A "Message Oriented Middleware" approach to HYSYS extensibility

R. Rallo, J. Ferré-Giné, A. Arenas and Francesc Giralt[#]
Departament dEnginyeria Informàtica i Matemàtiques, ETSE
[#]Departament d'Enginyeria Química, ETSEQ
Universitat Rovira i Virgili
Ctra. De Salou, s/n
43001 Tarragona, Catalunya, SPAIN

Abstract and Scope of the Work

The present work is part of a research project that is currently in progress at Tarragona with the main objective to develop an open and distributed architecture capable of supporting applications of collaborative intelligent techniques to control chemical processing plants. In this paper new ways to integrate external components with the HYSYS Process Simulator are explored, so that applications designed by developers can be distributed across TCP/IP based networks. The registration of applications with components generated by HYSYS requires the development of a module that links the process simulator with an external "Message Bus". To validate the proposed framework or architecture an intelligent control application based on Radial Basis Functions (RBF) neural networks has been conceived and implemented. This neuro-monitoring application based on Kohonen's self-organizing maps (SOM) registers with a dynamic simulation running on HYSYS and uses the simulated output to learn the dynamics of the process. A third party library for the construction of intelligent agent systems is also used to demonstrate that the proposed framework can be easily integrated with other systems.

Keywords: neural networks, process monitoring, java, CORBA, DDE, intelligent control, Kohonen maps.

1. Introduction and Specific Objectives

The goals of the process engineer when defining the operation conditions of a chemical plant are usually directed towards pursuing a better yield, looking for better quality products, a higher production rate, or an improved profit. Often, small variations in some process conditions produce important changes on the whole dynamics of a chemical plant. In this respect it would be very useful to develop tools to visualize the variables involved and the state of the process, as well as to explore and learn its evolution.

The amount of data available in real-time (and at high sampling rates) about a chemical process grows exponentially with factory automation. Usually, chemical plants incorporate a large amount of sensors and field measurement devices (often ranging from hundreds to thousands) acting as sources of huge data sets. Also, the popularization of the concept of datawarehouse makes possible the storage of large historical data sets ready to be processed for knowledge extraction. This implies that the old problem of having enough data records for gaining a better and deeper understanding about the dynamics of a given process shifts to a new one in which the main difficulty is how to exploit (use) the knowledge contained on these data files.

Is under this situation that soft computing techniques, like neural networks and other biologically inspired systems, may help engineers in the task of automatic (or human guided)

knowledge extraction from databases. This leads to the emergent concept of data mining, which is embedded on the more general category of “knowledge discovery from databases (KDD)” techniques, as the adequate methodology to perform these tasks.

Simultaneously, the fast development of high-speed and reliable communication technologies and infrastructures causes that many companies involved in process re-engineering activities also consider the objective of moving from their legacy applications to distributed applications running on “open systems”. As a result of the above changes, usually a mixture of legacy, client/server and intranet/internet applications forms the core of corporate information systems. At the same time and mainly due to the grow and expansion of INTERNET, there has been important developments related to open connectivity and application interoperability, which resulted in "independent programming languages" such as JAVA and open communication technologies such are CORBA and XML. On the other side, today’s corporate information technology (IT) policies require that data from factory floor systems feed enterprise applications and to implement control strategies on “open” systems. As a consequence, there has been a boom in the use and implementation of open distributed applications.

It should be kept in mind that Chemical Manufacturers are one of the sectors with the highest heterogeneous computing resources and applications. The integration and interoperability of all these systems (see Fig. 1) is a difficult and expensive task. Advanced Process Control (APC) applications are a paradigmatic example of that situation. As process control technology moves towards artificial intelligence and soft computing areas, the emerging applications need more and more resources. As a consequence, concepts such are real-time collaboration, open connectivity, transparent data exchange and mobility are gaining momentum. One of the key components of APC applications are process simulation systems, because the “open” and “distributed” access to simulation data will play an important role in the new corporate IT environment.

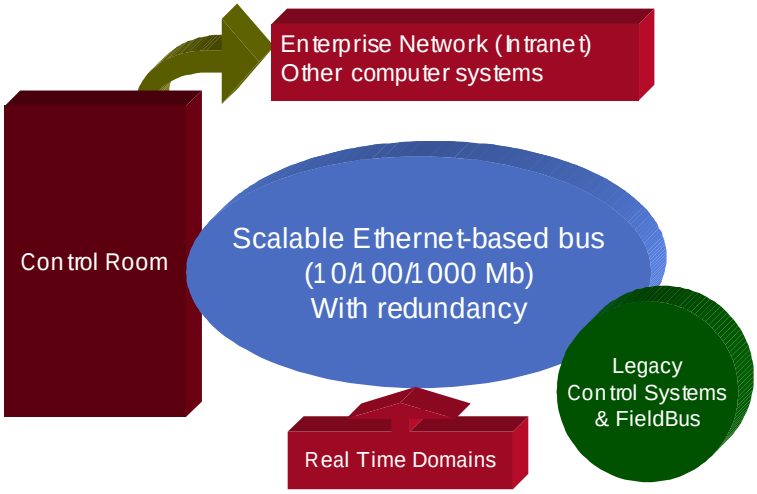

Figure 1. Integration of new APC applications and corporate IT systems

The specific objective of the present work is the development of a core architecture and a basic programming framework for achieving an "open and distributed" access to HYSYS to integrate intelligent control or supervision systems with simulations. One way of achieving this goal is by using the Java technology platform, because it offers a solution to meet the needs of enterprise IT with low disruption on the existing factory environment. The proposed architecture will also allow the implementation of the aforementioned intelligent systems within factory communication and information systems.

2. Distributed Architecture

2.1. Access to HYSYS objects

APC applications require frequent data transfers among them and, in the specific problem of the current work, with the simulations running on HYSYS. The better known and frequently used techniques applied to this end are based on OLE Automation with Visual Basic as programming language. The main drawback of this approach resides in the lack of portability of the code.

The first step towards the design and implementation of an open interface to the HYSYS simulator is to expose its internal objects to java applications. This could be done by using the Java Native Interface (JNI) framework since it allows java codes running in a Java Virtual machine (JVM) to access native libraries written in other languages (C, C++ or assembly) as if they were native java objects. The process of embedding native codes into java applications could be done in two different ways: the first one involves the access to the native source code and its header files for its processing with JNI tools to create the appropriate java wrappers; the second is based upon the use of a Java – COM bridge to access the native objects. In this second model the bridge talks to COM objects using distributed COM (DCOM) layered over Remote Procedure Calls (RPC), which in turn will reside on top of the TCP/IP protocol. This second approach has been selected here because it is appropriate for users that will not have direct access to the HYSYS source code. There are several commercial and open source applications that perform the abovementioned bridging process; the LINAR's *J-Integra* or IBM's *Bridge2Java*. For validation purposes the later has also been used in the example application described in the section.

IBM's *Bridge2Java* allows the creation of Java objects through a typelib corresponding to a COM enabled application. Once created these wrappers can be used to access methods and properties of COM objects. Using these wrapper classes we have implemented a server capable of talking to the HYSYS simulator (see Figure 2).

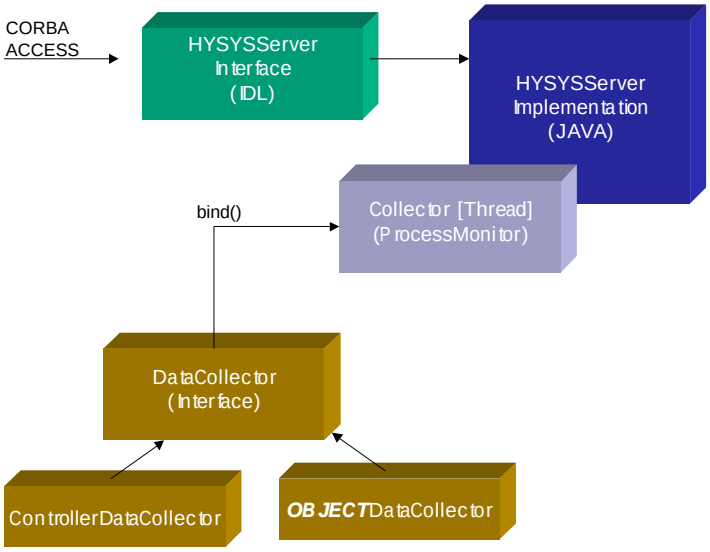

Figure 2. HYSYS Server Architecture

Since in the current development the HYSYS server runs as a local application on the same machine that is running the HYSYS simulator, it is necessary to provide remote access to

applications interacting with the simulator. This could be achieved with distributed object technologies. In the current case either the Java Remote Method Invocation (RMI) or the Common Request Broker Architecture (CORBA) can be used for accessing remote objects as the server has been implemented using the Java programming language. CORBA has been chosen as distributed object framework because with the RMI method only java objects are allowed to interact and generality would be lost.

The next step is how to transfer data back and forth from the simulator efficiently. To achieve this bidirectional data transfer two different communication models can be adopted. The first is based on the Dynamic Data Exchange protocol (DDE); HYSYS acts as a DDE client through the DCS interface. For this model a DDE server has been designed and implemented. The server is based on Neva Object Technology Inc. classes that implement the DDE protocol using JNI; the server acts as a daemon that waits for DDE transactions and performs the requested process through a *transaction Handler* interface, as shown in Figure 3.

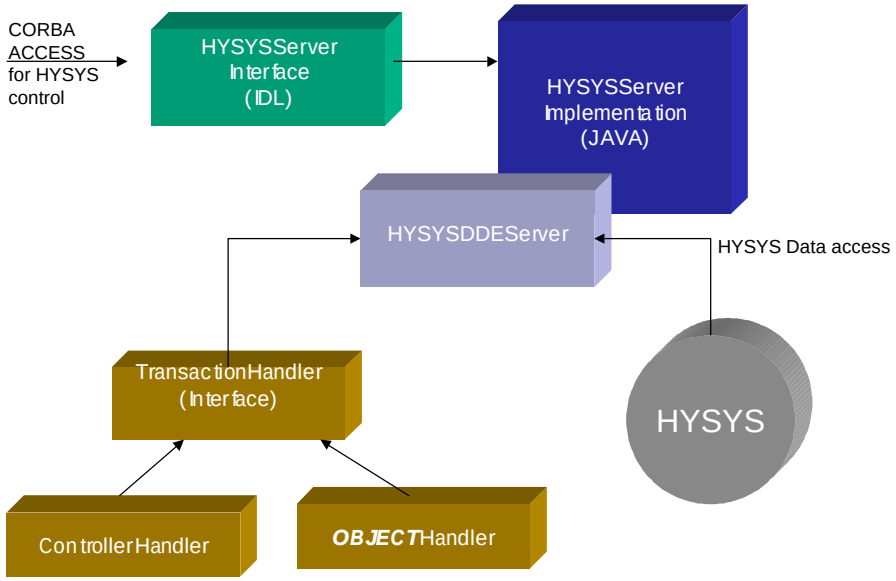

Figure 3. DDE based HYSYS server

The server supports both, tag-based and array-based clients. Bi-directional data communications are achieved using either POKE or REQUEST transactions.

The second communication model is based on CORBA method invocation (see Figure 4). Since we have a set of wrapper classes that encapsulate the behavior of each HYSYS object, the HYSYS server is capable of reading the values for each property of a HYSYS object.

Both approaches require the start of a polling process from the server side, making thus necessary the setting of a polling interval. Nevertheless, this polling policy could request values of properties that have not changed, consuming an unnecessary amount of CPU cycles. On the CORBA based model this effect can be reduced by using the Process Data Table object of HYSYS and attaching to it an event handler that activates the reading of an object's property only when its value has changed.

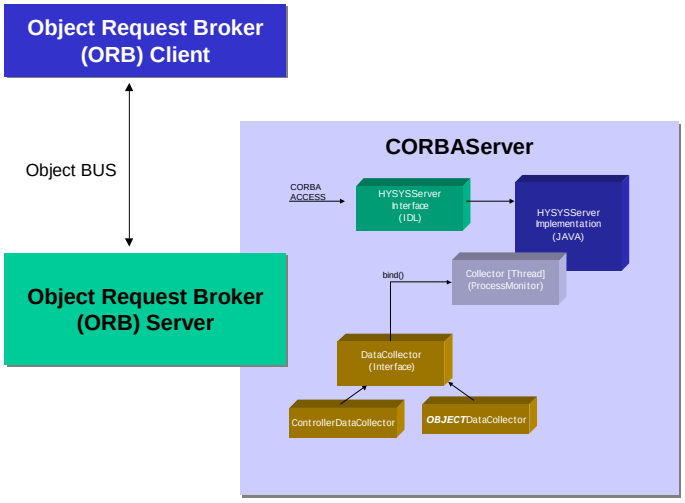

Figure 4. CORBA based HYSYS Server

2.2. Distributed access to HYSYS data

Now, that the primary goal of open access to HYSYS has been achieved by the DDE based or CORBA based HYSYS server design it is necessary to provide access to HYSYS data from remote workstations. The main purpose of this access is to send new values of any variable to all the external applications that are observing it. To perform this task a flexible, fast and reliable architecture for distributed event notification is needed. In this case a Message Oriented Middleware (MOM) approach is used.

Message Oriented Middleware (MOM) is one of the categories of connectivity middleware that provide program-to-program communications by message passing. MOM generally supports multiple protocols and, thus, will support reliable and scalable high-performance distributed application networks. Most Message Oriented Middleware are implemented with queued message store-and-forward capability, which is Message Queuing Middleware (MQM). Usually MOM applications support two modes of operation: The Point-to-Point (producer/consumer) and Publish/Subscribe (broadcast). In the first mode, applications send and retrieve messages from named message queues. In the second, applications publish messages to named channels and listen asynchronously for arriving messages on selected channels. There are many commercial and open source products that provide this functionality. A group of companies that are producing MOM products have released a joint specification to define a uniform interface from Java to MOM: The Java Message Service (JMS).

In the current framework the publish/subscription mode, with the ANTS API that exposes an easy interface to MOM systems, is used. In addition, the ELVIN system has been used as MOM infrastructure.

2.3. Framework proposal

As a starting point for the design of the present framework a layered model (see Figure 5) will be adopted to represent information systems in a chemical plant.

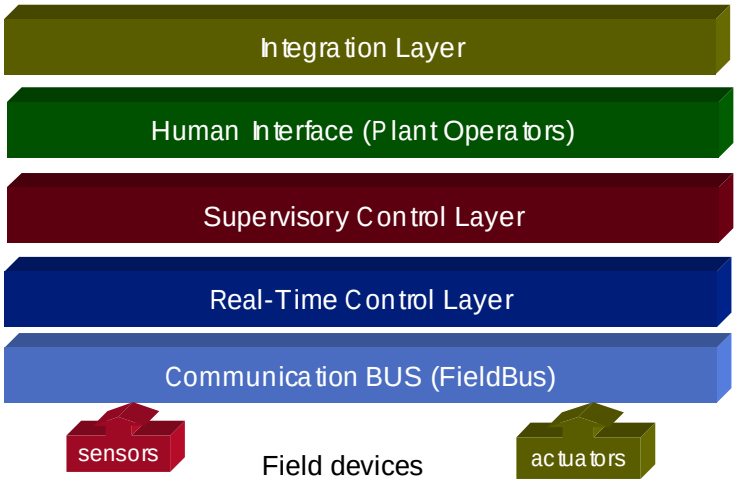

Figure 5. Layered Model of Information Systems in Chemical Plants

The main objective is to design a robust and generic framework for interfacing each of these layers. Dynamic simulators, such as HYSYS, play an important role in this information systems architecture because they can interact with low level layers (real-time control) as well as with higher-level layers (supervisory control and operator interface). This is the reason why the current framework for open and distributed access to HYSYS is the starting point for the design of a more generic one.

The proposed framework shown in Figure 6 is based on two key components: the HYSYS Server and the Message Bus.

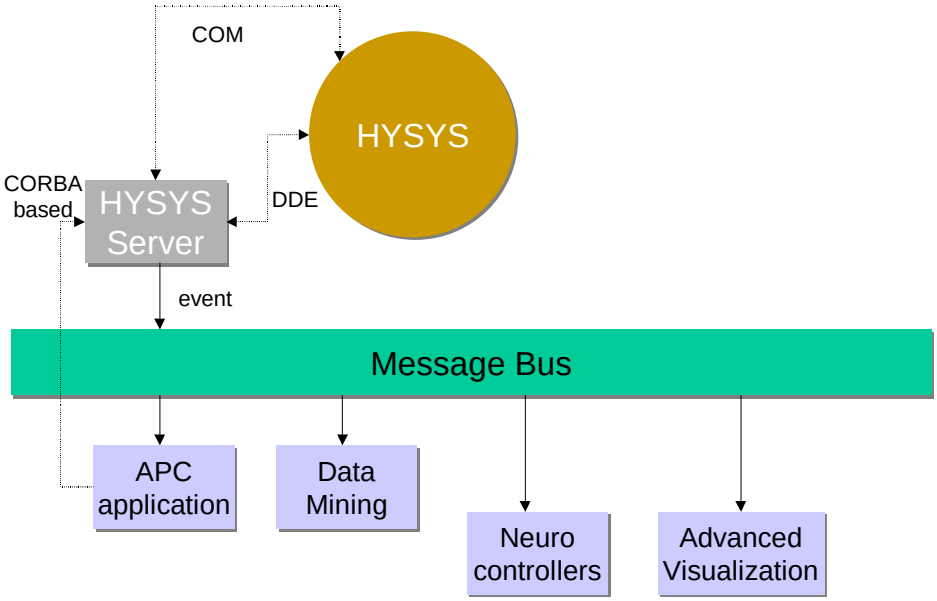

Figure 6. Schema of the proposed framework.

The HYSYS Server should interacts with the HYSYS simulator through a Java-COM bridge, by using DDE or a combination of both techniques. The external control of the simulator is carried out with CORBA, so that APC applications can control the simulation without the need of any external interaction with HYSYS. Simultaneously, registered *Data Collectors* or *DDE Transaction Handlers*, depending on the model used, acquire the simulation data at each time step (or polling interval) and publish them to the message bus. This allows applications interested on selected kinds of events to receive the data. Also, since all applications that

interact with the message use the publish and subscribe model the process of data exchange between these applications is easily implemented.

3. **Validation of the proposed architecture.**

3.1. Problem Statement

Two problems have been considered to evaluate the "usability" of our approach and validate the framework or architecture. The first is the monitoring (and visualization) of a simulated chemical process by means of Kohonen's self-organizing maps. The second is the training of a neurocontroller based on HYSYS PID controller implementation.

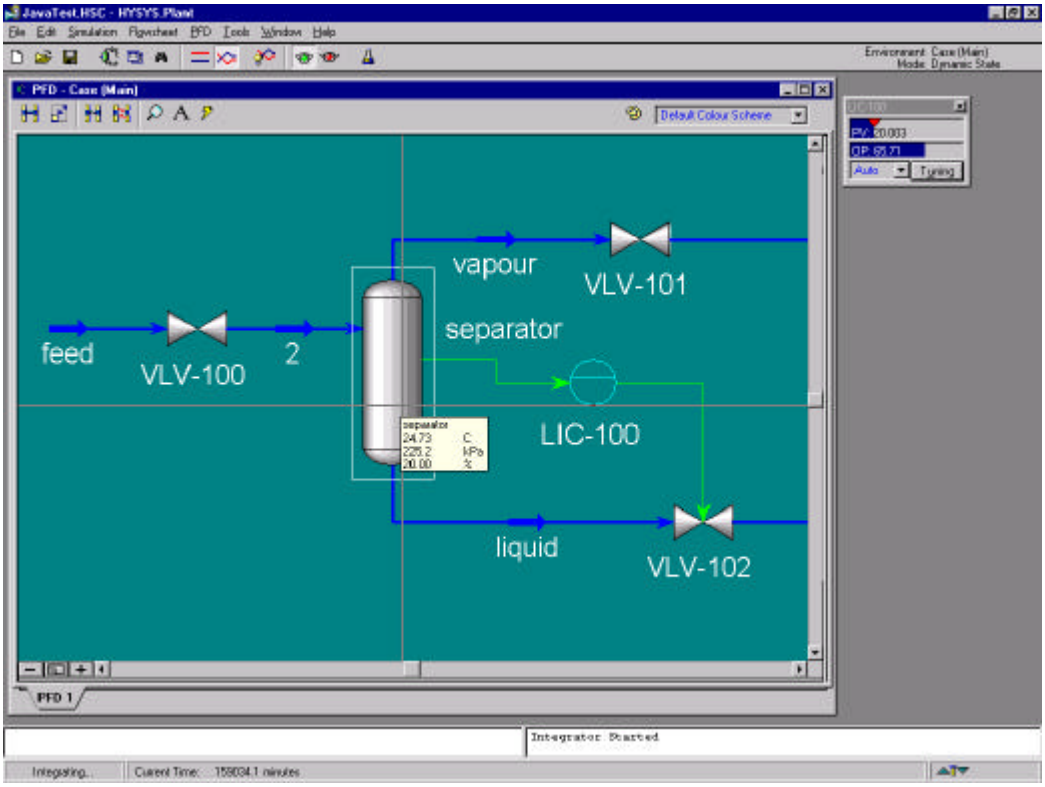

Figure 7. Example of the dynamic simulation of a two-phase separator.

The test case selected, shown in Figure 7, is one of the simple separation examples taken from HYSYS. It consists of a two-phase separator and a level controller measuring the liquid level of the tank. The separator is feeded with a mixture (50%) of ethane and n-C20 at a temperature of 25ºC and a pressure of 300 kPa.

3.2. Example of Monitoring

The main difficulty that a process engineer faces is the extremely high dimension of the state space that characterizes the operation of a chemical plant. The usual solution to this problem resides in the design and construction of more (or less) "ergonomic" control panels showing a complete point-of-view of process data. Nevertheless, the problems inherent to visually monitoring a large amount of data as well as establishing their relationships remain.

A Self-Organizing Map (SOM) performs a topology preserving mapping from the high dimensional input space (the state space of the process being analyzed) onto map units so that

relative distances between data points are preserved and, thus, approximating the density function of the input space in an ordered manner. In this way, data patterns lying near each other in the input space will be mapped onto nearby map units, hence serving as a clustering tool. Furthermore, because a well-constructed SOM will have good generalization capabilities, it can be used also as a modeling tool.

The application of the SOM algorithm to process analysis and visualization is usually carried out in four steps:

- Data preprocessing
- Map training
- Map validation and interpretation
- Visualization

When using the SOM for process analysis we several approaches can be adopted (Vesanto, 1999): (i) Analysis of structure and shape of the map, (ii) analysis of the codebook (or prototype) vectors, and (iii) applying the map to the analysis of new data, i.e., data never seen before by the neural network.

For the monitoring problem a set of variables representing the dynamics of the separation process in Figure 7 have selected. During the training stage of the self-organizing several disturbances to the whole dynamics of the system have been introduced to present to the neural system inputs located at different points of the separator's state space. After a self-organization process using a "winner takes all" learning procedure (see Appendix), the map is capable of classifying new inputs and associating them to a class over the map surface. By using this procedure a dimensional reduction is achieved and a high dimensional input can be mapped (in a non-linear fashion) into a two dimensional point. Furthermore, a set of successive time steps can be represented as a trajectory over the map surface and, thus, the dynamic behavior of the process can be easily pictured.

We have developed a clusterer implementing algorithm for the Kohonen's self organizing map. Once implemented, the clustering algorithm has been embedded on a client application that registers with the message BUS and receives from the HYSYS server the data from the dynamic simulation. Using these data the map self-organizes in order to create a two-dimensional projection of the separation process given in Figure 7. Once trained, the values of selected process variables at each time step of the simulation could be represented over the map, allowing plant operators to have an easily understandable image of the whole process dynamics, without the need to monitor each process variable individually.

The main results concerning this validation phase will be presented during the conference.

3.3. Integration of third-party applications

The purpose of this validation test is twofold. First, To test further the integration capabilities of the porposed framework and second to test the capabilities of a Radial basis Function (RBF) to mimic the behavior of a HYSYS PID controller.

ABLE, the IBM's Agent Building and Learning Environment, has been selected as the third-party application. This is a public domain environment designed to build leaning agent systems. For this test case a java component (JavaBean) that is capable of receiving the data published on the message bus has been developed.

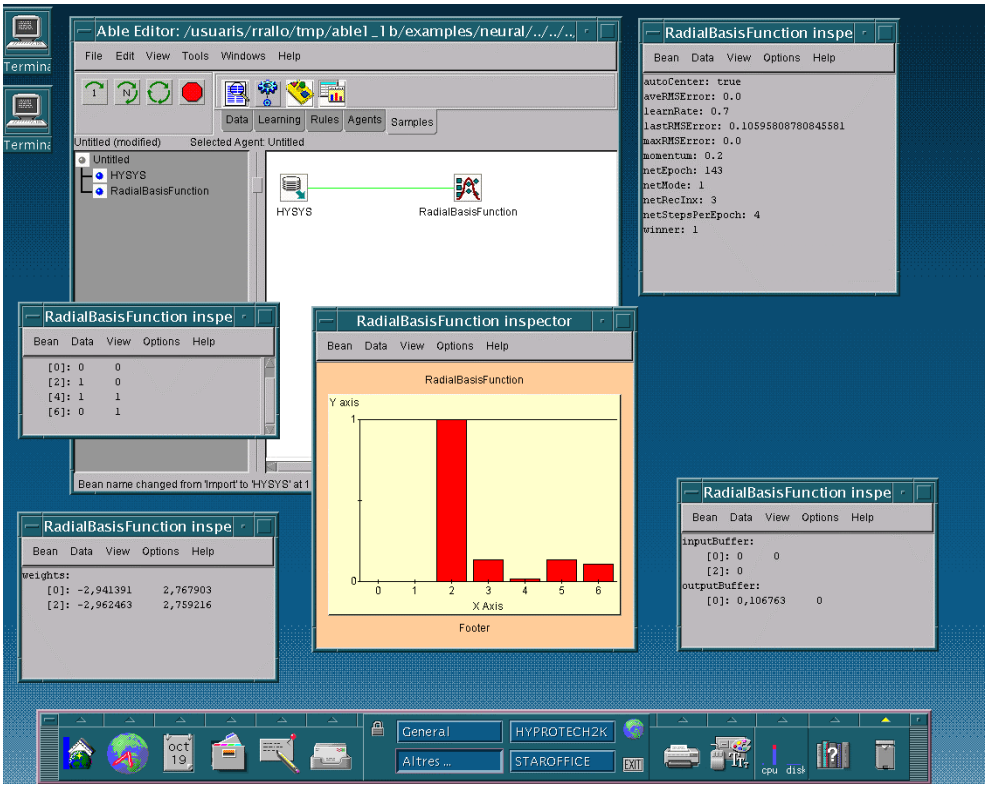

Figure 8. Sample application using ABLE Radial basis Functions

The training procedure of the neural network has been attained as follows. The main parameters of the PID controller, the set point (SP), gain, control action (% of valve aperture) and actual target value (% of liquid in the tanc), are published by the HYSYS Server. A RBF network is trained to mimic the behavior of the controller, so that when the SP is changed the trained network drives the valve aperture based on the actual liquid volume of the tank in Figure 7 towards the desired SP.

3.4. Work in progress

The first goal of developing and open and distributed access to HYSYS has been reached. The developed framework is capable of allowing any java or CORBA enabled application to interact with a simulation running on HYSYS. Also, as we have achieved a distributed access to the simulator all the data can be accessible through Internet based services, such the World Wide Web, with easily created web based interfaces to the simulator.

The work in progress comprises the development of a set of intelligent control agents that will be capable of fully interacting, within the proposed communication architecture, not only with the HYSYS simulator but also with any device capable of publish its data on the message bus. This approach will allow to advance towards **intelligent collaborative control schemes** that will facilitate the automatic, real time optimization of the global plant operation based on high level constrains, such are economic parameters, business rules, or environmental regulations, that can be published in real time to the message bus.

Acknowledgments

The work was supported by the "Dirección General de Investigación Cinetífica y Técnica" (Spain), DGICYT project no. PB96-1011, and the "Programa de Grups de Recerca Consolidats de la Generalitat de Catalunya", CIRIT project no. 1998SGR-00102.

Selected references

García, P., Skarmeta, A., Koch T. (2000) A generic collaboration bus for BSCW. *Proceedings of Intelligent Systems and Applications, Symposium of Interactive and Collaborative computing*.

IBM Alphaworks. Bridge2Java. http://alphaworks.ibm.com

Kennedy, J.P. (1999) Next Generation Plant Information Systems: Prepare for a web-centric architecture. *Hydrocarbon Processing*, April, 95-100.

Kohonen, T. (1990) The Self-Organizing Map, *Proc. IEEE*, 78(9), 1464-1480.

Kohonen, T. (1993) Physiological Interpretation of the Self-Organizing Map Algorithm, *Neural Networks*, 6(7), 895-905

Kohonen T. (1995/1997) Self-Organizing maps. Springer Verlag (1st & 2nd editions), Berlin.

LINAR, Inc. Introducing J-Integra: Whitepaper. http://www.linar.com

NEVA OBJECTS TECHNOLOGY, Inc. JavaDEE. http://www.nevaobject.com/java

Rallo, R. Arenas, A. Ferre-Gine, J and F. Giralt (2000) A neural virtual sensor for the inferential prediction of product quality from process variables. *Submitted to Computers and Chemical Engineering.*

Spoelder, H.J.W. (1999) Virtual Instrumentation and Virtual Environments. *IEEE Instrumentation & Measurement Magazine*, September, 14-19.

Ultsch, A., Siemon, H.P. (1990) Kohonen's Self-Organizing Feature Maps for exploratory data analysis. *Proc of INNC'90, Int. Neural Network Conf*, 305-308.

Vesanto, J. (1999) SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2), 111-126.

Appendix: Kohonen's Self-Organizing Map.

This neural architecture has been applied to visualization and dimension reduction of high dimensional data spaces, mainly due to its properties, related to topology and density of probability preservation on the reduced dimension space (output space).

The SOM algorithm (Kohonen, 1990) performs a topology preserving mapping from a high-dimensional input space onto a low dimensional output space formed by a regular grid of map units. The lattice of the grid can be either rectangular or hexagonal. This neural model is biologically plausible (Kohonen, 1993) and is present in various brain's structures, being used as an ordered low-dimension internal model of the external environment. From a functional point-of-view, the SOM resembles Vector Quantization (VQ) algorithms. These algorithms approximate, in an unsupervised way, to the probability density functions of vectorial variables by finite sets of "codebook" or reference vectors. The only purpose of these methods is to describe class borders using a nearest-neighbor rule, (see for example K-Means algorithm (ref. del K-Means Alg.)). In contrast, SOM's units are organized over the space spanned by the regular grid and the whole neighborhood is adapted, not only the winner unit.

Each unit of the map is represented by a weight vector, $m_i = [m_{i1},...,m_{in}]^T$ where n is equal to the dimension of the input space. As in vector quantization, every weight vector describing a class is called a codebook. Each unit has a topological neighborhood N_i determined by the form of the grid's lattice, either rectangular or hexagonal. The number of units as well as their topological relations are defined (and fixed) at the beginning of the training process. The granularity (size) of the map will determine its subsequent accuracy and generalization capabilities. During its training process, the SOM forms an elastic net that folds onto the cloud formed by the input data, trying to approximate the probability density function of the original data by placing more codebook vectors where the data are dense and few units where are sparse.

The training of the SOM proceed as follows, at each training step, one sample pattern **x** is randomly chosen from the training data, then similarities (distances) between **x** and the codebook vectors are computed (usually, the euclidean distance is used), looking for the "best matching unit" (BMU). This similarity matching can be expressed as:

$$\|x - m_{bmu}\| = \min_i \{\|x - m_i\|\} \qquad (1)$$

After finding the BMU and its topological neighbor cells, their degree of matching is increased by moving their codebook vectors in the proper direction in the input space. This competitive learning process follows a winner-takes-all approach, that can be described by the following rules:

$$m_i(t+1) = \begin{cases} m_i(t) + a\,(t)[x(t) - m_i(t)], & i \in N_{bmu}(t) \\ m_i(t), & i \notin N_{bmu}(t) \end{cases} \qquad (2)$$

where t denote time, $N_{bmu}(t)$ is a decreasing neighborhood function around the best matching unit, and $a(t)$ is a monotonically decreasing learning rate. Concerning the variation of the learning rate, we should consider two modes of operation of the SOM: an initial ordering phase in which the map is formed, and a later convergence phase, were fine tunning of codebook vectors is done. This approach corresponds to Hebbian learning on the topological neighborhood and active forgetting