INTERACTING SOFTWARE-AGENTS TO SUPPORT A KIND OF EXPERT SUPERVISION SYSTEMS

Contreras, O.C., De La Rosa, J. and Melendez, J.

Grup d'Enginyeria de Control i Sistemes Intel.ligents (*eXiT*), Institut d'Informàtica i Aplicacions (IIiA), Universitat de Girona (UdG) & European Associated Laboratory-Intelligent Systems and Advanced Control (LEA-SICA), Av. Lluís Santaló s/n, 17071 Girona (Spain) T.+3472 418391/418487 F.+3472 418098 E-mail: {orlando, peplluis, quimmel}@eia.udg.es

Abstract - *Expert supervision systems are software applications specially designed to automate the process monitoring. The goal is to reduce the dependency on human operators to assure the right operation of a process including when faulty situations. Construction of this kind of applications involves an important task of design and development in order to represent and to manipulate process data and behaviour at different degrees of abstraction for interfacing with data acquisition systems connected to the process. This is an open problem that complicates notably with the number of variables, parameters and relations to account for the complexity of the process. Multiple specialised modules tuned to solve simpler tasks that operate under a co-ordination provide a solution. A modular architecture based on concepts of software-agents, taking advantages of the integration of diverse knowledge-based techniques, is proposed for this purpose. The components (software-agents, communication mechanisms and perception/action mechanisms) are based on ICa (Intelligent Control architecture), a software middle-ware supporting building-up of applications with software-agent features.*

1. INTRODUCTION

Nowadays, the requirements related to quality assurance and uniformity of products together with exigencies of availability and flexibility of processes cause the necessity of automation of surveillance systems. The detection of deviations from normal operation and the proposing of appropriate correction actions are the tasks of **expert supervision systems (ESSs)**. These systems are software applications specially designed to automate the process monitoring. The goal is to reduce the dependency on human operators to assure the right operation of a process including when faults (misbehaviours) are present. Three basic tasks are differentiated to achieve the supervision goals: fault detection (analysis of process variables and detection of deviations), fault diagnosis (reasoning on detected faults for determining the origin) and reconfiguration (proposing of correction actions to recover normal operation condition). These tasks constitute the rationale that an ESS should have as influence on the process behaviour through variables, parameters and relations (among them) of a process behaviour model.

Multiple knowledge-based techniques and methods (heuristic rules, fuzzy logic, analytic reasoning, qualitative reasoning, neural network and so on) have been proposed to achieve the supervision goals [1]. Those techniques and methods that take benefit of knowledge, experience or heuristics extracted from process operators and engineers constitute the base of the **expert process supervision**. But none of the techniques and methods is a unique solution. The results could be improved by combining them in order to take benefit of all available information from both the process data and behaviour.

The necessity of representing and processing data and behaviour at different degrees of abstraction, and of interfacing with data acquisition systems connected to the process is an open problem that increases notably with the number of variables, parameters and relations to treat the process complexity. As a consequence, it is difficult to build a unique structure to a decision system based on the flow of information as in Fig. 1. In such case, multiple specialised modules tuned to solve simpler tasks that operate under co-ordination provide a solution. Software-agents [2], [3] offer capabilities (solving focus, autonomy, co-operation, etc.) that can solve the complexity of dealing with multiplicity of tuned tasks to achieve the supervision goals.

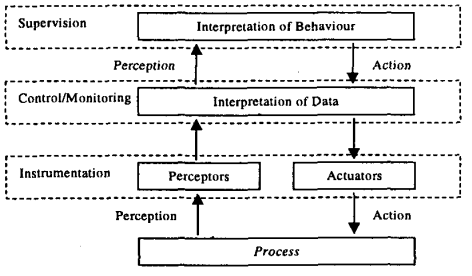

Fig. 1 An overview of the expert process supervision

0-7803-5670-5/99/$10.00 ©1999 IEEE

In the following sections a modular architecture based on concepts of software-agents is proposed to achieve supervision goals in complex processes taking advantage of integration of diverse knowledge-based techniques and methods. The components (software-agents, communication mechanisms and perception/action mechanisms) are based on ICa (Intelligent Control architecture)[1] [4], a software middle-ware to support building-up of applications with software-agent features. Conclusions and future work are also presented.

2. COMPUTATIONAL COMPLEXITY OF EXPERT SUPERVISION SYSTEMS

The computational complexity in the design and development of ESSs is basically due to:

- The information comes from heterogeneous sources: data are quantitative/qualitative values obtained from process variable measures. Behaviour is defined through analytic/heuristic models according to the knowledge that operators and engineers have on process operation. Models describe the situations of interest that determine the behaviour through variables, parameters and relations (among them) in mathematical terms, or qualitative terms such as causalities or if-then rules, or hybrid [1].
- The information comes from distributed sources: process might be the combination of multiple interacting components.
- The information is imprecise: some situations are ambiguous due to the fact that the knowledge on process behaviour (or part) is based on experience from operators and engineers. Some situations are imprecise due to uncertainties in the behaviour, e.g. intermittent or unpredictable faults. Some relations are imprecise descriptions of numerical variable magnitudes; e.g. considering the symbol "*too hot*" for a variable "*temperature*", it is an imprecise interpretation of "*temperature*" as it represents certain range of temperature values.
- The information is insufficient: data acquisition systems provide only real, actualised and instantaneous values of variables, which do not constitute sufficient information to know the process state. In addition, those values must be elaborated (by means of additional analysis functions) and stored in order to deduce deviations.
- The information is time dependent: process evolves through the time so that input data must be periodically actualised and output responses must be produced in a restricted time. Also, information on past process states is needed for knowing the present process state.
- The information is voluminous: big number of variables, parameters and situations might be involved in the process behaviour (consequence of size and complexity); a big number of relations are needed for describing that behaviour.

[1] ICa was developed by the Autonomous Systems Laboratory (ASLab) of the Universidad Politécnica de Madrid (Spain). The *eXiT* group has been authorised to use ICa in researches and developments.

3. STATE OF THE ART

Researchers have tackled the ESS computational complexity with the integration of multiple applications, motivated by the positive aspects of distributed processing performance, reliability, flexibility, modularity and resource sharing [5], [6], [7], [8], [9], [10]. Researches have been centred mainly on the co-operation among expert systems [11], [12] and their integration with database systems with capabilities to store and update information from process [9], [13]. Generally the integration have been carried out with object-based techniques and the co-operation with exchange of information based on methods. The interfacing with data acquisition systems has been generally carried out with *dynamic data exchange* (DDE). But, that integration has always been directed to closed solutions composed of applications that work together.

4. OUTLINE OF THE SOFTWARE-AGENT BASED EXPERT SUPERVISION SYSTEMS

The ESS computational complexity is tackled with modules based on software-agents and focused on the treatment (acquisition, abstraction, storing, controlling and reasoning) of process data and behaviour. Specialised software-agents should supervise multiple parts of a process, whose interactions might allow supervising the global process; a software-agent should have knowledge on only the behaviour of a process part. The proposed modular architecture is founded on this conception and it is named **software-agent based expert supervision system architecture**. So, modules are software-agents focused on acquisition, abstraction, storing, controlling and reasoning of process information, which interact to achieve the supervision goals.

A **software-agent based ESS (A-ESS)** is defined as a "*software application with the ability to sense a process and act on it, composed of specialised software-agents for reasoning (detecting and diagnosing faults) about process behaviour in order to propose appropriate actions to maintain the operating conditions in case of faults*". Process constitutes the environment where the software-agents inhabit. Process variables data (measures and abstractions) constitute the perceptions, which determinate the current process state. Fig. 2 shows a view of the expert process supervision based on software-agents.

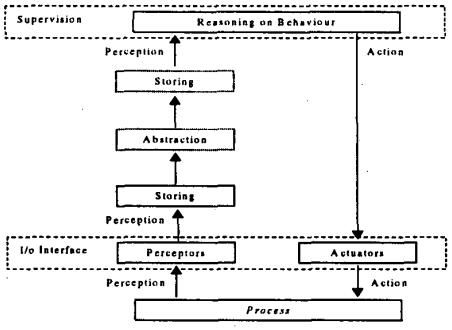

Fig. 2 An overview of the software-agent based expert process supervision

4.1 Basic Components

The basic A-ESS components are software-agents. The behaviour of a software-agent is determined by the services that it supports, with an A-ESS, for dealing with process information: acquisition, abstraction, storing, controlling and reasoning. So, four kinds of software-agents named abstractor agents, reasoning agents, control agents and facilitator agents are defined. Perception/actuation mechanisms named perceptors/actuators and a kind of databases named perception bases are also defined. The tasks related to all these components are the following:

- **Perceptors** and **actuators** constitute the interface with a process. Perceptors perceive real, actualised variable measures. Actuators execute actions on process (by means of reconfiguration).
- **Perception bases** store variable data (measures and abstractions) that indicate to software-agents how a process evolves through the time.
- **Abstractor agents** are in charge of abstracting information from acquired variable measures. They elaborate (by means of analysis functions) significant information for interpreting the current process behaviour.
- **Reasoning agents** are in charge of reasoning on perceptions. The tasks associated to this kind of agents are to detect faults, to diagnose faults and to propose (partial) actions to cope with them.
- **Control agents** are in charge of controlling the information flow and the restrictions of time among process and software-agents, and of taking final decisions to cope with faults.
- **Facilitator agents** are in charge of performing support operations for other software-agents, e.g. mathematical operations.

The communication roles among A-ESS components are established on client/server bases in the following way (see Fig. 3):

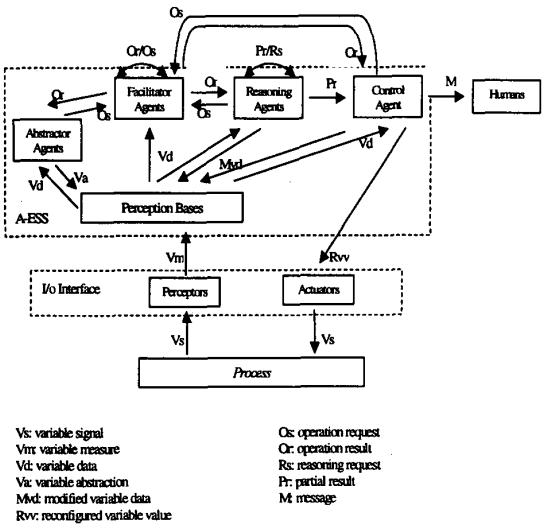

Fig. 3 Process – software-agents - humans interaction based on the A-ESS structure

Perceptors
- They are servers of real, measured variable values to perception bases.

Actuators
- They are clients of reconfigured variable values from control agents.

Perception Bases
- They are clients of real, measured variable values from perceptors.
- They are clients/servers of variable data (measures and abstractions) from/to abstractor agents, reasoning agents and control agents.
- They are servers of variable data to facilitator agents.

Abstractor agents
- They are clients of variable data from perception bases.
- They are servers of variable abstractions to perception bases.

Reasoning agents
- They are clients/servers of variable data from/to perception bases.
- They are clients for services from facilitator agents.
- They are clients/servers for services from/to other reasoning agents.
- They are servers for services to control agents.

Control agents
- They are clients/servers of variable data from/to perception bases.
- They are clients for services from facilitator agents.

- They are clients for services from reasoning agents.

Facilitator agents

- They are clients of variable data from perception bases.
- They are servers for services to abstractor agents, reasoning agents and control agents.

4.2 Operation Cycle

When a process is in operation, an A-ESS perceives real, actualised variable measures by means of perceptors, which are saved into perception bases. Control agent informs to the abstractor agents on new changes in acquisitions. Then, abstractor agents access the perception bases and apply abstraction functions on the data. They save results into the perception bases. At the same time, they acquaint control agent about the new state of perception bases. The control agent asks reasoning agents on process behaviour. Those software-agents access the perception bases and reason on available data to detect and diagnose faults (if exist). At the same time, a reasoning agent could request to one other for partial results (if needed). Also, it could request to a facilitator agent on needed operations. Depending on decisions, reasoning agents save modified variable data into perception bases, or send partial solutions to one other reasoning agent, or to the control agent. Also at that time, control agent could request to a facilitator agent on needed operations. It reasons on results and takes final decisions. In that case and depending on decisions, it sends reconfigured variable values to actuators or messages to human operators and engineers.

4.3 Basic Features

With the A-ESS approach, we expect to gather up a set of desirable features in order to decrease the ESS computational complexity. These are:

- **Modularity:** perceptors, actuators, perception bases and software-agents are self-contained entities that make A-ESS easier to understand, to build and to maintain.
- **Solving focus:** not all information is needed for solving all tasks in the supervision goals. The software-agents might be designed and developed in a way that is more likely to pay off.
- **Hierarchical structure:** an A-ESS is understood as a hierarchical organisation for managing process data and behaviour. Components have different ranks in the structure: acquisition, abstraction, storing, facilitation, reasoning and control.
- **Integration of heterogeneous components:** software-agent knowledge might be constructed with the most appropriate software technology, whether heuristic rules, procedural programming, fuzzy logic and so on, or hybrid.
- **Sharing reasoning:** software-agents could share information (calculated data or partial results) for making decisions to arrive to global supervision solutions (final actions). Then, the co-operation is needed because none of all the software-agents should have global view of the solutions.
- **Distributed work:** the available data from a process might be located in different logical/physical points (according to process nature). From the point of view of inter-operation, the components could inter-act and work on a network of machines with the overall functionality distributed among these machines.
- **Interfacing with data acquisition systems:** perceptors and actuators constitute the data input/output interface between process and A-ESS. Perceptors throughput real, actualised variable values into perception bases. Control agents throughput reconfigured variable values in actuators. Both must become part of data acquisition systems.
- **Interfacing with humans:** control agents should inform humans on process situations.
- **Reusability:** if two tasks are functionally similar, one same software-agent could achieve them. Also, once a set of components have been constructed for one A-ESS, it should be possible to construct new ones that use these components.
- **Software patterns:** software patterns for every component that can be reused to implement several entities on them, allowing that process engineers should focus on the task-solving rather than on the design of the components.
- **Evolution/maintenance:** if a process changes, the modifications on the A-ESS structure must be done only on the components where the changes are involved. Also, replace and/or add components to modify that structure, according to process modifications.
- **Openness:** all components could be integrated with other software applications in different supervisory control environments. For this, implementations on software patterns of perceptors and actuators might become part of the other applications.

5. DESCRIPTION OF THE SOFTWARE-AGENT BASED EXPERT SUPERVISION SYSTEM ARCHITECTURE

Perceptors, actuators, perception bases, and software-agents are created with agent interfaces based on ICa. Communication and interaction mechanisms are created with ICa communication means. ICa is a distributed-object-based software middle-ware to support the build-up of flexible and reusable distributed applications and services with software-agent features [4].

5.1 Characteristics of the ICa

ICa is an object-based framework developed in C++ language that allows building applications (with software-agent features) on distributed objects in the frame of process control. It is based on CORBA (Common Object Request Broker Architecture)[2] specifications [14], with extensions to cope with requirements of industrial environments (time-dependency, fault tolerance and, mainly, multithreading).

The ICa has the following features:

- It has ICa Agent Definition Language (ICa ADL) to generate software-agent interfaces[3], as software-patterns[4], in which attributes and operations that determine the identity and the behaviour of software-agents are defined. ICa ADL is both a declaration language and a programming language on C++ language.
- It has ICa Object Request Broker (ICa ORB), a communication infrastructure that allows requests/receives among ICa software-agents across distributed heterogeneous computer-environments. ICa offers various pre-constructed communication means to pass information among ICa software-agents called transports, providing support for shared memory and TCP/IP communications. Of course, it is possible to develop new transports on the pre-constructed transports for domain specific domain architectures.
- It presents a common API (Application Programming Interface) and behaviour in all the platforms that it operates hiding hardware and operating system particularities, and allowing for platform-independent common management of resources such as threading. It permits interoperability among platforms such as Win32 on Intel x86, Linux on Intel x86, etc.
- The ICa ADL has mapping for Java language. It permits the development of open applications that inter-operate in C++ and Java environments.
- It incorporates capabilities for development of real-time systems such as timeout call, call processing-time estimation and dynamic thread priority management. Plus, it incorporates supports for static and dynamic redundancy that permits the development of fault-tolerant systems.

[2] CORBA is a public-license open standard for the construction of distributed-object-based applications. The Object Management Group (OMG) developed it.
[3] Interfaces are similar to classes in C++ language and interfaces in Java language.
[4] A software pattern is a pre-designed component that fit on fixed situations [14], [16].

5.2 Structures of the A-ESS Components

Software patterns for perceptors, actuators, perception bases and software-agents are constructed on ICa agent interfaces. Software patterns for communication mechanisms are constructed on ICa communication means. They are pre-designed components that span the A-ESS components and which could be reused to implement several entities on them. The software patterns for perceptors and for actuators capture the design of data input/output interfaces and of interaction between process and A-ESS. The software pattern for perception bases captures the design and the interaction of this kind of components into an A-ESS. The software pattern for software-agents specifies the roles and interactions of all kinds of software-agents into an A-ESS. The software patterns for A-ESS communication mechanisms specify the communications among all A-ESS components. All these software patterns should allow that control engineers could focus on the task solving rather than on the design of components.

Perceptors share a basic structure consisting of:

- **Identification layer:** that identifies perceptor. It has attributes that define the identity of a perceptor such as name, self-number and description.
- **Interfacing layer:** that permits the interaction with perception bases. It constitutes the transmission mechanism, and is made with an ICa transport[5] and a defined method to support the throughputs of process variable measures into the perception store. Implementers fix the variables in the data acquisition system. The transmission method is:
  - put(pb_name, data): puts variable measures into a specified perception base.

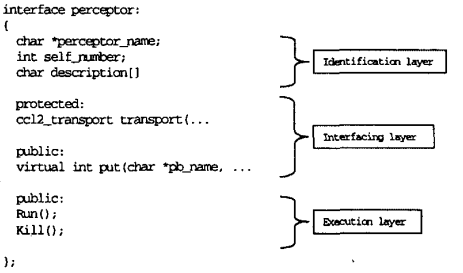

Fig. 4 Partial definition of perceptor interface

- **Execution layer:** that runs and/or stops a perceptor. It is composed of a collection of ICa methods that control the execution of a perceptor. The methods are:

[5] ICa transports are objects with ICa transport-methods to support the communications across ICa ORB. Communications are done through calls to those methods.

- Run( ): begins the execution, i.e. executes the ICa transport.
- Kill( ): stops the execution. The ICa transport no longer remains in execution.

Actuators share a basic structure consisting of:

- **Identification layer:** that identifies actuator. It has attributes that define the identity of an actuator such as name, self-number and description.
- **Interfacing layer:** that permits interaction with control agents. It constitutes the acquisition mechanism, and is made with an ICa transport to support the receives of final actions.
- **Acting layer:** that accomplishes operations for executing reconfiguration actions by means of algorithms. It constitutes the action mechanism of an actuator, and is made with specified-code written by implementers.
- **Execution layer:** that runs and/or stops actuator. It is composed of a collection of ICa methods that control the execution of an actuator. The methods are:
  - Run( ): begins the execution, i.e. executes the ICa transport.
  - Kill( ): stops the execution. The ICa transport no longer remains in execution.

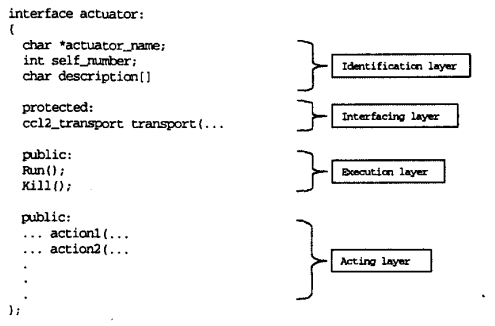

Fig. 5 Partial definition of actuator interface

Perception bases share a basic structure consisting of:

- **Identification layer:** that identifies perception base. It has attributes that define the identity of a perception base such as name, self-number and description.
- **Interaction layer:** that controls the interaction with perceptors and with other software-agents. It constitutes the communication mechanism, and is made with an ICa transport and defined methods to support the takes/puts/removes of variable data from/into/from the perception store. The communication methods are:
  - check(agent_id, message): checks if there is any available message for it in the transport queue. If there is one, it reads the message using the "get" method.
  - get(agent_id, message): reads the first available message for it from transport queue. After this, the message is deleted from the transport queue. The "agent_id" argument allows the perception base to return an answer message to the sender. Then, the message is processed. The receiver builds an answer with variable data using another message that is put in the transport to be relayed to the software-agent sender, in case of "take".
  - return(agent_id, message): returns an answer message with variable data to the software-agent identified with the "agent_id" argument.
- **Storing layer:** that constitutes the perception store. It is composed of a collection of objects that define the process variables. Each variable has attributes that set data and abstractions. Implementers fix them.
- **Execution layer:** that runs and/or stops perception base. It is composed of a collection of ICa methods that control the execution of a perception base. The methods are:
  - Run( ): begins the execution, i.e. executes the ICa transport.
  - Terminate( ): stops the execution. The ICa transport remains in execution.
  - Kill( ): stops the execution and removes all messages from the transport queue, i.e. the ICa transport no longer remains in execution.

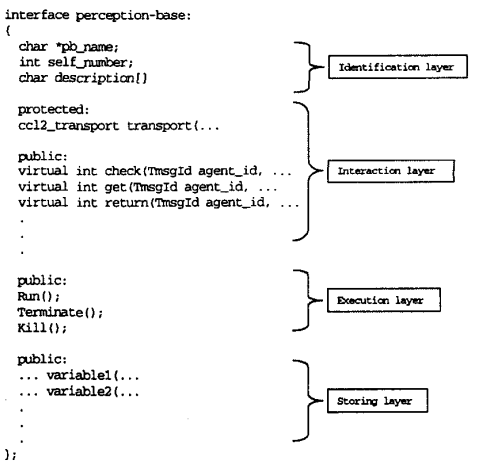

Fig. 6 Partial definition of perception base Interface

Software-agents share a basic structure consisting of:

- **Identification layer:** that identifies software-agent. It has attributes that define the identity of a software-agent such as name, self-number and description.

- **Communication layer:** that controls the communication with other A-ESS components. It constitutes the communication mechanism, and is made with an ICa transport and defined methods to support the requests/receives of information. The communication methods are:
  - send(agent_name, task, In_message, Out_message): sends a message to a specified software-agent and to a specified task and waits for the answer in the incoming message.
  - send(agent_name, task, message): sends a message to a specified software-agent and to a specified task and does not wait for the answer message.
  - check(component_id, message): checks if there is any available message for it in the transport queue. If there is one, the software-agent may decide to read or to remove the message using the "get" or "remove" methods respectively.
  - get(component_id, message): reads the first available message for it from the transport queue. After this, the message is deleted from the transport queue. The "component_id" argument allows receiver to return an answer message to the sender. Then, the message is processed. The receiver builds an answer using another message that is put in the transport to be relayed to the component sender.
  - remove( ): removes the first available message for it from the transport queue.
  - return(agent_id, Out_message): returns an answer message with partial results to the software-agent identified with the "agent_id" argument.
  - take(pb_name, data): takes variable data from a specified perception base.
  - put(pb_name, data): puts variable data into a specified perception base.
  - remove(pb_name, data): remove variable data from a specified perception base.
  - act(actuator_name, action): sends specified final actions to a specified actuator. Only control agents should use this method.
- **Task-solving layer:** that accomplishes the operations for solving assigned tasks by means of algorithms. It constitutes the reasoning mechanism, and is made with specified code written by implementers. Algorithms must be constructed with the most appropriate software technology, whether heuristic rules, procedural programming, fuzzy logic and so on, or hybrid.
- **Execution layer:** that runs and/or stops software-agent. It is composed of a collection of ICa methods that control the life of a software-agent. The methods are:
  - Run( ): begins the execution, i.e. executes the ICa transport.
  - Stop( ): stops the execution and maintains readiness for execution.
  - Terminate( ): stops the execution and terminates the tasks. The ICa transport remains in execution.
  - Kill( ): stops the execution and removes all messages from the transport queue, i.e. the ICa transport no longer remains in execution.

```
interface software-agent:
{
  char *agent_name;
  int self_number;
  char description[]

  protected:
  ccl2_transport transport(...

  public:
  virtual int send(char *agent_name, ...
  virtual int send(char *agent_name, ...
  virtual int check(TmsgId component_id,
  virtual int get(TmsgId component_id, .
  virtual int remove();
  .
  .
  .

  public:
  Run();
  Stop();
  Terminate();
  Kill();

  protected:
  ... task1(...
  ... task2(...
  .
  .
  .
);
```


Fig. 7 Partial definition of software-agent interface

5.3 Structure of A-ESS Applications

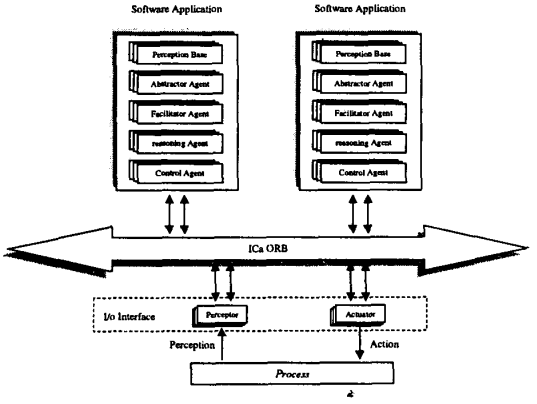

Fig. 8 Framework of A-ESS applications and interaction with process

An A-ESS application is a set of modules composed of one or more perceptors, one or more actuators, one or more perception bases, zero or more abstractor agents, zero or more facilitator agents, one or more reasoning agents and of one control agent. They might be in interaction and under co-ordination of the control agent. Perception bases and software-agents live in applications from which they are executed. Perceptors and actuators should become part of data acquisition

systems from which they are executed (see Fig. 8). Applications must be running before the A-ESS components can be executed. The applications could be distributed over a network of machines.

An A-ESS is intended to work on-line with a process. It works as a dynamical system according to a sampling period. Process variable measures are actualised (and saved into perception bases) every sampling time. Subsequently, software-agents are executed. They reason on the data in the perception bases and deduce outputs (manipulate perception bases or deduce partial results or final actions).

6. CONCLUSIONS AND FUTURE WORK

The A-ESS architecture has been defined and presented. It is based on concepts of software-agents. Theirs components (perceptors, actuators, perception bases, abstractor agents, facilitator agents, reasoning agents and control agents) and communication mechanisms are created on ICa (Intelligent Control architecture), a distributed-object-based software middle-ware to support the build-up of flexible and reusable distributed applications and services with software-agent features. A set of architecture features has been also briefly described. These features are modularity, solving focus, hierarchical structure, integration of heterogeneous components, sharing reasoning, distributed work, interfacing with data acquisition systems, interfacing with humans, reusability, software patterns, evolution/maintenance and openness. Those features should allow managing the A-ESS computational complexity and that make A-ESS much easier to understand, to build and to maintain.

The structures of the components and the interaction among them and with the process have been briefly described. However, none example of a real process has been presented because the A-ESS architecture is in the phase of design and prototype. Thus, the next step should be the implementation. The objective is to prototype tools without dependency of any process. So, the next step should be an advanced test using a well defined benchmark (for instance the COSY benchmark) as well as the increase in the complexity of process in order to detect drawbacks and correct them in the definition.

The other aims of this research is to provide:

- A method to apply the architecture, which should specify the conceptual construction of an A-ESS through different phases.
- Tools to support the architecture, which will be directed to assist the design and development of A-ESS applications from a point of view of the computer-aided control system design (CACSD).

ACKNOWLEDGMENTS

This research is supported by the CICYT project TAP96-1114-C03-03, "Plataformas Integradas de CAD de Supervisión y Metodologías", of the Spanish Government.

Special thanks to the Autonomous Systems Laboratory (ASLab) of the Universidad Politécnica de Madrid (Spain). The *eXiT* group has been authorised by them to use ICa in researches and developments.

REFERENCES

[1] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes", IFAC-13th Triennial World Congress, ref. 7f-01, San Francisco, USA, 1996.

[2] M. Wooldridge and N.R. Jennings, "Intelligent agents: theory and practice", Knowledge Engineering Review, vol. 10, 1995.

[3] S. Russell and P. Norving, Inteligencia Artificial, un Enfoque Moderno. Mexico. Prentice Hall S.A., 1996.

[4] A. De Antonio and M. Segarra, ICa, An Intelligent Control Architecture, Advanced User's Guide. Madrid. ASLab, Universidad Politécnica de Madrid, 1998.

[5] J.L. De La Rosa, Heuristic for Co-operation of Expert Systems, Application to Process Control, Ph.D. dissertation, Universitat Autónoma de Barcelona, Bellaterra, 1994.

[6] B. Moulin and B. Chaib-Draa, "An overview of distributed artificial intelligence" in Foundations of Distributed Artificial Intelligence, USA, John Wiley & Sons Inc., 1996.

[7] N.R. Jennings, E.H. Mamdani, J.M. Corera, I. Laresgoiti, F. Periollat, P. Skarek and L. Varga, "Using ARCHON to develop real-world DAI applications, part 1", IEEE expert, pp. 64-70 December 1996.

[8] R. Sanz, F. Matía, A. Jiménez, R. Galán, A. De Antonio and M. Segarra, "Heterogeneous software integration for intelligent process control: the HINT project", Valencia COSY Workshop, Valencia, Spain, 1996.

[9] O.C. Contreras and J.L. De La Rosa, "El enfoque orientado a agentes en el diseño de sistemas expertos aplicados en supervisión", Bulletí de l'ACIA, No. 12, Tardor 1997.

[10] B. Chaib-Draa, "Industrial applications of distributed AI" in Readings in Agents, chapter 2: applications, USA, Morgan Kaufmann Publisher, Inc., 1998.

[11] E. Rich and K. Knight, Artificial Intelligence. USA. McGraw Hill, Inc., 1994.

[12] G.P. Lekkas, N.M. Avouris and G.K. Papakonstantinou, "Development of distributed problem solving systems for dynamic environments", IEEE Transactions on Systems, Man and Cybernetics, vol. 25, No. 3 March 1995.

[13] G. Fiol-Roig and M. Ferrer-Gili, "Expert system for supervision of real-time control process", UIB report 1998.
[14] S. Vinoski, "CORBA: integrating diverse applications within distributed heterogeneous environments", IEEE Communications Magazine, vol. 14, No. 2 1997.
[15] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice. USA. Addison-Wesley, Inc., 1998.
[16] B.P. Douglass, Real-Time UML, Developing Efficient Objects for Embedded Systems. USA. Addison-Wesley Inc., 1998.