**Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005**

Using cyclic executives for achieving closed loop co-simulation

U. Gangoiti, M. Marcos, *Member, IEEE*, E. Estévez

Abstract— **Co-simulation between models that focus in different aspects of the same system is a powerful tool during the design of a system. In this paper, a co-simulation application that closes the loop between the process model and the control system, running in different software tools, is presented. The target systems are Industrial Process Measurement and Control Systems (IPMCS) that usually are implemented using PLCs as the primary control equipment. The configuration of the co-simulation is achieved by extracting information from the software model of the system being designed. Mainly, the concurrent control algorithms as well as their temporal requirements (period and priority) and the process model input/output variables are used for designing a simple cyclic executive. This cyclic executive marks the time instants in which data exchange for each control loop must be performed. These synchronization times are used by the co-simulation application which allows performing incremental validations of the control system before its implementation. The framework is based on CORBA middleware. This middleware allows performing the co-simulation among heterogeneous and distributed tools.**

I. Introduction

THIS paper describes an approach for the design of a co-simulation framework towards the design of distributed Industrial-Process Measurement and Control Systems (IPMCS).

Nowadays, the use of Programmable Logic Controllers (PLCs) is widely spread in industry. Technological advances in these controllers have allowed the improvement of the manufacturing processes, reducing costs and optimizing production. The use of standards has a great force in the fast growth and development in the control and the instrumentation fields of the industrial processes. In this sense, the International Electrotechnical Commission's (IEC) 1131 [1] standard, developed with the input of vendors, international end-users and academics, is the international standard for programmable controllers. It provides five programming languages and guidelines to implement vendor independent applications. It also provides facilities for the implementation of multithreaded, multi-rated and distributed applications.

On the other hand, many of today's manufacturers are faced with difficult challenges including reducing time to market and achieving right first time designs. To achieve this goal, computer models are replacing the physical prototypes. There are many benefits to this approach including a faster and lower costs development with greater number of evaluated design alternatives and more optimal designs achieved. Plenty of proprietary tools exist to design, program and configure their systems, but it is difficult to reuse the work in other equipments. Open systems concept tries to overcome these difficulties through the integration of proprietary COTS tools so that they collaborate in the phases of the distributed system development cycle.

In the current market it is possible to find plenty of well-known software packages that can be used during certain phases of the design cycle of this kind of applications: modelling tools, analysis and simulation tools, code generation tools, configuration tools, etc. These tools make it possible designing, simulating and validating the design before its implementation. However, the integration of these tools, which contain different looks of the same system, is still a major problem. In fact, there are neither methodologies nor tools that support integrating all these kind of components.

Within the IPMCS application field, it is also possible to find powerful hardware components as well as multi-disciplinary COTS tools that are used in the design of this type of systems, but most of the commercial tools have little support to be integrated with others. Some software vendors are adopting proprietary tool integration approaches, following the final developers demands. There are also a few software packages for system analysis and simulation that allow exchanging information among the tools that cover the design phase. For instance, Extessy [2] integrates the Matlab modelling and simulation environment [3] with ARTiSAN RtS UML tool [4]. HyPneu [5] integrates hydraulic and pneumatic models with Simulink applications; CosiMate [6] integrates Mechatronics and Simulink.

Tool integration has also been investigated in the automotive sector integrating ADAMS and Xmath tools [7]. Some attempts have also been done by universities; in [8] a high level architecture to communicate different simulators is presented. In all cases, a proprietary integration is achieved for a few tools allowing them to collaborate within a closed environment. These solutions only cover certain phases of the design cycle. It is not possible to easily add new tools to the environment offered by a software vendor.

In this work a co-simulation application between commercial tools is presented as a pragmatic approach towards the achievement of tool integration. CORBA middleware is used to achieve co-simulation among heterogeneous and distributed tools. The steps to be followed for adding a new tool to the set have been described in previous works of the authors [9], [10]. Each tool acts as a CORBA server, offering a set of services to be

Manuscript received September 7, 2005. This work was financed by the European Union's Information and Science Technologies programme (FLEXICON project IST-2001-37269) and by and by MCYT&FEDER under project DPI 2003-2399.

G. U., M. M. and E. E. Authors are with the Department of Automation and Systems Engineering, University of the Basque Country, Spain (e-mail: unai.gangoiti@ehu.es, marga.marcos@ehu.es, elisabet.estevez@ehu.es).

performed on the simulation model. The synchronization among tools is achieved through a configuration file, which defines the time instants in which information exchange and command execution must be achieved.

This paper presents a co-simulation application that closes the loop between the control system (a model in a PLC programming tool) and the process (a model in a modelling and simulation tool). This application has been integrated into a general framework that supports the design cycle of IPMCS. In [11] a detailed description of the framework, based on a formal model for IPMCS proposed by authors in previous works [12], can be found. The formal model captures all the aspects of the system being designed in terms of functionality and implementation issues (hardware and software). The software model of the application follows the IEC 1131-3 standard and it defines the tasks containing control algorithms as well as their temporal requirements. The integration of the co-simulation application is achieved extracting this information from the application model and generating automatically the configuration file where the synchronization time instants are defined. The co-simulation application is used during the test phase of the application development for achieving the validation of the control system.

The layout of the paper is as follows: section 2 briefly describes the formal modelling of IPMCS proposed in previous works. In this section, and taking into account that the co-simulation application extracts the information concerned to the software architecture. In section 3, the design of the closed loop co-simulation is described. Firstly, the general scenario for the co-simulation through CORBA middleware is presented. Secondly, the design of the closed loop co-simulation application as a cyclic executive is detailed. Finally, the automatic generation of the co-simulation configuration is illustrated. Section 4 describes the general framework developed within the European project FLEXICON IST-2001-37269 and it also illustrates the co-simulation application through a case study.

II. Formal Modelling of IPMCS applications

Normally, industrial applications present few variations in their functional structure. In fact, in many cases, parts of this structure are re-usable in other applications. Reusability is one of the main requirements in the design and development of this type of applications. In order to facilitate the reusability of the design it is advisable to define a hierarchical functionality as well as to base the description of the application on modular components.

The whole application model is thus defined as the set of components illustrated in Fig 1. Note that the implementation issues are defined in the hardware architecture (processing and I/O nodes) and the software architecture is defined for each processing resource.

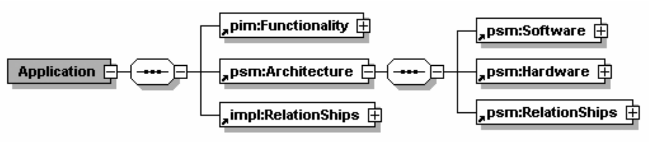

Fig. 1. Application scheme

A detailed description about the overall system modelling can be found in [12]. The software model, proposed by the IEC 1131 standard, defines the following elements illustrated in Fig.2.:

- Configuration: e.g. PLC
- Resource: It provides support for program execution (e.g. CPU or Virtual Machine)
- Task: It allows the designer to control the execution rates of different parts of the program.
- POU (Program Organisation Unit): Program, Function Block (FB) and Functions. They provide software reuse.

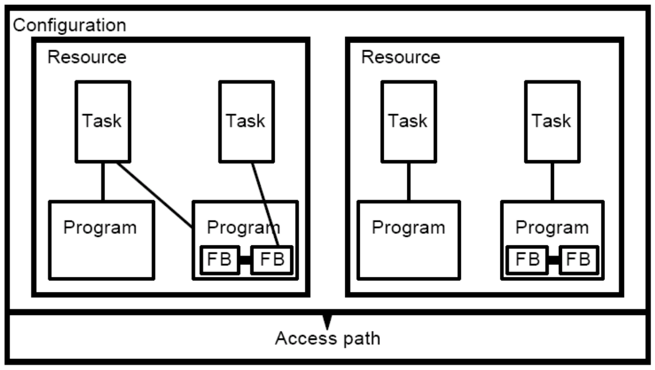

Fig.2. IEC 1131 software model

In particular, the task elements within resources contain the information related to control algorithms. They are mainly characterized by their activation period and priority. They are in charge of the timed execution of programs. Programs that are not associated to any task, executed at every cycle, are those in charge of performing the logical and sequential control. The definition of the programs has been enriched with temporal characteristics. These characteristics include the worst, average and best execution times (WCET, ACET, and BCET). This information, extracted from the model, can also be used to ensure the response time constraints are fulfilled using a temporal analysis tool.

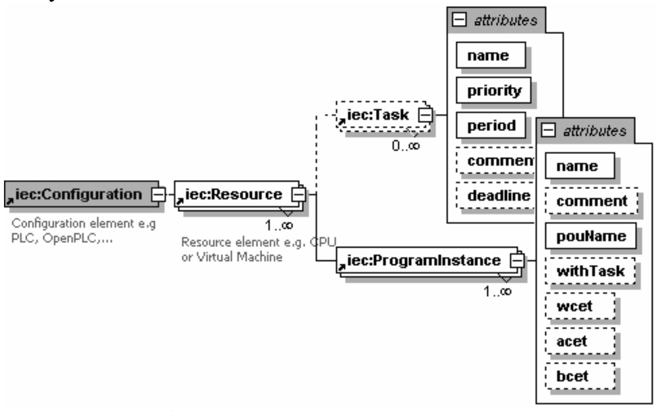

Fig.3. Hierarchy of the IEC 1131 elements

Besides task and program information of each configuration, it is also important to obtain the inputs and outputs that have to be updated each synchronization instant between simulations of tools. This information can be found in the architecture elements shown in Fig 4. On one hand it defines the relation configuration/PLC. On the other hand it relates the variables of each resource contained in a configuration that are mapped to input/output nodes. Thus, these variables and their properties (in/out, type and name) define the set of inputs and outputs that must be exchanged between the control model and the process model.

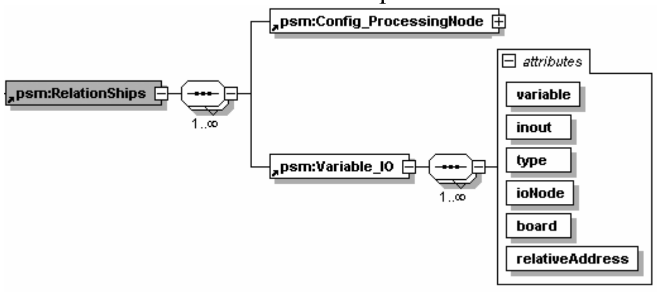

Fig.4. Software to Hardware Architecture relationships

III. Closed Loop Co-simulation Application

A generic co-simulation framework based on CORBA middleware was proposed in [9] and [10]. The main goal of this co-simulation framework is to achieve the collaboration of N tools, as it is shown in Fig.5. In the generic co-simulation of N tools, there is a CORBA server per each tool which offers data write and read as well as command dispatching services using the tool API. The co-simulation application, which acts as the referee during the co-simulation, follows the instructions defined in the co-simulation configuration file. In this configuration file, the co-simulation logic is defined. The co-simulation application sends instructions to the servers in the order and time instants defined in the configuration file. Any new tool can be easily added to the co-simulation framework.

In this sense, in order to add a new tool to the set, the following steps have to be followed:

Firstly, a detailed study of the tool API as well as the necessary services must be performed. With this information the CORBA interfaces are defined. The second step is to develop the CORBA servers. These servers are the implementation of the services defined in the first step. Finally, the co-simulation logic, including the actions on the tool and time interval of tool simulation must be defined in the co-simulation configuration file.

The generic co-simulation framework, illustrated in figure 5, has been integrated within the FLEXICON toolset for IPMCS. In this paper, the integration of closed-loop co-simulation between a control system and the process model is described. The control is developed in a PLC programming tool (ISaGRAF Enhanced), and the model of the process in a modeling and simulation tool (Simulink/Matlab). During the integration of the co-simulation framework the co-simulation configuration file is automatically generated. In order to generate this file, the following information must be extracted from the system model: period, priority and worst case execution times and associated variables for the timed tasks, as well as the name, type and direction of all the variables exchanged between the control system and the process model.

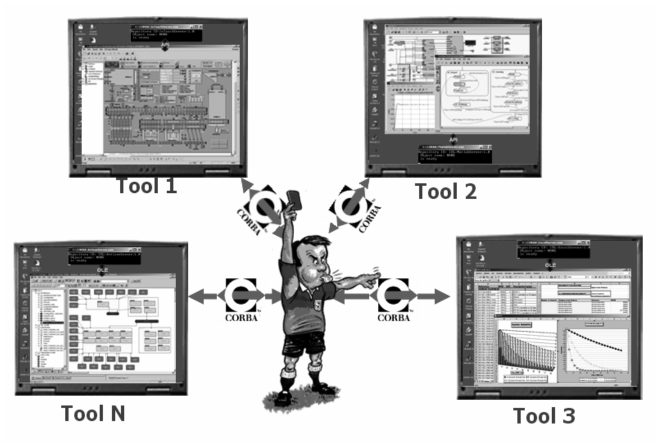

Fig.5. Generic co-simulation scenario

Detailed information about the generic co-simulation framework can be found in [9].

A. Services needed for closed-loop co-simulation

The control algorithms of this kind of systems are usually implemented in PLC programming tools and the process is usually modelled in simulation tools. The complete control system is usually composed by a set of periodic control loops that have to be executed at its sampling period and the logic control part that has to be executed every PLC cycle. Thus, the process has to be simulated as frequently as possible refreshing the control inputs and outputs that are related to the login control. However, the variables of the control system associated to timed tasks (control algorithms) must be updated only when its sampling period is met.

To perform this operation, the necessary services include:

- Starting the simulation tools and initializing variables.
- Reading and writing values of the types used in the co-simulation. Taking into account the different data types used by the modelling tool and the control tool different CORBA services for variable write/read have to be implemented. For instance, tables I and table II illustrates the data types defined by Matlab/Simulink environment and ISaGRAF Enhanced programming tool respectively.
-

TABLE I
PROCESS DATA TYPES

	type	Size	Range
boolean	logical	1 bit	true,false
integer	int8	8 bits	-128..127
word	int32	32bits	$-2^{32}..2^{32}-1$
real	Float	32bits	-1E8..1E8

TABLE II
CONTROL DATA TYPES

	type	Size	Range

boolean	boolean	ISA_TYPBOOL	uchar
integer	short integer	ISA_TYPSINT	char
word	double integer	ISA_TYPDINT	int32
real	real	ISA_TYPREAL	float

- Finally, a service that executes a simulation step in both tools.

Fig. 6. shows the CORBA services for reading/writing the different types of variables in Matlab and for executing any command (*evalString).*

```
module Matlab {
   interface MatlabServer{
    short iRead (in string variable);
    float rRead (in string variable);
    double wRead (in string variable);
    boolean bRead (in string variable);
    void iWrite (in string variable, in short value);
    void rWrite (in string variable, in float value);
    void wWrite (in string variable, in double value);
    void bWrite (in string variable, in boolean value);
    void evalString (in string command);}}
```

Fig. 6. Matlab/Simulink services

Fig. 7. shows the CORBA services for reading/writing the different variables in ISaGRAF Enhanced as well as to execute the next cycle of a resource (*nextCycle*).

```
module ISaGRAF {
    interface ISaGRAFServer{
    short iRead (in short resource, in string variable);
    float rRead (in short resource, in string variable);
    double wRead (in short resource, in string variable);
    boolean bRead (in short resource, in string variable);
    void iWrite (in short resource, in string variable, in short
                  value);
    void rWrite (in short resource, in string variable, in float
                  value);
    void wWrite (in short resource, in string variable, in
                  double value);
    void bWrite (in short resource, in string variable, in
                  boolean value);
void nextCycle (in short resource);}}
```

Fig. 7. ISaGRAF Enhanced services.

In [9] a detailed description of the services used in the Simulink/Matlab and ISaGRAF Enhanced co-simulation can be found.

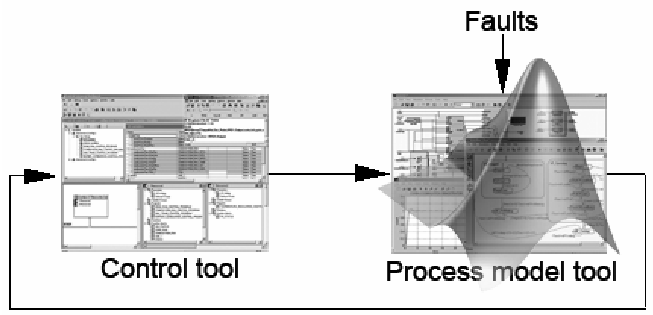

Fig.8. Co-simulation scenario

The closed loop co-simulation scenario between the process modelling tool and the control tool is shown in Fig.8.

B. Design of the co-simulation application as a cyclic executive

In order to achieve the co-simulation it is necessary to synchronize the tools in simulation time and the logic control has to be executed continuously. Specifically, control algorithms are characterized by their period, the synchronization times between tools correspond to the instants in which the control loops have to be executed.

The co-simulation application needs to know the successive instants in which any task that implement control algorithms have to be executed. Note that the interval between two of these instants define the next simulation step for the process model. In order facilitate the design of the simulation steps in both tools; it is proposed to apply cyclic executive design techniques to obtain periodic simulation steps, the so called secondary cycles of the cyclic executives.

The design also includes the tasks in each resource that should be updated each secondary cycle. Specifically, the logic control has to be executed continuously and the control algorithms (i.e. tasks in resources) have to be executed at their sampling periods)

In this way, a cyclic executive organizes the global execution of the co-simulation. To design this cyclic executive the following information has to be extracted from the application model:

- The period, T, and priority, P, of all tasks. This information is contained in the software architecture elements as illustrated in Fig.3.
- The execution time of each task. This is automatically computed as the sum of the execution times of all the programs associated to each task (see Fig.3.
- The type and direction of the variables to be exchanged: field signals are the input/output signal to/from the control and process. They are obtained from the hardware and software relationships, shown in Fig.4. Note that this information is needed to associate the input and output variables of each control algorithm.

From this information, a simple cyclic executive can be automatically generated from the following algorithm [14]:

- First, it is checked that the processor utilization is less or equal to one (necessary condition for a schedulable cyclic executive):

$$u = \sum_{i=1}^{n} \frac{c_i}{p_i} \le 1$$

- Calculates the major cycle duration; $T_m = lcm(T_i)$
- Calculates all the possible secondary cycle durations T_s, following the next rules:
    - o There must be a whole minor cycle between the activation and the deadline of every task (necessary condition): $\forall i : T_s \le d_i$
    - o Every task must execute within a minor cycle: $T_s \ge \max(c_i)$

- T_s must divide the major cycle duration T_m, every minor cycles must have the same duration: $\exists k : T_m = k \cdot T_s$
- There must be a whole minor cycle between the activation and the deadline of every process (necessary and sufficient): $\forall i : 2 \cdot T_s - gcm(T_s, T_i) \le d_i$

With this information the different values for secondary cycles are tried starting from the biggest.

At this point, the secondary and major cycle durations have been stated. The main structure of the cyclic executive has been created with the temporal requirements of the given tasks. The next step is to fill the empty frames with the different tasks (i.e. with the control algorithms that have to be updated).

- There are two options in order to define the mapping order of the tasks into the secondary cycles: "by priority" or "by worst execution time"
- Following the order defined in the last step, for each task, the possible secondary cycles where it is possible to place the activation of each task is calculated.
- Assign the activation of each task to a secondary cycle. There are two options in order to select the secondary cycle: "to fill the first free secondary cycle" or "to fill the minor cycle with the higher execution time free".

If it is not possible to schedule the cyclic executive with these options the user can try other filling order or divide tasks and try the scheduler again.

C. Automatic Generation of the Co-simulation Configuration

The co-simulation application interprets a configuration file in which the command execution and data interchange order is defined.

The co-simulation configuration file consists of three main parts. These parts are illustrated in Fig.9.

- Boot: In this section models loadings and tool initialization is defined. Tool initialization usually includes: setting the working path, opening models and initializing variables to their default value.
- Loop: The commands introduced in this section are sequentially and systematically repeated while the co-simulation is running. In this section the sequence of the different commands to be sent and data to be exchanged is defined.
- Faults: In this section the commands that punctually are going to be sent to the tools are defined. A practical case of use is fault injection in order to verify the behavior of the control system.

Each command defined in the file, contains the following information (see Fig.9.):

- Id: command identifier. It is unique in the file.
- Object: object used by the command
- Method: name of the object method to be invoked.
- Description: optional field that contains a brief description of the command.
- Order: this field defines the execution order of the commands

Depending on the characteristics of the method to be invoked, it can also contain:

- Argument: with its corresponding type and value. This value can be a literal or the output of the previous executed method

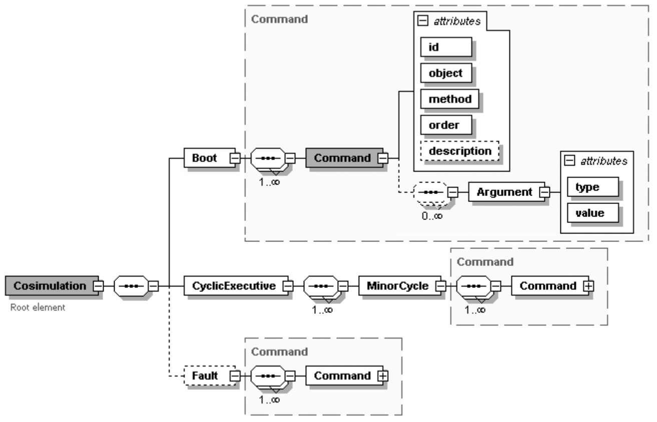

Fig.9. Generic co-simulation configuration file

The design of the configuration file as a cyclic executive in pseudo code is illustrated in Fig.10..

```
Procedure cyclicExecutive
begin
  loop
    forEach minorCycle
      repeat – logicControl
        sendData from control to process;
        simulateProcessStep;
        sendData from process to control;
      end repeat;
      forEach controlLoopToBeUpdated
        execute controlAlgorithm;
      end forEach;
    end forEach;
  end loop;
end cyclicExecutive;
```

Fig.10. Configuration file pseudo code

IV. THE FLEXICON TOOLSET FOR IPMCS

In this section the framework for IPMCS that includes the co-simulation application proposed in this paper is described. This framework has been developed within the European project FLEXICON IST-2001-37269. The general goal of the project is to develop methodologies that enable Commercial Off-The-Shelf (COTS) tools integration for the design and deployment of Distributed Control Systems (DCS) with high degree of flexibility, dependability and re-usability. This project specifically addresses the development of the capability to produce open, high performance, dependable, distributed fault tolerant systems in reduced timescales and at lower cost.

Within the toolset, the different phases of the design cycle are addressed using the appropriate tools. Fig. 11 illustrates the different tools involved during the application development cycle. In particular, the toolset integrates a set of COTS tools: ARTiSAN RtS UML tool as the modelling tool, ISaGRAF Enhanced as the PLC programming tool and Simulink as the simulation tool for modelling the process. The co-simulation application that has been integrated allows performing incremental validation of the control system.

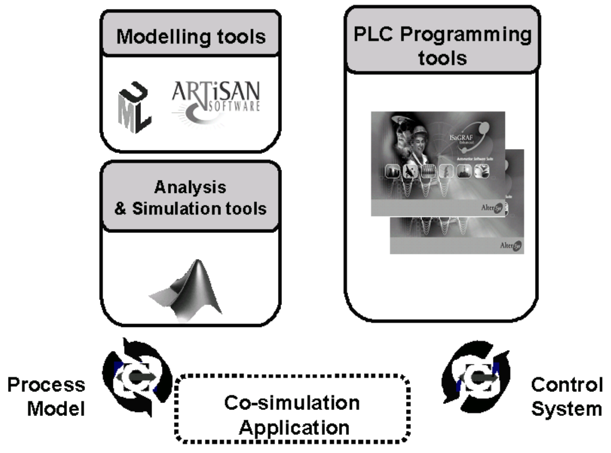

Fig. 11. The FLEXICON toolset for IPMCS

During the analysis phase, the process model is developed using the Matlab/Simulink environment as the analysis and simulation tool. On the other hand, the basic control functions have to be selected from repositories or developed within the PLC programming tool (ISaGRAF Enhanced). This phase concludes with the high level design of the control system as well as the identification of I/O and networking needs.

The Design phase uses the modelling tool, ARTiSAN RtS, for modelling the functional specification of the control system as well as the hardware and software architectures that implement it.

Coding and Testing phases consist of generating the control system project in the PLC programming tool and validating it through the co-simulation framework. In this sense, the co-simulation application allows to close the loop between the ISaGRAF Enhanced tool that models the control system and the Matlab tool that models the process. The flow of information is illustrated in Fig. 12.

The co-simulation framework gets the information from the application model, related to the cyclic and timed resources as well as the information to be exchanged between the process model and the control system. This information is needed in order to compute the time instants in which the data exchange should be performed.

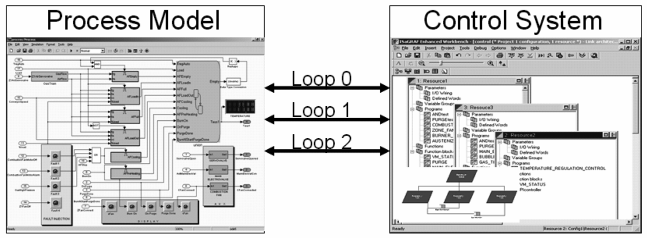

Fig. 12. Data flow between tools

V. Conclusion

In this paper a co-simulation application for closing the loop between a PLC programming tool and a process model simulation has been proposed. For the design of the co-simulation logic, cyclic executive design techniques have been applied. Thus, the goal is to obtain the major and minor cycles at which the control loops have to be updated to run the next simulation step of the process model. The use of these techniques can also be used to perform temporal analysis of the co-simulation behavior. This co-simulation application has been integrated in a general framework for designing IPCMS that covers the application life cycle by integrating a set of COTS tools.

References

[1] Lewis R.W. "Programming Industrial Control Systems using IEC 1131-3, IEE Control Engineering series 50. ISBN-0 85296 950 3", 1997.

[2] Extessy AG. (2003) URL: www.extessy.com

[3] The MathWorks (2002), Using Matlab version 6.5

[4] Artisan (2002). Manual of ARTiSAN Real-Time Studio, Version 4.2, ARTiSAN Software tools.

[5] BarDyne, Inc: Hydraulic and Fluid Power Experts. http://www.bardyne.com

[6] TNI Software: CosiMate. www.tni-world.com

[7] George N. (1998). "Cosimulation of an Automotive Control System using ADAMS and Xmath". International ADAMS User Conference. Utrecht, The Netherlands

[8] Adriano, B.A and Wagner, F.R. (2001). A Standardized Co-simulation Backbone. University of Porto Alegre, Brasil

[9] Marcos M., Gangoiti U., Estévez E., Portillo J., Calvo I. "A CORBA-based Co-simulation Framework for Integrating COTS Tools", submitted to: Sixth Portuguese Conference on Automatic Control CONTROLO 2004, Faro, Portugal. 2004.

[10] Marcos M., Gangoiti U., Calvo S. Estévez E., Orive D., Barandiarán J. "Design and Validation of Industrial Distributed Control Systems", The 43rd IEEE Conference on Decision and Control. Atlantis, Paradise Island, Bahamas. 2004.

[11] Marcos M, Estévez E., D. Orive "A Tool Integration Framework for Industrial Distributed Control Systems ". Joint 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005. Sevilla (Spain). Submitted for publication.

[12] Marcos M, Estévez E. "Formal modelling of Industrial Distributed Control Systems". 16th IFAC World Congress in Prague.2005. Accepted for publication.

[13] Foresign Systems, Inc. (2002) Combining Foresight and Matlab for Complex System Design.URL: www.Foresight-Systems.com

[14] Zamorano J., Alonso A., De la Puente J.A. "Building safety-critical real-time systems with reusable cyclic executives" Control Engineering Practice, v 5, n 7, July 1997, p 999-1005