(p, t_{out}) \in F$ and V_{out} the multiplicity of (p, t_{out}). Then is defined

$$u = V_{out} + max(V_i|i = 1(1)|Fp|) - 1.$$

Now each $p \in P$ will be transformed into a place construct with a set of places $P'(p)$ defined by

$$P'(p) = \{p'_0, .., p'_i, x_1, .., x_e\}$$

with $i = 0(1)u$ and $e = 1(1)|t_{out}F|$.

For the definition of the sets of transitions and arcs $C_1(p)$, $C_2(p)$ and $C_3(p)$ are defined by:

$$\begin{aligned} C_1(p) &= \{(a,b,c)|a = 0(1)V_{out} - 1, \\ &\quad b = 1(1)|Fp|, c = a + V_b : \\ &\quad a + V_b < V_{out}\} \\ C_2(p) &= \{(a,b,c)|a = V_{out}(1)u, b = 0, \\ &\quad c = a - V_{out} : a \geq V_{out}\} \\ C_3(p) &= \{(a,b,c)|a = 0(1)V_{out} - 1, \\ &\quad b = 1(1)|Fp|, c = a + V_b - V_{out} : \\ &\quad a + V_b \geq V_{out}\} \end{aligned}$$

With these definitions let

$$C(p) = \bigcup_{i=1}^{3} C_i(p).$$

Corresponding to the elements of $C(p)$, the following transitions and arcs are added for each $(a, b, c) \in C(p)$ to the sets $T'(p)$ and $F'(p)$, respectively:

$$t_{a,b,c}(p) \in T'(p),$$

$$(p'_a, t_{a,b,c}) \in F'(p) \quad with \quad V((p'_a, t_{a,b,c})) = 1$$

and

$$(t_{a,b,c}, p'_c) \in F'(p) \quad with \quad V((t_{a,b,c}, p'_c)) = 1.$$

Finally, for each $(a, b, c) \in C_2 \cup C_3$ arcs have to be added with

$$(t_{a,b,c}, x_i) \in F'(p) \quad with \quad ((t_{a,b,c}, x_i)) = 1$$

for all $i = 1(1)|t_{out}F|$.
In a last step places without pretransitions and their postarcs and posttransitions will be removed from the place constructs. An example of such a place construct is given in figure 4.

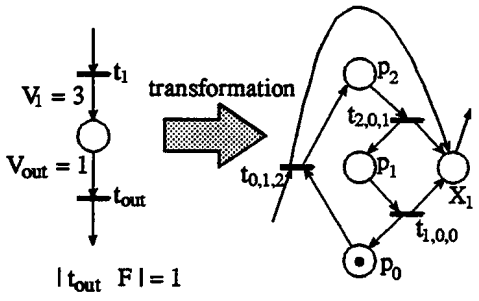

Fig. 4. Example of an Easy Place Construct

(2.)
Let $e = 1(1)|t_{out}F|$. Then one $x_e \in P'(p)$ exists corresponding to each of the postplaces $v_1, v_2, .., v_e$ of t_{out}. Furthermore $t_{a,b,c}(v_e) \in T'(v_e)$ are the transitions of the corresponding place constructs. Now for all $(a, b, c) \in C_1(v_e) \cup C_3(v_e)$ add an arc to F' with

$$(x_e, t_{a,b,c}(v_e)) \in F' \quad with \quad V((x_e, t_{a,b,c}(v_e))) = 1.$$

(3.)
A marking m' of Φ' is said to be corresponding to m of Φ if for all place constructs

(i) $\sum_{p'_i \in P'(p)} m'(p'_i) = 1$
(ii) $\forall i, j : m'(x_i(p)) = m'(x_j(p))$
(iii) $\forall i, j : if \quad m'(p'_i) = 1, p'_i \in P'(p)$
$i + m'(x_j(p)) * V_{out} = m(p)$

The result of the transformation is a transformed Petri Net Φ' which simulates the behaviour of Φ. An important property of Φ' is that the multiplicity function is equal 1 for all arcs of the net.

4. IMPLEMENTATION

Implementing Φ' means to find out interpretations for special elements of the given Petri Net. This work falls into two parts: implementing the conflict structures and implementing the transformed remaining Petri Net Φ'.

The main problem with implementing conflict structures results from the shared use of a data object representing places with more than one posttransition. The new approach avoids these problems, because all elements of the conflict structure $(K(\Pi, \Theta))$ will be cut out and implemented as a single process, containing all elements for the complete solution of the conflict in a loop. The connection of the conflict structures with the remaining net can be represented by messages, as described above.

Now consider the remaining Petri Net Φ'. One advantage of the described transformation is that the place constructs without the places x_i are state machines. These state machines are connected via x_i and their incident arcs, thus forming so-called systems of concurrent state machines (SCS).

In a first approach these SCS can be implemented by creating a single process of a parallel program for each state machine (Unger, 1992). Following this idea, the x_i-elements of Φ' are interpreted as communication structures between these processes.

Places connecting state machines are usually implemented as data objects in a shared memory or a server process. But resulting from the transformation described above, each x_i has prearcs only relating to transitions in exactly one state machine, and has postarcs only relating to transitions in exactly one other state machine. Therefore, information about the state of any x_i will be managed by only one process and so this communication can be implemented by the use of *send* and *receive* procedures and the belonging message buffers.

A second approach implementing the transformed Petri Net is based on special structure effects in the transformed Petri Net. Consider Φ' without the places p_i of P' and without their transitions $t_{a,b,c}$ derived from the elements of $C_2(p)$. It can be shown that such nets consist of six basic elements with an interpretation shown in figure 5. Because the multiplicity of all arcs is equal 1, each token in one of the x_i-places corresponds to a set of parallel processes corresponding to the given interpretation of elements. For more than one token one gets a superposition of such process groups.

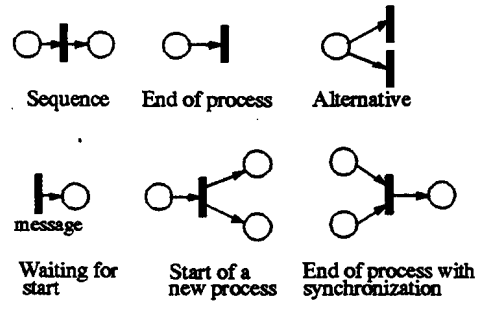

Fig. 5. Elements in the Reduced Transformed Petri Net

In all cases there is the restriction that in a given moment only one transition of each place con-

struct can be fired. This will be achieved by a special interpretation of the p_i-elements of the transformed Petri Net. The marking of these places can be considered as special values of a marking of p in the original net. Thus the values can select the fireable transitions and in this way solve the conflicts in the processes. In the parallel program the value of a counter will be implemented by messages circulating between the processes. Only one process can receive the message, and therefore only one process can do the next step corresponding to the firing process of exactly one transition. Leaving the sector of the given place construct the process sends a message with the new counter information and any process that needs this information can receive it.

At last, consider the interpretation of the transitions $t_{a,b,c}$ derived from the elements of $C_2(p)$. Firing one of these transitions entails creating tokens on x_i and processes, respectively. The firing process of these transitions directly depends on firing $t_{a,b,c}$, if $t_{a,b,c}$ derives from C_3 and $c \geq V_{out}$. This algorithm is implemented by creating a new process which receives as its argument the data from the circulating message. The mentioned process creates other new processes, changes the information of the message ($-V_{out}$ for each new process up to the moment when the data are less V_{out}) and sends the updated message to any process requiring it. The choice of the implementation method depends on the properties of the given net. If the number of places is not too high, the first approach is more effective, a lower number of tokens favours the second method but a mixed use of both methods is possible too.

Results from a experimental implementation of a control program achieved by several methods are shown in figure 6.

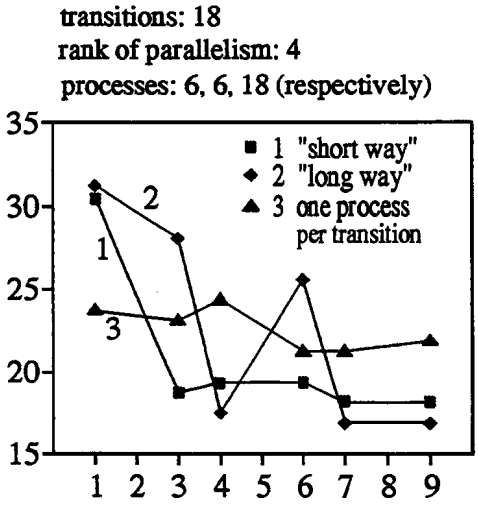

Fig. 6. Time Behaviour of a Parallel Program

5. CONCLUSION

A new method for the automatic generation of parallel software from Petri Nets has been shown and some transformation and implementation details have been discussed. The method enables the generation of more efficient parallel code by preventing some communication overhead resulting from conflict situations in the net. A first experimental implementation has shown the expected results.

6. REFERENCES

Boillat, J. E. *et al.* (1991). *Parallel Computing in the 1990's.* Institut für Informatik, Universität Basel.

Fengler, W. and I. Philippow (1991). *Entwurf industrieller Mikrocomputersysteme.* Carl Hanser Verlag, München-Wien.

Reisig, W., W. Brauer and G. Rozenberg (1987). Petri nets: Applications and Relationships to Other Models of Concurrency. In: *LNCS 255.* Springer Verlag, Berlin-Heidelberg-New York.

Sunderam, V.S. (1990). Parallel Virtual Machine. In: *Concurrency: Praktice and Experience*, **No. 12,** 315-339.

Thomas, G.S. (1991). *Parallel Simulation of Petri Nets.* Technical report, University of Washington.

Unger, H. (1992). A Petri Net Based Method to the Design of Parallel Programs for a Multiprocessor System. In: *LNCS 634.* Springer Verlag, Berlin-Heidelberg-New York.

Unger, H. (1994). *Untersuchungen zur Implementierung von Petri-Netz-Modellen auf Mehrprozessorsystemen.* Dissertation, TU Ilmenau.

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 (Universidad 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.

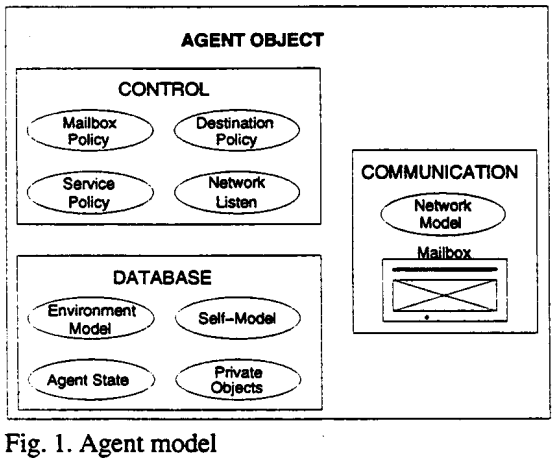

Fig. 1. Agent model

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.

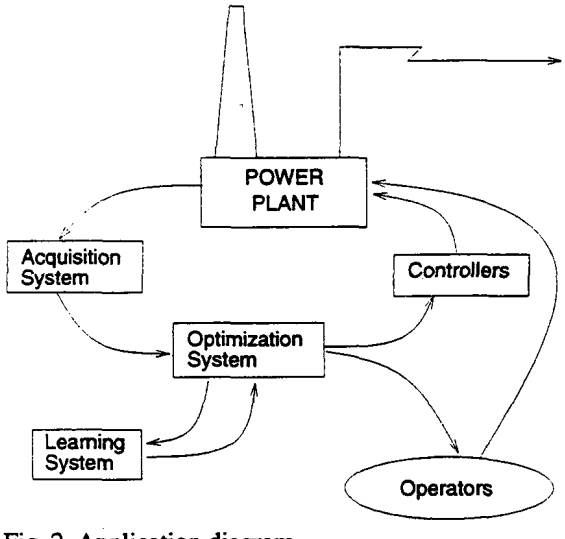

Fig. 2. Application diagram.

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.

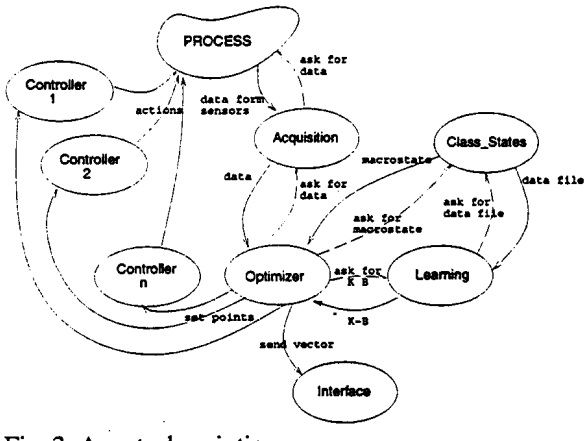

Fig. 3. Agents description

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.