Pergamon

Control Eng. Practice, Vol. 4, No. 6 pp. 839-845, 1996

0967-0661/96 $15.00 + 0.00

PII:S0967-0661(96)00076-7

MULTIAGENT-BASED CONTROL SYSTEMS: A HYBRID[1] APPROACH TO DISTRIBUTED PROCESS CONTROL

J.R. Velasco*, J.C. González*, L. Magdalena* and C.A. Iglesias**

**Universidad Politécnica de Madrid - E.T.S.I. Telecomunicación, Ciudad Universitaria s/n., E-28040 Madrid, Spain (jvelasco@gsi.dit.upm.es)*

***Universidad de Valladolid - E.T.S.I. Telecomunicación, C/Real de Burgos s/n, E-47011 Valladolid, Spain*

(Received October 1995; in final form March 1996)

Abstract. This paper presents a general architecture and a platform developed to implement distributed applications as a set of cooperating intelligent agents. It also shows how this architecture has been used to implement a distributed control system for a complex process: the economic control of a fossil-fuel fired power plant. Agents in this application encapsulate different distributed hardware/software entities: neural and fuzzy controllers, a data-acquisition system, presentation manager, etc. These agents are defined in ADL (Agent Description Language), 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 deals with a way of approaching a 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 control model, use the mechanisms of message-passing for communication purposes, and are usually implemented from an object-oriented perspective.

The multiagent architecture developed here can be used to implement general distributed applications, not just a distributed control system. In this general framework, several software elements (the agents) cooperate with each other to reach their own goals. The system designer has to decide on the set of agents to 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 (see below and (González, *et al.*, 1995)). The problem of how to interchange data between agents is solved by using a common knowledge-representation language.

As an example of how to apply this architecture to distributed control, an actual system is shown: a fossil-fuel fired power plant. In particular, the goal is to achieve strategic (not tactical) control: the system has to reduce the heat rate (the ratio *fuel/generated power*), suggesting appropriate set points for automatic controllers or human opera-

[1] 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.*

tors.

At present, two versions (distributed and non-distributed) of a control system for a real power plant sited in Palencia (Spain) (García, *et al.*, 1993) are being implemented. This paper focuses principally on the distributed one.

2. DESCRIPTION OF AGENTS

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

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

Every agent is composed of a control block, a database (including a model of the agent, a model of the environment, the state of the agent, private objects and global data), and a communication block (the network communications model and a mailbox) (see Fig. 1).

Any agent may include *goals* (that is, processes which start when the agent is "born"), *services offered* (the agent offers a set of services to the rest of the agents, and these services may be executed in a concurrent – as an independent process – or non-concurrent way) and *services required* (a list with the names of the services that this agent may need).

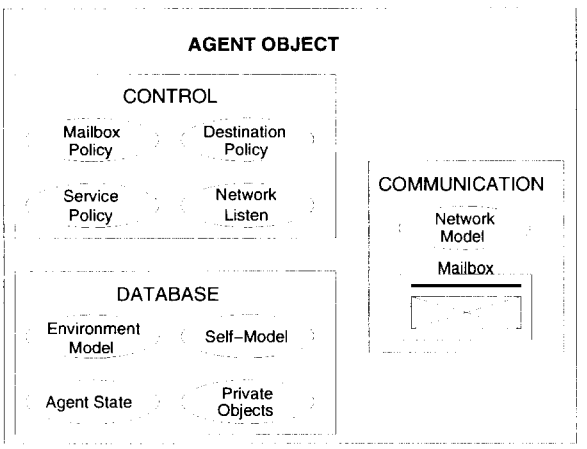

Fig. 1. Model of a multiservice agent

One of the major features of these agents is that their services (if concurrent) are executed as separate 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. THE MIX MULTIAGENT PLATFORM

The MIX multiagent platform defines a layered model of the network (Iglesias, *et al.*, 1996) that provides agents with a uniform view of the net. This model distinguishes two kinds of agents: network agents, which offer services for maintaining the network, and application agents, which address a particular problem. A network agent, called YP ("yellow pages"), provides different network facilities for registering network addresses, services offered, services an agent is interested in, etc. YP selectively gives information on the dynamic changes of the society of agents. There is also the possibility of communication with groups of agents, which can be configured dynamically. In this way, a service petition can be performed with point-to-point communication or with point-to-multi-point communication (in a broadcasting or multi-casting mode).

YP agents continuously update the information needed by their registered agents. Therefore, application agents can establish direct communication links among themselves, so avoiding collapse due to YP saturation or network failures. If any YP agent disappears, registered agents will not receive new information. But they can still work with their known network and services model.

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

At present, the MIX platform (González, *et al.*, 1995) is made up of four elements:

- MSM (Multiagent System Model) C++ library, with the low-level functionality of the platform. This is a modified version of the work carried out by Domínguez, (1992).
- ADL translator. The Agent Description Language has been designed to specify agents. ADL files gather descriptions of agents, and the translator generates C++ files and the appropriate makefile to obtain executables.
- CKRL ToolBox. A reduced version of *Common Knowledge Representation Language*, by the MLT consortium (Causse, *et al.*, 1992), 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.

[1] The platform allows the use of any other language for intercommunication between processes. In this way, KIF (Knowledge Interchange Format) (Genesereth and Fikes, 1992), another widely used language, is being considered as the second native language of the platform.

- Standard ADL agent definitions and CKRL ontologies.

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

A fossil-fuel fired power plant is a very complex process with a large number of variables which operators can actuate. The objective of this control system is to reduce the fuel consumption while generated power is kept constant. The first problem is that no reliable model of the process exists; so the system needs to learn how the power plant works. The second problem is that the quality of fuel 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 limited homogenization of the last hour's fuel, so the coal quality changes with a smooth curve. This coal quality is used for the *heat rate* calculation, that is the optimization variable.

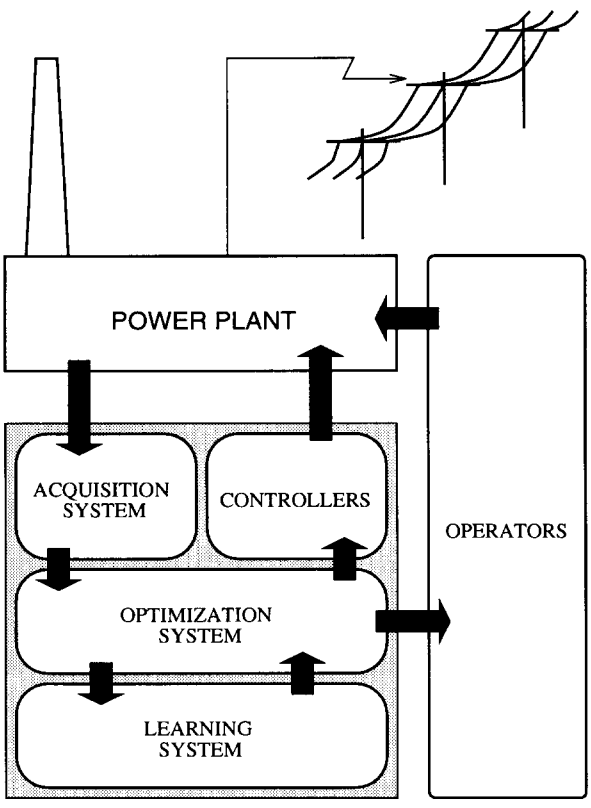

Fig. 2. Application diagram

This last problem implies that the control system can only have access 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 for the heat rate:

(1) *Principal air flow to the boiler*: This air flow carries the coal powder from the mills to the boiler. So if this variable decreases, the fuel consumption decreases, whatever the coal quality.

(2) *Boiler output gas temperature*: A commonsense analysis says that a low temperature at the output of the boiler is better than a high one. If the temperature is high, 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 various performance tests in the power plant, one of these variables will be selected as the objective.

In order to obtain good-quality values for the control variables, a data-acquisition system filters the signals that reach the control system from sensors. The acquisition module receives 200 variables, and produces 23 to the optimization module. These 23 variables are known as the *context vector*. The optimization module will make suggestions over 11 operation variables (the so called *operation vector*) to the controllers or operators. The acquisition/filtering module is a very important part of the whole system: reliable inputs are even more urgently needed than 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 the known states is divided into several large areas (called macrostates). These macrostates can be defined by experts (Velasco, *et al.*, 1992), or computed using fuzzy clustering techniques (Velasco and Ventero, 1994) or a neural network. In this case, the second approach is used.

The control system has as many rule bases as macrostates. To create the fuzzy knowledge bases, a modified version of the C4.5 algorithm (Quinlan, 1993) was used. This modification creates fuzzy rules from sample data files: to make the C4.5 function learn the system must provide it with 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 thereby to classify the actions in the operation vector as bad, indifferent or good. After this classification, the algorithm creates fuzzy control rules.

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

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

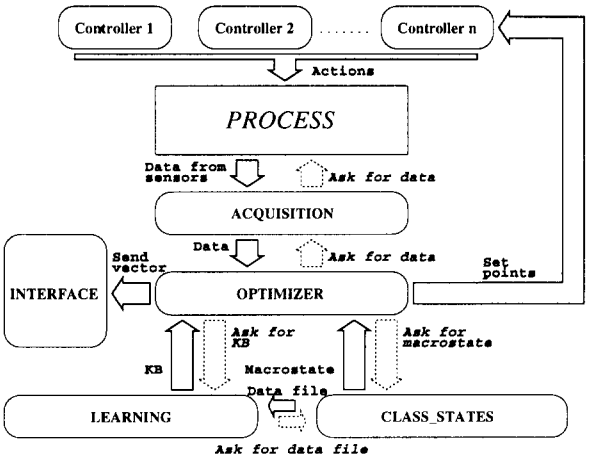

Fig. 3. Description of agents

For the design of this application using the MIX platform, this distributed control system has to be seen as a set of agents with their respective goals and services, communicating among themselves through the exchange of 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.

The *Acquisition* agent obtains data from process sensors periodically, and gives context vectors to the *Optimizer* upon demand. The *Optimizer* agent asks the *Class_state* agent for the appropriate macrostate, and will use the corresponding Knowledge Base(s) to obtain the operation vector. The values of the variables of this vector will be sent to specific *Controllers* as set points, or will be shown to operators for a manual adjustment. The *Optimizer* agent will ask the *Learning* system for a new KB if it sees that the cost value (the indirect heat rate) is increasing.

The MIX architecture uses ADL 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 on different computers. The complete ADL file for this application is shown in Appendix A. In this section just the agent definition process is presented and it is focused on the *Optimizer* agent and the *Learning* agent.

The *Optimizer* agent has as its own 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 type-face lines show service petitions that will be requested from different specialized agents: the *Acquisition* agent gives the context vector, the *Learning* agent creates new KBs, the *Class_states* agent classifies the context vector and each *Controller* tries to adjust the different set points.

However, at the design level, the agent description only needs to know the name of the services required (it does not have to know which agents will be available to perform them), the names of the functions that implement the services and goal, and the C++ file where these 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
```

optimizer.C and *learning.C* are C++ files where service and goal functions are defined. At design time, only function names and input/output data are needed. Of course, programmers must write the C++ code according to this specification. The

"Makefile" generated by the MIX platform links source files and libraries to construct agents as independent executable files.

In the *Learning* agent, classes 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. Programmers need to know only the CKRL specification to be able to manage input and output messages. The complete CKRL file for this example is shown in Appendix B.

6. REAL-TIME ISSUES

Restrictions on space in this paper prevent the presentation of further details of other interesting aspects of the MIX platform, in particular those relevant to real-time applications. This kind of critical application was always borne in mind during the design phase of the platform. Therefore performance, efficiency in the use of resources, fault tolerance, flexibility, etc. were key design criteria. In particular, some architectural decisions and built-in mechanisms that are useful for real-time systems are briefly addressed:

- Network model
  In the MIX network model, communications are established directly between the agents that demand and provide a service, without intermediaries (routers, or facilitators in other architectures). Besides, agents have complete and continuously updated knowledge on net addresses of the agents providing the services that they may require. Therefore, there is no need to find out these addresses (supplied by specialized agents, sometimes called *traders*).
- Concurrency
  As pointed out before, services can be executed in a *concurrent* or *non-concurrent* way. The first method implies starting a new process for carrying out the task. So, it allows the agent to continue its internal working (taking care of new incoming messages and executing new services). The second method blocks the agent, stopping its control loop, as a way of ensuring perfect control over the global agent activity, or to improve efficiency.
- Loose/tight coupling
  A salient feature of the MIX architecture is the possibility of incorporating two levels of integration. Agents are, by default, loosely coupled. This means that inter-agent communication is carried out via message passing. However, a tighter coupling mechanism is often needed (mainly for the sake of efficiency). This happens when there is a continuous flow of interaction among agents. MIX-ADL permits users to specify that a group of agents has to be treated as a strongly coupled society. Only part of the services offered by the society as a whole are exported (known from the outside). The remaining - internal - services are offered only to the agents in the society. Petitions for these internal services are executed by the platform as function calls, instead of using message passing. However, the concrete method used for service handling is kept hidden from the user.
- Contracting policies
  The architecture allows the integration of different protocols between the agent demanding a service and candidate providers: one of these protocols is Contract Net (Smith, 1988). In order to use this protocol it is necessary that services have an associated cost function. In this way, potential providers evaluate this function and send back the result to the agent that initiated the protocol. Cost functions may involve different criteria and, therefore, have different interpretations: price to be charged to the petitioner, estimation of the error made when performing the service, estimation of complexity, resources consumed (e.g., completion time), etc. The agent that initiated the protocol decides, by analyzing the results received, which agent (or agents) is (are) awarded the contract. This protocol can be combined with the establishment of constraints on response times. For instance, time-outs may be applied, after which no more answers to a service petition will be taken into account.
- Synchronization
  Three synchronization modes can be used for communication purposes: synchronous, asynchronous and deferred. In the first one, the sender remains completely blocked until the peer agent replies. This situation does not occur in asynchronous mode, in which the sender is able to continue executing other processes. Deferred mode can be used when the sender does not require an immediate answer. In this case, it can continue carrying out the same task until the time the response is required (synchronization point).

7. CONCLUSIONS

Multiagent systems are proposed as an effective 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-fuel fired power plant. Although full evaluation of the system has not yet been completed, some preliminary conclusions can be advanced. In comparison to the conventional (cen-

tralized) architecture previously used, the distributed solution shows evident advantages:

- Interfaces are simpler, thus speeding up the development phase of the system's life-cycle.
- Systems are more reliable in terms of fault-tolerance and protection against noise.
- Control is more versatile, in the sense that this approach facilitates the simultaneous use of several controllers based on different techniques (each with its own errors and response time depending on the problem state). As a consequence, if the system has several controllers to perform a particular action or process, the error estimation received can be used to improve system accuracy by selecting the best controller.
- The MIX architecture is specially suitable for real-time applications due to the architectural decisions and built-in mechanisms explained in the previous section.

8. ACKNOWLEDGEMENTS

This work would have never been done without the experience accumulated over the years and the tools developed by Mercedes Garijo and Tomás Domínguez in their Multiagent System Model, the basis for the MIX agent model. The authors are also indebted to Jaime Alvarez and Andrés Escobero (from their own group) and Marc Vuilleumier (from Université de Genève, Switzerland) for their contribution to the implementation of the platform.

9. REFERENCES

Causse, K., M. Csernel and J. Kietz (1992). Final specifications of the Common Knowledge Representation Language (CKRL) of the ML-Toolbox. Deliverable D2.2. MLT Consortium, ESPRIT project 2154.

Domínguez, T. (1992). Definición de un modelo concurrente orientado a objetos para sistemas multiagente. PhD thesis. E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid. (In Spanish).

García, J., J.R. Velasco, J.A. Castineira and J. Martín (1993). CORAGE: Control por razonamiento aproximado y algoritmos genticos. propuesta de proyecto. Technical report. UITESA, DIT-UPM, IBERDROLA, Grupo APEX. (In Spanish).

Genesereth, M. and R. E. Fikes. (1992). Knowledge Interchange Format, version 3.0. Reference manual. Technical report. Computer Science Department, Stanford University.

González, J. C., J. R. Velasco, C. A. Iglesias, J. Alvarez and A. Escobero (1995). A multiagent architecture for symbolic-connectionist integration. Deliverable D1. MIX Consortium, ESPRIT project 9119.

Iglesias, C. A., J. C. González and J. R. Velasco (1996). MIX: A general purpose multiagent architecture. In: *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95).* Wooldridge, M., Müller, J. P. and Tambe, M., (Eds.) Lecture Notes in Artificial Intelligence. Springer-Verlag.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann. San Mateo, CA, USA.

Smith, R. G. (1988). The contract net protocol: High-level communication and control in a distributed problem solver. pp. 357–366. In: *Readings in Distributed Artificial Intelligence.* Bond, Alan H. and Gasser, Les, (Eds.). Morgan Kaufmann. San Mateo, CA, USA.

Velasco, J.R. and F.E. Ventero (1994). Some applications of fuzzy clustering to fuzzy control systems. In: *3rd Int. Conf. on Fuzzy Theory and Technology.* Durham, NC, USA.

Velasco, J.R., G. Fernández and L. Magdalena (1992). Inductive learning applied to fossil power plants control optimization. In: *Control and Power Plants and Power Systems.* E. Welfondera, G.K. Lausterer and H. Weber, (Eds.). pp. 205–210. Number 9. *IFAC Simposia series.* Pergamon Press. Oxford, UK.

Appendix A. ADL FILE

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

AGENT YP_Agent -> YPAgent
END YP_Agent

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

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

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

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

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

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
```

Appendix B. CKRL

```
defsort intpos range (integer (1:*));
defproperty class_number sortref intpos;
defconcept Class
  relevant class_number;

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;

defsort point range (real (0.0:1.0));
defproperty pointdata sortref point;
defconcept Point
  relevant pointdata;

defsort regla_s range string;
defproperty regla_p sortref regla_s
defconcept Rules
  relevant regla_p;
```