DISTRIBUTED CONTINUOUS PROCESS SIMULATION: AN INDUSTRIAL CASE STUDY

[1]Raúl Alves Santos, [1]Julio E. Normey-Rico, [2]Alejandro Merino Gómez & [3]Cesar de Prada Moraga

[1] Department of Automation and Systems. Federal University of Santa Catarina.88040-900 CTC-UFSC. Florianópolis-SC Brazil. {ralves, julio}@das.ufsc.br
[2] Center of Sugar Tecnology. University of Valladolid. Edificio Alfonso VIII, 47011. Valladolid. Spain. alejandro@cta.uva.es
[3] Department of Systems Engineering and Automatic Control. University of Valladolid c/ Real de Burgos s/n 47011, Valladolid, Spain. prada@autom.uva.es

Abstract: This paper presents a case study of the creation of distributed continuous simulations using DCOM components. The proposed approach considers each part of the simulation as a DCOM component and uses an efficient simulation modeling language, EcosimPro, to perform the simulations contained in each one. This procedure has been applied to an industrial scale simulation corresponding to a sugar beet factory. The effect of the distribution degree in the performance has been studied. The simulation has been used to test advanced control algorithms and also in the construction of a simulator training system. *Copyright© 2005 IFAC*

Keywords**:** Distributed Simulation, Process Control, Process Simulators, Training.

1. INTRODUCTION

Simulation is recognised as one of the key technologies in the process industry. It is used for many purposes but perhaps the two most important applications are operator training and controller tuning and evaluation. In these cases, it is very important to execute the simulation in real time. For simple processes it is easy to obtain using conventional PCs. But when the complexity of the process increases, for instance if a complete factory must be simulated, parallel and distributed simulation must be used. Parallel simulations systems use multiprocessors computers with shared memory. Distributed simulations systems use a set of computers distributed in a network. Both solutions bring a series of benefits (Fujimoto, 2001): reduced execution time, allows the geographical distribution of the machines and the execution in heterogeneous systems. But the option of using conventional PCs distributed in a network is usually preferred because it can be implemented using existing infrastructure.

There are several tools and standards in the field of distributed simulation like HLA (High Level Architecture) (DMSO, 2004) and CAPE-OPEN (Computer Aided Process Engineering Open Simulation Environment) (CAPE-OPEN, 1999). HLA comes from the Department of Defence of US, it was approved as an IEEE standard in September of 2000, IEEE Standard 1516. It is focused on interoperability and reusability of the components and offers mechanisms for time management, as well as sophisticated data distribution concepts. It uses their own communication and synchronization libraries, called Runtime Infrastructure (RTI). HLA has been used mainly in military simulation applications, like combat simulators and also in some civil simulators applications like flight simulations, traffic control, etc. Most of these applications are discrete events or hybrid simulations. CAPE-OPEN is an European Community project whose purpose is the development, specification, test and publication of standards for the software components interfaces used to the development of process simulators. Like

HLA, does not specify the programming language in which the simulations must be develop. It standardizes the interfaces of the simulation components for the middlewares OMG CORBA (OMG, 1998) and Microsoft DCOM (Microsoft C., 1995). This solution is adopted by commercial simulators like INDISS (RSI, 2004).

The military simulations commonly are written in languages like C++, FORTRAN or Java, while the civil simulations are developed with commercial simulation tools like ACSL, Arena, GPSS/H, Dymola, gProms, Abacuss, EcosimPro, etc.

Distributed simulation is widely used with discrete event systems but there are almost no applications in the continuous process area. The interesting for this type of application is growing and the studies and implementation of solutions for this area is an open field of research (Acebes, 1999).

In this paper, a sugar beet control room training simulator is presented. It consists of a plant wide model and a control room environment. The simulation is based in EcosimPro (EcosimPro, 2004). As an OOML (Object Oriented Modeling Language) it allows the construction of libraries of component models that can be connected to simulate the different parts of a sugar factory. In order to simulate the whole factory a distributed architecture is used so that different parts of the factory simulation run in different interconnected computers performing a synchronized execution.

Next section presents the implementation of the proposed solution. Section 3 analyses the application to the sugar factory and the obtained results. The paper ends with the conclusions.

2. DISTRIBUTED SIMULATION

EcosimPro acts as a C++ simulation code generator that must be compiled with a Microsoft Visual C++ compiler. For distributed simulation, anyone of the two mentioned middlewares can be used, but for its execution under a Windows operating system and in a homogenous environment it is more appropriate the direct use of DCOM, because it is included in the operating system and does not requires the installation of additional packages. Also, Microsoft development tools include facilities for the development of COM components.

In our case study we have a series of simulations developed with EcosimPro, corresponding to different parts of a sugar factory, including the process and its control system. The main steps for the development of the distributed simulation with EcosimPro vía DCOM are: i) identification of the cut points of the simulation, following criteria of data interchange, process integrity and number of distributed modules; ii) generation of the C++ code correspondent to the simulations; iii) encapsulation of the simulations in DCOM servers; and iv) communication and synchronization among them.

2.1 Simulation and partition.

EcosimPro belongs to the family of the so called modelling languages, such as gProms, Abacus, Dymola… in the sense that they support non-causal models able to be modified automatically according to the context in which they are used. This means that the user can specify different boundary conditions without modifying the model code, and EcosimPro will manipulate symbolically the equations to adapt them to the specifications, which increases its re-usability. Being object-oriented it support the paradigms of encapsulation, inheritance, etc.

Its simulation language, called EL, allows the description of process models, named components, in a natural way by means of continuous differential-algebraic equations and discrete events variables. Once the user has built a system interconnecting components by ports, it is compiled and, after establishing a partition, that is, describing which variables constitute the known boundary conditions, EcosimPro generates C++ simulation code linked to the numerical solvers. This increases the open character of the language. EcosimPro can deal with large models and has been tested in different fields such as aerospace, power stations or process industries. A typical process model involving several thousand DAEs will be integrated normally in a fast and reliable way, but the speed of execution, assuring a given precision, depends not only in the size, but, mainly, in the nature of the DAE involved. This means that, if real time execution is required, some large models must be executed in a distributed environment.

The choice of the points where a compact simulation can be split in several ones is not an easy task. Important aspects to be considered are not only the number of variables that must be interchanged among the parts during the simulation execution, but, having in mind the constraints imposed by the numerical integration of the model differential equations, it is also important to consider the speed of change of these variables. If fast changing variables are involved, the time interval between data exchanges should be reduced in order to maintain integration errors below a desired threshold, which can represent a heavier load than another distribution with a higher number of slower changing variables to interchange. The variable that determines the communication time interval is called CINT. It is very important to establish a correct value to it. Increasing the CINT introduces errors in the simulation but at the same time increases the execution performance because the communication times are smaller.

2.2 Encapsulation in DCOM Servers.

DCOM is the Microsoft solution for a component software bus, and has the same philosophy as CORBA. DCOM extends COM (Component Object Model) to support communication between objects in different computers, in a LAN, WAN, or even in Internet and allows the distribution of components throughout a network. COM consists of a set of interface specifications that allow communications between clients and a server in the same machine. It provides support for the creation of software components, and incorporates properties like language independence, location transparency, etc. Using this characteristic, simulations can be encapsulated in DCOM components. In order to control and access the encapsulated simulation, an interface with a set of methods must be created (Fig. 1). After the components were created they must be installed in the different computers of the network.

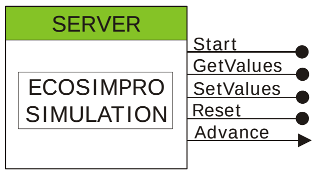

Fig. 1. Interface for the component which includes the simulation.

The access is done through the methods contained in the components interface, which will allow the management of the simulation.

In the components, the CINT variable establishes the time interval for data interchange between simulations. The data that are being interchanged are the values of the boundary conditions of every model component. Notice that, each simulation assumes that during the CINT time interval, the boundary conditions remain constant, which is the main source of errors.

2.3 Communication and Synchronization.

To perform the cooperative execution of the simulations it is necessary communicate and synchronize them. For this purpose there is an application (coordinator) that acts as a client.

Simulation advance is handled by a thread located in each component. The execution basic cycle consists of: waiting until all the data have been received, integrating the model up to the next time interval and notifying the coordinator that the calculation has finished and the data are available. Written and read data are stored in an intermediate buffer, in order to avoid the reads and writes while the simulation is performing the model integration.

In the client side, the coordinator, in a first step, initializes the servers that are going to be used. After that, a cycle of actions begin. The client waits until it receives the advance signals from each server, which indicate the end of the simulation interval calculation. Then it reads the data from the servers, it generates the corresponding writing vectors and writes them to the target servers. After that, checks if the simulation is finished, if not, another iteration cycle is done. Once the simulation ends the servers finalize their execution. Finally the function that closes all the communications is called. The synchronization between the client and the servers is made through the data writings and the advance signals sent from the servers to the client. The chronogram in Fig. 2 illustrates the joint execution of the servers and the client, and shows how the synchronization is made.

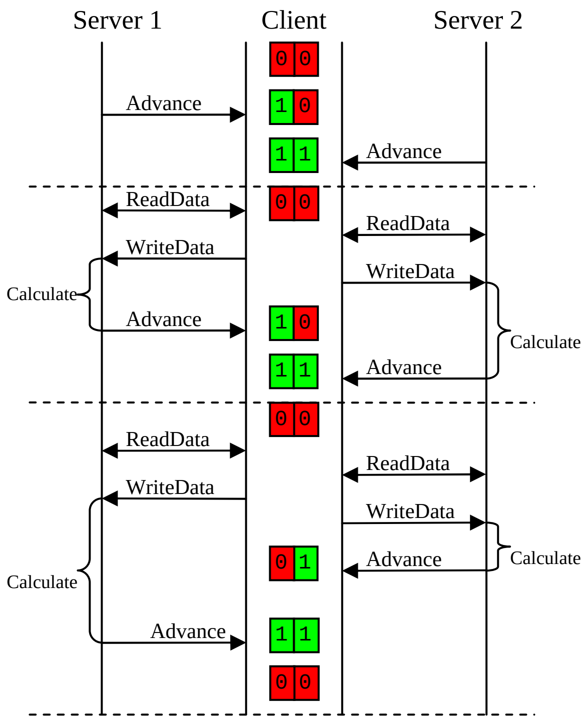

Fig. 2. Chronogram of the communication and synchronization.

3. APPLICATIONS AND RESULTS

This section describes the process, the training simulator and some results related to its operation.

3.1 The simulated process.

Sugar factories are fairly complex plants that involve a great variety of processes, ranging from external services such as a power plant, distillation and fermentation, waste water treatments, to the typical sugar process units: diffusers, evaporators, reactors, heat exchangers, crystallisers, dryers, filters, etc. operating both in continuous and batch modes. A schematic of the process can be seen in Fig. 3, showing the main sections that compose the main line of beet sugar production each one involving dozens of process units and thousands of variables.

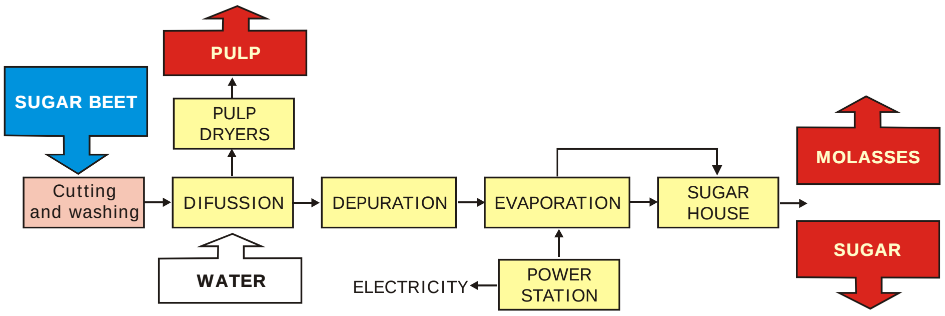

Fig. 3. Main sections in a sugar factory.

The simulation of the factory has been performed in EcosimPro, organising several libraries of process units, products properties, ports and control equipment The components were then linked together as in the real process and parameterised using process data and technical documentation from a factory in Benavente, Spain. As the complete simulation of the sugar factory represents a huge model that requires a great amount of computational time, the execution in a single conventional computer is not possible, if real time specifications are required, and a distributed simulation must be implemented. The fact that each section in Fig. 3 is linked to the adjacent ones by a small number of pipes which mainly send juices and steam, facilitates the task of defining cut-points for the distribution of the simulation which correspond roughly to the working departments or sections. Thus, the complete simulation system was divided into seven individual simulations, one for each section, except for the depuration, the biggest in terms of computation because it involves a lot of mixed continuous and batch units, which has been subdivided in two parts. Fig. 4 shows how the different sections are connected, as well as the number of interchanged data. Table 1 illustrates the size of each section.

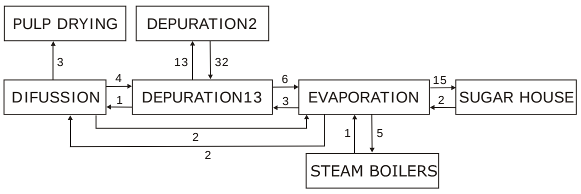

Fig. 4. Data interchange between the different sections of the sugar factory.

Table 1. Size of each section.

	Equations	Variables	Outputs	Inputs
Difussion	3105	5176	9	3
Pulp Drying	1303	2670	0	3
Depuration13	4505	7325	20	39
Depuration2	2608	4643	32	13
Evaporation	1936	4020	25	11
Sugar House	5578	9534	2	15
Steam Boilers	2545	4765	1	5
	21580	38133	89	89

The architecture of the system has seven servers and one client coordinator that interchange data and synchronize them.

3.2 Test results.

The tests were done in a set of seven computers, with Windows 2000 Pro operating system, connected to a standard 10/100 Mb network. The characteristics of these computers are listed in Table 2.

Table 2. Computers characteristics.

Id.	Processor
PC1 / PC3 / PC4 / PC5	PIV 1.6 GHz
PC2	PIII 800 MHz
PC6	PIII 1GHz

In the tests, different combinations of the sections to be distributed were considered, representing several degrees of distribution. The simulations were assigned to the available computers and the calculation and communication times were obtained. Table 3 shows how the distribution was done, indicating in each row in which computer the simulations will be executed. The tests, named Pi, begin with the execution in only one computer and end with the execution in six computers.

Table 3. Distribution of the simulations.

Test	DIF	PD	D13	DE2	EVA	SH	SB	CL
P1	PC1	PC1	PC1	PC1	PC1	PC1	PC1	PC1
P2	PC1	PC3	PC1	PC1	PC3	PC3	PC3	PC1
P3	PC3	PC2	PC1	PC1	PC2	PC3	PC1	PC1
P4	PC3	PC2	PC1	PC1	PC3	PC4	PC2	PC1
P5	PC3	PC3	PC1	PC5	PC2	PC4	PC1	PC1
P6	PC3	PC6	PC1	PC5	PC6	PC4	PC2	PC1

The results obtained according to the distribution indicated in Table 3 are shown in Table 4. The times correspond to the execution of 300 seconds of the simulation with a CINT of 5 seconds. The last column gives the average communication time for one time.

Table 4. Execution times (in seconds).

Test	Time	Comm	% Comm	Interval Time
P1	918,037	0,0878	0,0096	0,001489
P2	533,112	0,1713	0,032	0,002904
P3	541,806	0,2330	0,043	0,003949
P4	358,144	0,2196	0,061	0,003722
P5	320,415	0,2292	0,072	0,003884
P6	284,965	0,2743	0,096	0,004650

Notice that, in spite of the fact that when the degree of distribution increases the time needed to interchange the data between simulations increases, the overall execution time decreases.

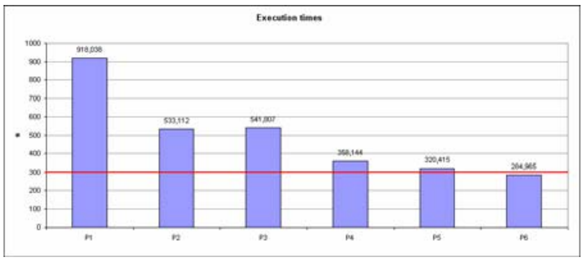

Fig. 5. Execution time for different distribution degrees.

With the increase of the distribution degree the execution time is not always smaller (see P2 P3 in Fig. 5). These times are conditioned for the computational load of the simulation, and the performance of the computer in which it is executed. Fig. 6 shows the execution time of each simulation and how this time depends on the computational requirements of the simulated process, the distribution degree and the performance of the computer where the simulation is located.

Looking at the times of each simulation it is possible to determine how to distribute the simulations when the distribution degree increases. Also is it possible to determine when the increase in distribution will not give an increase in the performance. For instance, if we are in case P6 and we add another computer, with a performance lower or equal to the faster computer used, the final execution time would not be reduced because the simulation n. 3 in the computer 1 have a execution time higher than the execution time of the simulation 2 and 5 in the computer 3.
As can be seen the distribution of the simulation gives: (a) lower execution times, (b) larger communication times. In this case the communication time is always below 0.1% of the total time, with an interval communication time lower than 5ms.

3.3 Application to a training simulator.

One of the possible fields of application of simulation is in a training simulator. To do this, real time execution is needed. The simulations have to finish their calculation before the CINT, and the client must wait until the end of the CINT to synchronise them, simulating in this way real time operation. In this section a training simulator of sugar factories is presented briefly (Acebes, 2003).

The architecture of the simulator includes, besides the distributed simulation, a SCADA (Supervisor Control And Data Acquisition System) connected to the DCOM servers. This SCADA has been developed in Microsoft Visual C++, an includes an OPC (OLE Process Control) client in order to access the developed servers (Iwainitz *et al*., 2002).

Nevertheless, the DCOM servers who contain the simulations are not directly accessible by the SCADA. Thus an intermediate OPC server has been developed, which allows the access to the variables of the simulation via OPC.

Fig. 7 shows the architecture of the training simulator, of the sugar factory. The communications for the synchronization is done by a DCOM client, and the communications with the SCADA are done through the OPC servers.

This architecture provides communications at two levels. At low level, in which the interchange of data and synchronization is made, the communication time must be reduced to the minimum possible. With this strategy the real time execution of the simulation is not delayed. Also, this allows the executions with smaller CINTs obtaining greater fidelity. The other level is used to communicate the simulations with the SCADA. These are not critical communications, and the communication times can be higher.

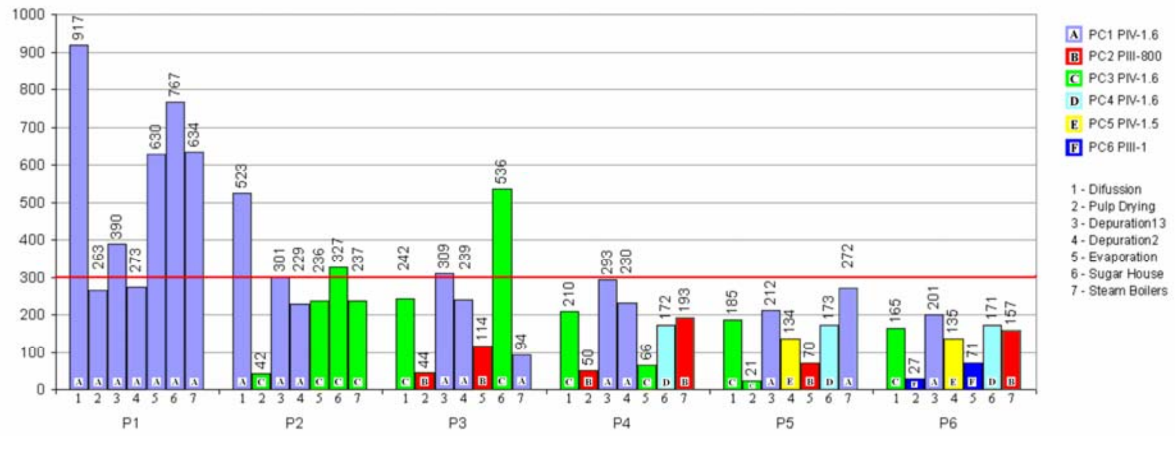

Fig. 6. Calculation times of each simulation when the distribution degree increases.

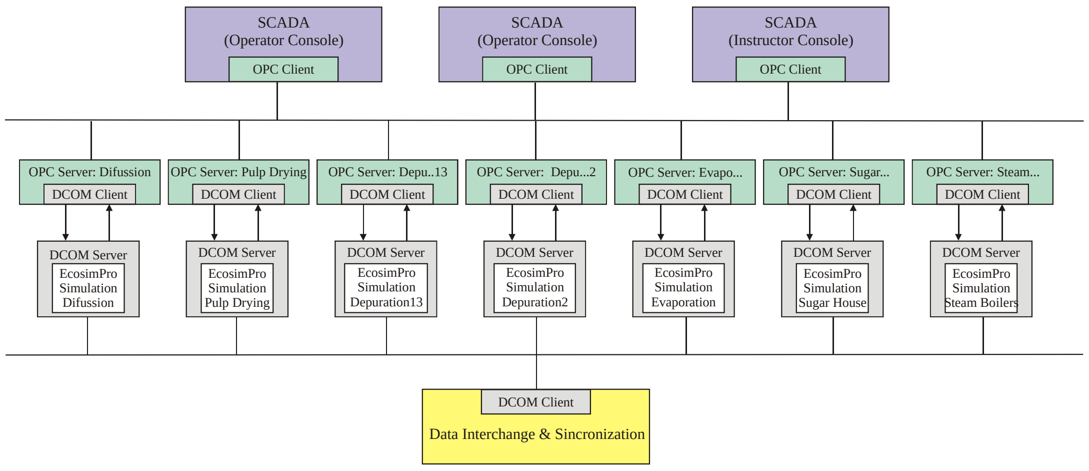

Fig. 7. Sugar factory simulator training system architecture scheme.

3.4 Control application.

The sections of the sugar factory have a set of local control loops (mainly PID controllers) that manages the dynamic behaviour of the process. These ones are executed as a part of the process simulation. With the use of a SCADA system the parameters and set points of these local regulators could be modified on line during the simulation execution. Also advanced control algorithms could be used to implement a MIMO global control of the system. In the particular case of the sugar factory model based predictive controllers were designed and tested. This procedure allows the tuning of the controller parameters before the implementation in the real factory.

4. CONCLUSIONS

A case study for the development of distributed simulations via DCOM oriented to a simulator of the process industry has been presented. The communication and synchronization problems have been solved.

The main advantages of this approach are: (i) independent programming of the simulated models and the mechanisms of communication, (ii) use of a well-known and widely used technology like DCOM, (iii) use of conventional computers available in a network, (iv) it could be directly applied to industrial scale simulations.

Finally, this case study has shown the utility of the distributed simulation in the continuous process area, when accelerated or real time execution is required. With the increase of the distribution degree the desired execution times had been obtained. With more powerful machines real-time execution has been obtained, in almost all of the interval times.

REFERENCES

Acebes L. F., Prada C. (1999).”Process and Control Design Using Dynamic Simulation”. *CITS General Assembly’99* Amberes, Holland.

Acebes L.F., de Prada C., Alves R., Merino A., Pelayo S., García A., Gutierrez G., Rueda A. (2003). “Development Tools for Full Scale Simulators of Sugar Factories”, *CITS’03. , 22th General Assembly of the CITS*, Madrid.

CAPE-OPEN. (1999). “Next Generation Computer-Aided Process Engineering Open Simulation Environment: Public Synthesis & Roadmap”

DMSO, Defense Modeling and Simulation Office. (2004) [Online]. Retrieved from http://www.dmso.mil.

EcosimPro by EA Internacional (2004). Dynamic Modeling & Simulation Tool [Online]. Retrieved from http://www.ecosimpro.com.

Fujimoto Richard M. (2001). “Parallel and Distributed Simulation Systems”, *Proceeding of the 2001 Winter Simulation Conference*.

Iwainitz F. and Lance J. A. (2002). “OPC-Fundamentals, Implementation and Application”, ISBN 3-7785-2883-1.

Microsoft Corporation and Digital Equipment Corporation (1995). “The Component Object Model Specification, Draft Version 0.9”.

OMG, Object Management Group (1998). “The Common Object Request Broker: Arquitecture and Specification”, 2.2 ed.

RSI (2004), [Online]. Retrieved from http://www.rsi-france.com.