Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Hierarchical and Distributed Embedded Control Kernel

R. Simarro*, J. Coronel, J. Simó**, J.F. Blanes****

**Departamento de Ingeniería de Sistemas y Automática (DISA)*
***Departamento de Informática de Sistemas y Computadores (DISCA)*
Universidad Politécnica de Valencia, Camino de Vera, s/n. 46022 Valencia (Spain)
(e-mail: rausifer@ai2.upv.es, jacopa@ai2.upv.es, jsimo@disca.upv.es, pblanes@disca.upv.es)

Abstract: This paper presents how to get a high control performance and a reliable operation, by means of a suitable combination of several Embedded Control Systems. For this purpose, a hierarchical and distributed control model is proposed. The model holds a set of activities that should be executed on it, such as change, switch and delegate new code of controllers into embedded nodes. All of these activities are managed by a middleware component following the *control kernel* concept. This model was tested on real processors interconnected in a CAN network, using a XScale microcomputer with a real time operating system (RTLinux) running a high level controller (GPC) and a dsPIC microcontroller for signal acquisition and delivering of control actions.

Keywords: Embedded Systems, Control Kernel, Distributed Control, General Predictive Controller, Real Time Control.

1. INTRODUCTION[1]

Distributed Control Systems (DCS) are used in industrial and civil engineering applications to monitor and control distributed equipment with remote human intervention. Moreover, these systems use a network to interconnect sensors, controllers, operator terminals and actuators.

While computing power of embedded systems is increasing over time, the networking technology trend is to move from control-specific networks to Ethernet based infrastructures and wireless communication. In this scenario the control strategy should be tolerant to variations in the message delays as well as bandwidth availability. Some work has been done in communication protocols with traffic characterisation, bandwidth allocation and clock synchronisation (Coronel et al., 2005). But this kind of cyber-physical interaction motivates a big amount of innovations in many Information Technology related fields including DCS architectures and controller design (Lee, 2006).

The DCS architecture assumed in this work is based on the "control kernel" concept (Albertos et al., 2006). These principles of organisation are intended to the automatically distribution of control code in a DCS insuring safe operation. The functionalities are provided by a specific middleware responsible of information and code distribution over the communication infrastructure.

Section 2 presents a distributed control model and gives an overview of the main components of the architecture. In the section 3, the main elements of the experimental platform are presented. Afterward, in section 4, two experimental cases studies are presented using the developed experimental platform depicted in section 3. The first experiment illustrates the controller switching mechanism while the second experiment present the implementation of a more complex hierarchical controller that uses a predictive controller (GPC) to develop a safe control system. Finally, section 5 summarizes the conclusions of the work.

[1] This work is part of the KERTROL project DPI2005-09327-C02-01/02 under the sponsorship of the Spanish Ministry of Education and Science.

2. AN APPROACH TO DISTRIBUTED EMBEDDED CONTROL

Safety is a crucial issue in embedded control systems. Independently of the number of variables to be controlled by the same processor, the systems with hard real time requirements must ensure the delivering of control actions to all actuators. The quality of the delivered signal can depend on the processing level: used data, the computational algorithms, resources availability, among other, but always must ensure the safe performance of system (Albertos *et al*., 2006).

Apart from components malfunction, in complex DCS, safety can be affected by the variation of the controlled system dynamics that requires controllers switching, missing execution deadlines, loosing messages and variation of communications delays. In this context, for running control applications in a safe mode, the following activities should be taken into account:

- Communication links with other activities should be activated.
- Some data should be recorded, displayed, stored and updated.
- It exist at least one controller that computes the control action based on available data at each time instant and using the predefined algorithms.
- According to the system behaviour, it must advance actions such as: disconnect and switch controllers. Controllers are parts of code that run spread in a distributed environment.

- If the control action has not been delivered by the current controller on time, a safe control action should be delivered at time required to the process. This signal may be the result of a simple calculation (but sufficiently safe) an emergency shutdown or simply a safe back-up response such as: *keep unchanged*. Note that this operation can be interpreted as a controller switching.

For this purpose, a distributed embedded control model can be defined as composed by two node types: *light nodes* and *Service nodes* (Fig.1). *Service nodes* are powerful embedded computers running a full featured RTOS and complete networking with I/O capabilities. Light nodes are small and low power consumption SoC processors with limited computing and networking capabilities but complete I/O features.

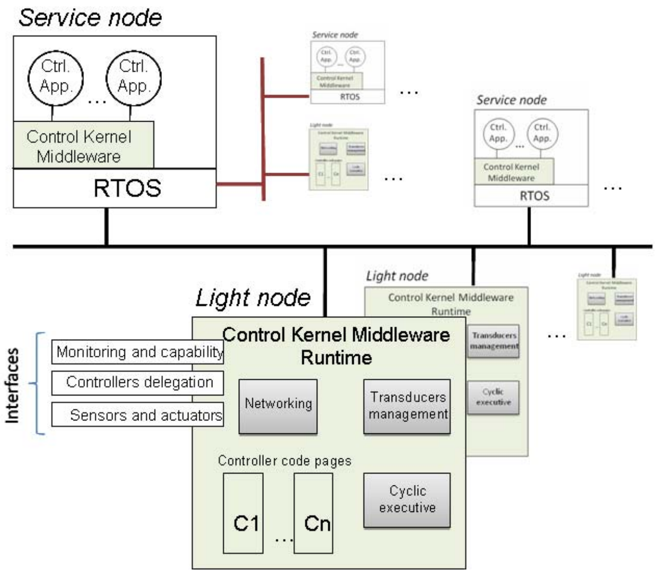

Fig. 1. Architecture of the Proposed Control Model

Control Applications run in *service nodes* on top of a full featured *Control Kernel Middleware* (CKM). This middleware offer abstractions and functionalities related to control tasks real time execution, access to sensors and actuators, and communications management. The programming model of CKM follows the concept of code delegation. In this sense, a control application delegates the execution of some control code to the CKM that provides computational resources to execute it. Note that a control task, once inside the CKM, can run on whatever *service node* of the DCS that have access to the communications space of the task.

Light nodes are a cost-effective solution to have some computer power as close as possible of each actuator. This is mandatory in order to reduce the indeterminism in the time of delivering control actions to the controlled process. *Light nodes* run a retail of the CKM: the *CKM Runtime*. This Runtime communicates with the CKM offering interfaces for management, sensing and acting and code upload. Features of *CKM Runtime* include network interfacing to sensors and actuators and controller code pages upload. A *light node* can be used as simple slave component to interface DCS or can run locally controllers in a cyclic executive environment.

Any controller of a Control Application that has been delegated to the CKM with attached native code page for the *light node* type, can be delegated to this light node by uploading this codepage and asking for switching. Controller pages can be uploaded through the *CKM Runtime* without any interference with the controllers currently running in the node. The uploaded pages are activated for running by the switching mechanism provided by the *CKM Runtime*.

In particular, *service nodes* may include supervising and optimising control activities and *light nodes* can run activities to drive the system to a safe position or run simple algorithm that guarantees a minimum of stability in the system at any time.

Light node ensures that always exist a control action ($u_f(k)$) to be sent to the process. This signal may be just a safe action (disconnect, open, close, unchange, etc.) or the result of a simple calculus (computed locally in the node) ($u_s(k)$) or it may be the signal calculated ($u_o(k)$) and received from a *service node*.

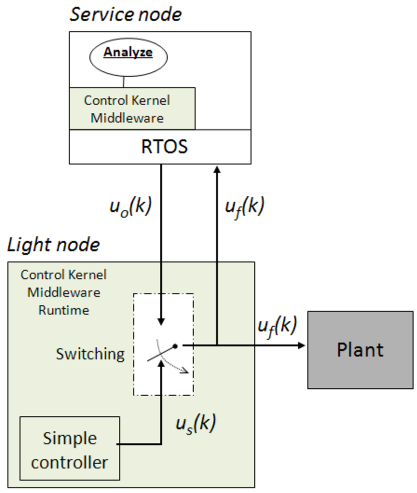

Fig. 2. The light node ensures that always exist a control action to be sent to the process ($u_o(k)$ or $u_s(k)$)

Let's consider the model depicted in figure 2. Two modes are defined for this control model:

- The *service node* produces high-quality control responses ($u_o(k)$) which are sent to the *light node* to be applied on the plant. If $u_o(k)$ is not received or it has some delay, then, the *light node* will apply his calculated control action $u_s(k)$.
- The *light node* controls directly the process and the *service node* only monitors and analyzes the sensory data and the control action $u_s(k)$ to determine if it is suitable.

When $u_s(k)$ is not detected or is wrong, immediately the signal $u_f(k)$ is switched to a safe signal $u_o(k)$. This switching may be executed into a *light* or a *service node*. Under these circumstances, the *service node* can determine if it is necessary to change and delegate new code into *light node* to execute other controller.

3. EXPERIMENTAL PLATFORM ELEMENTS

3.1 Embedded Nodes

The *light* and *service nodes* are based respectively on a dsPIC microcontroller and a XScale microprocessor.

The Microchip dsPIC (Microchip 2006) combines the huge computation speed of a Digital Signal Processor (DSP) with a powerful 16-bit microcontroller (MCU), to produce a tightly coupled single-chip single-instruction stream solution for embedded systems design. This dsPIC device achieves speeds of up to 30 MIPS, is efficient for C programming and has Flash program memory, data EEPROM, data SRAM, powerful peripherals and a variety of software libraries.

On the other hand, the XScale embedded architecture has been chosen as development platform for our *service node* due to their low power consumption, their high performance and their low cost. All the generations of XScale are 32-bit ARM v5TE processors manufactured with a 0.18 µm process technology. This processor support changes of core frequencies between 100 MHz and 400 MHz for optimization of power consumption with a top of computing power of 700 MIPS.

Service node uses the Real Time Operative System "RTLinux" (Yodaiken and Barabanov, 1996) which offers characteristics of a hard real-time system in a multi-threaded real-time kernel and which can be used as embedded O.S. But, for this it was necessary to make a porting to adapt the original RTLinux source code to XScale architecture. We have developed an RTLinux version with support to XScale architecture, available on http://rtportal.upv.es/

3.2 Communication protocol

Although there is a great variety of real time buses, CAN (Controller Area Network) (CiA, 1996) is one of the preferred solutions to communicate distributed real time systems (Coronel *et al.*, 2005). Therefore, for our experimental case, the communication middleware will use CAN as infrastructure to interconnect the two units.

In order to incorporate this communication protocol into XScale node, it has been designed an expansion board with a PIC microcontroller which has an embedded CAN chip. This PIC communicates with the XScale through a double port RAM memory. Moreover, the respective CAN drivers have been developed for RTLinux. On the other hand, the dsPIC node already has an embedded CAN chip.

4. EXPERIMENTAL WORK

In order to test the characteristics and capabilities of the proposed distributed control kernel model (Crespo *et al.*, 2006), two cases study will be presented in this section: first, we evaluate the ability to switching simple controllers located on different computation nodes interconnected through a shared communication channel, and after that, we take advantage of the distributed computing availability to run a predictive control algorithm to control a real process and provide fault tolerance to communication sporadic error.

4.1 Case Study 1. Switching of Process Controllers

The switching of controllers is one of the key features in the control kernel model to run control applications in a safe mode. For this case study the light node is directly connected to a simulated process and it send information about process state to the service node through the CAN communication bus.

4.1.1 Description of the Simulated Process

The *light node* has been connected through a DAQ card to a PC running a simulated system in MatLab with Simulink and Real-Time WorkShop toolboxes (MathWorks 2006). The system is a simulation of the real HUMUSOFT CE 152 Magnetic Levitation educational scale model. The simulated model transforms the error signal into a real analog one through the DAQ analog output. An analog input of the DAQ is used to get the control or feedback signal.

As shown the figure 3, the error signal (*e(k)*) is directly sampled by the dsPIC and its value is transmitted to *service node* by means of the communication bus. The *light node* also applies directly the final feedback signal (u_f) on the control process, whose u_f can be the u_o or u_s signal. This last depend of the switching mode.

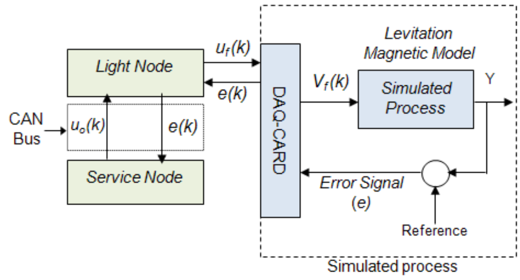

Fig. 3. Distributed System Diagram with Simulated Process Model

4.1.2 Regulator design

In this first test, for simplicity and in order to illustrate the controllers switching, the *light node* gets the tracking error signal of the system through an A/D converter pin and solves a simple proportional derivative PD discrete regulator. The control algorithm on the *light node* is:

$$u_s(k) = q_0 \cdot e(k) + q_1 \cdot e(k-1) \qquad (1)$$

The *service node* receive the error signal and executes a PID discrete regulator. The control algorithm on the *service node* uses the following difference equation (2):

$$u_0(k) = u_0(k-1) + q_0 \cdot e(k) + q_1 \cdot e(k-1) + q_2 \cdot e(k-2) \qquad (2)$$

where *e(k)* is the error in the instant "*k*", and $u_o(k)$ is the calculated control signal to apply to the simulated system.

Several tests have been made on a simulated environment to obtain the optimal coefficient values for the given process. With that result a local control for the simulated plant could be done at 5ms control cycles.

The result is sent from the light node to the system by a PWM output, though a RC low-pass filter to be converted into an analog continuous signal.

4.1.3 Running Modes

Two types of situations are defined for this control model (see figures 2 and 3): first, when the service node produces a high-quality control response ($u_o(k)$) which is sent to the light node to be applied on the plant, and therefore the light node acts only as an interface; and second, when the control is performed by the light node and the service node only monitors and analyzes the sensory data and the control action $u_s(k)$ received through the CAN bus.

For the first situation, if $u_o(k)$ is not received or has some delay, then, the light node (dsPIC) will apply his calculated control action ($u_s(k)$) (see figure 2). Thus, if an internal timer is up to a critical time delay, then CKM Runtime switches to the local PD controller into the dsPIC. A communication delay may represent that the *light node* is not working properly or the communication network is busy or down. This switching ensures that always exists a control action ($u_f(k)$) to be sent to the process. The figure 4 shows that the switching no affect the evolution of the process.

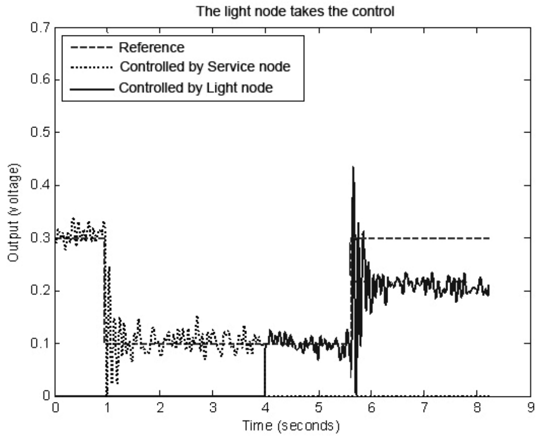

Fig. 4. Process signals evolution when some control CAN messages are lost and the system controller is switching from the service node to the light node.

For the second situation, the control action produced by the *light node* ($u_s(k)$) is analyzed to determine if it is suitable to control the system. When $u_s(k)$ is detected as wrong, immediately the signal $u_f(k)$ is switched to a safe signal $u_o(k)$ (see figure 2). The present and accumulated error signal values is analysed on the dsPIC, and if the error parameter exceeds a programmed value, the regulator changes to the *service node* controller. In the figure 5, the regulation is working successfully until the second 5 a lost data is simulated, then the process begins to be unstable, and therefore the regulator switches to *service node* controller to command it.

These nodes get the sensory data and deliver the control actions to the system through the communication middleware.

4.2 Case Study 2. Supervising control with local compensating

In this case the idea is to use the distributed computing availability to run a predictive control algorithm. This piece of control will provide future control actions that can be used by the local processor to feed actuators in the case of unexpected communications delays or missing data.

The predictive control algorithm is a Generalized Predictive Control (GPC). Next the developed strategy is explained.

4.2.1 Developed Strategy

The developed structure basically involves a distributed control system made by a *service node*, with a supervisor control GPC, and a *light node*. The *service node* is a system with wide capacity in computation and communication resources, whereas the light node is an embedded system with limited computation resources.

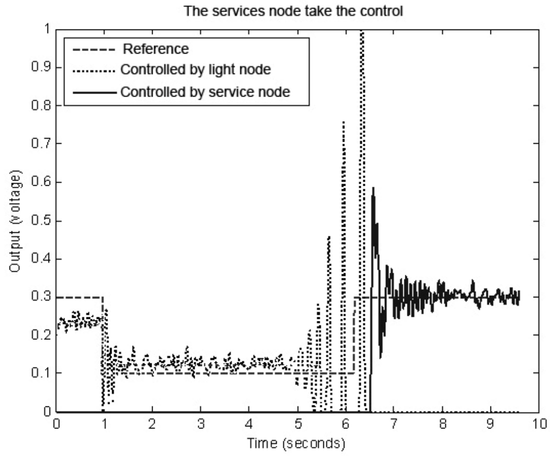

Fig. 5. Evolution of the process signals when a control error is detected and the controller is changed from the *light node* to the *service node*.

The service node manages the GPC control actions calculation that applies the light node. The *service node* sends, in each sampling period, the trajectories of control actions and outputs. If the *light node* applies the sequence of control actions that have calculated the GPC from the iteration "*k*", an open loop strategy would be done during the prediction horizon [N_1, N_2]. Therefore, the receding horizon strategy will be applied, so the *light node* uses only the first control action *u(k)* that has been sent by supervisor control, and this is a close loop strategy. This procedure is repeated in every sampling instant.

Nevertheless, the control actions sequence obtained by GPC is optimal, since it is obtained as a result of minimizing a cost function. In an embedded system with limited computation resources, the information of future control actions is very valuable, since it could be used, within the prediction horizon, to apply them under scarce measurements or excessive calculation time for that period.

In case of missing data in communication channel which becomes is to apply at the previous sampling instant, with the information received until instant k. For this strategy the prediction horizon are N_1=1 and N_2=N.

4.2.2 Generalized Predictive Control

The Generalized Predictive Control (GPC) is a Model-Based Predictive Control (MBPC) strategy. Their main characteristics are (Clarke et al., 1987):

- GPC use a process model of explicit form
- From minimize a cost function, is obtained the sequence of optimal control signals at every instant.
- Receding horizon strategy is applied. That is, although an optimal sequence of control is obtained, is only used the first control action signal of all of them, discount the others. At the next sampling

instant, the calculations are repeated again using the new information.

In the GPC design an index of quadratic cost is considered, expressed normally in the form:

$$J(N_1,N_2,N_u) = E\{\sum_{j=N_1}^{N_2} \psi_j \cdot [\hat{y}(k+j|k) - w(k+j)]^2 + \sum_{j=1}^{N_u} \lambda_j \cdot [\Delta u(k+j-1)]^2\} \quad (3)$$

The first term considers the error between the outputs and the references in the prediction horizon, whereas the second penalizes the control effort. ψ_j and λ_j are the weights of the output error and the control effort, respectively. $\hat{y}(k+j|k)$ is the prediction of the output in the instant k+i with the information available in the sampling instant k.

And minimizing this index with respect to ΔU analytically, the control law is obtained:

$$\Delta U = [G^T \cdot \psi \cdot G + \lambda . I]^{-1} \cdot G^T \cdot \psi \cdot [W - \Gamma \cdot \Delta U^{\mathrm{f}} - \mathrm{F} \cdot Y^f] =$$

$$= H_{aux} \cdot [W - \Gamma \cdot \Delta U^{\mathrm{f}} - \mathrm{F} \cdot Y^f] \quad (4)$$

and a prediction model:

$$Y = G \cdot \Delta U + \Gamma \cdot \Delta U^{\mathrm{f}} + \mathrm{F} \cdot Y^f \quad (5)$$

G, F and Γ are arrays with the coefficients calculated recursively (Clarke et al., 1987), ψ and λ are the arrays with the weights ψ_j and λ_j, and W is the array with the references.

The control law is transformed with the receding horizon strategy, as explained above. Considering this, a linear expression for the regulator is obtained.

4.2.3 Control and output postulates trajectories

Now the GPC must calculate the control actions postulated and the output trajectory for the applied prediction horizon. From prediction model matrices and the control law result a set of equations in differences (Camacho *et al.*, 2004).

From the control law (4), the values of postulated control actions for the control horizon are obtained:

$$\begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \dots \\ \Delta u(k+N_u-1) \end{bmatrix} = H_{aux} \cdot \left[\begin{bmatrix} w(k+N_1) \\ w(k+N_1+1) \\ \dots \\ w(k+N_2) \end{bmatrix} - \Gamma \cdot \begin{bmatrix} \Delta u(k-1) \\ \dots \\ \Delta u(k-n_t) \end{bmatrix} - F \cdot \begin{bmatrix} y(k) \\ \dots \\ y(k-n_a) \end{bmatrix} \right] \quad (6)$$

From the prediction model (5), the output trajectories for the prediction horizon are obtained:

$$\begin{bmatrix} y(k+N_1) \\ y(k+N_1+1) \\ \dots \\ y(k+N_2) \end{bmatrix} = G \cdot \left[\begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \dots \\ \Delta u(k+N_u-1) \end{bmatrix} + \Gamma \cdot \begin{bmatrix} \Delta u(k-1) \\ \dots \\ \Delta u(k-n_t) \end{bmatrix} + F \cdot \begin{bmatrix} y(k) \\ \dots \\ y(k-n_a) \end{bmatrix} \right] \quad (7)$$

Therefore, for every sampling instant k, and for the prediction horizon $N_1=1$ and $N_2=N$, the following trajectories are obtained:

$$Y = \{\hat{y}(k+1|k), \dots, \hat{y}(k+N|k)\} \quad (8)$$

$$U = \{u(k|k), \dots, u(k+N_u-1|k)\} \quad (9)$$

output and control trajectories are those that the supervisor control GPC will send to the local control in each period.

4.2.4 Light Node. Compensation of control action

Since it has been seen previously, this way to act has the problem of open loop strategy, with the usual problems as stability due to possible errors in the modelling of the plant and the inevitable disturbances in the measurement.

In order to minimize these problems, in case of using the postulated control actions calculates by the GPC, the *light node* makes modifications in the propose control actions, considering the discrepancy between the output calculated trajectory and the real output is applied.

$$e_y = \hat{y}(k+i|k) - y_{real}(k+i), \quad i = 1, \dots, N \quad (10)$$

$$If\ i < N_u - 1, then\ u = u(k+i|k) + K_{gain} \cdot ey \quad (11)$$

$$If\ i \geq N_u - 1, then\ u(k+i|k) = u(k+N_u-1|k) \quad (12)$$

where $\hat{y}(k+i|k)$ and $u(k+i|k)$ are elements of the Y and U vector, respectively. *Kgain* is the local compensator gain that is due to determine of empirical method, studying the influence of error "e_y" in the final control action. The propose strategy is shown in figure 6.

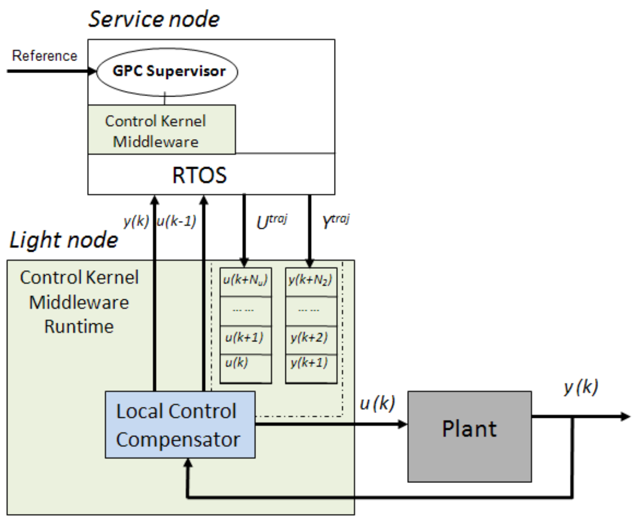

Fig. 6. Distributed control structure.

The advantage of these systems is that service node can make the supervision of several *light nodes*.

Considering that the maximum horizon calculated by the GPC corresponds to sampling instant $k+N_2$, it is necessary to design this horizon so that all the system dynamics is included, thus is case of missing all data makes sure that the output arrive at the reference.

In case of massive missing data, further the designed control horizon, it is necessary to apply a safe control strategy, that according to the controlled process it can consist of constant control action application or an emergency shutdown (Crespo *et al.*, 2006).

Finally, it is necessary to consider that when recovering the communications between *light* and *service node*, the supervisor control GPC must known the control action applied during missing data and the real output, for recalculate the trajectories with the real data.

4.2.5 Implementation Example

The plant tested is an electronic process of second order with a stable overshoot response in open loop. The transfer function is:

$$G_p(s) = \frac{6.818}{0.1021 \cdot s^2 + 0.9588 \cdot s + 7.818} \quad (13)$$

The parameters of the GPC are:

N_1=1, N_2=8, Nu=3, ψ=1, λ=0.1, T(z)=1 and Ts=0.05 seconds

Following the figure 6, a dsPIC microcontroller act as *light node*, it has been directly connected to process. As *service node* a XScale embedded computer has been used, it computes the GPC algorithm. The interconnection and data exchange between the nodes are carried out through the CAN communication bus (Coronel *et al*., 2005). The main characteristics of these nodes are described in (Martínez *et al.* 2007).

Basically, *light node* gets, via CAN, the control values computed by the GPC on *service node*, and apply them to the electronic process with its analogue outputs. The data values sent to *light node* are the present control action, two future predicted actions, and eight predicted output trajectories based on the GPC process model.

4.2.6 Results

The main advantage of this distributed system approach is its fault tolerance for sporadic error communication. As shown in the figure 8, when the control action is not received at the *light node* (data lost), it has to apply the predicted actions at the next sample times. If the communication errors are longer than control horizon, the *light node* has to apply the last control action value according to the predicted trajectory data. This is shown in the figures 8 and 9, where the data is lost during five sample periods (50 milliseconds).

Both figures are from the same experimental test, figure 8 shows the evolution of the process (system signal tries to follow the reference value), whereas figure 9 displays the control signal applied, the control value computed by the GPC and the difference between them.

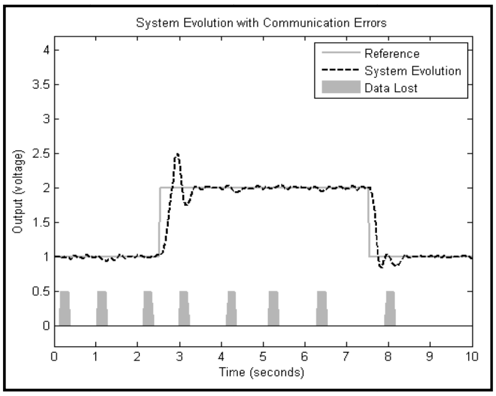

Fig 8. Process Evolution when some data is lost

The higher difference between the applied control value and the GPC computed value occurs when the data is lost just within the transitory response (third second in figure 9). Although this is the worst situation for loosing data, the control in the *light node* overcomes with the process evolution as it is shown in figure 8. The control in the *light node* corrects the action values considering the discrepancy between the calculated output trajectory and the present output, minimizing the output error.

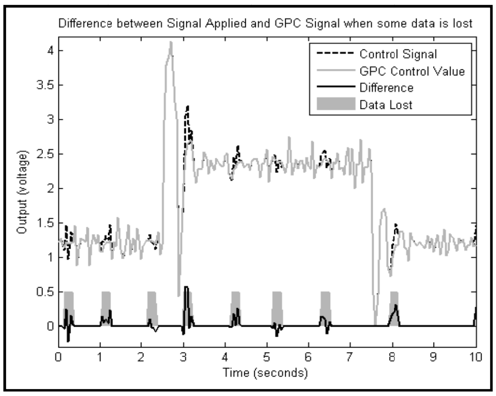

Fig. 9. Control Signal Applied, GPC Control Value computed and their discrepancy when data is lost.

5. CONCLUSIONS

In this paper a distributed control kernel model to complex control applications has been presented. Furthermore, from the implementation point of view, the proposed control model permits to perform a set of basic activities to ensure a safe operation of the system under control. A realization of a control algorithm GPC is described. The proposed distributed control scheme, through a combination of embedded systems, permits to ensure a safe and suitable operation of the system under control. This work is intended to be a proof of concept of some of the characteristics that are going to be implemented in the middleware kernel of the project KERTROL

REFERENCES

Albertos, P., Crespo, A., Simó, J. (2006). *"Control Kernel: A key concept in embedded control systems"*. In 4th IFAC Symposium on Mechatronic Systems.

Camacho, E.F. and Bordóns, C. (2004). *"Model Predictive Control"*. Ed. Springer.

CiA (CAN in Automation), (1996). *"CAN Application Layer for Industrial Applications"*, Documents No.: DS-201...207, v 1.1.

Coronel, J.O, F. Blanes, G. Benet, P. Pérez, J.E. Simó, (2005). *"CAN-based Distributed Control Architecture using the SCoCAN Communication Protocol"*, Proc. IEEE Int'l Conf. on Emerging Technologies and Factory Automation, ETFA'2005, vol 1.

Clarke, D.W., Mohtadi, C. and Tuffs, P.S. (1987). *"Generalized predictive control"*, parts I and II. Automatica, vol.23, nº2, pp 137-160.

Crespo, A., Albertos, P., Balbastre, P., Vallés, M., LLuesma, M. and Simó, J. (2006). *"Scheduability issues in complex embedded control systems."* Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design. Munich, Germany.

Lee, Edward A. (2006) "*Cyber-Physical Systems - Are Computing Foundations Adequate?"* Position Paper for NSF Workshop On Cyber-Physical Systems. Austin, TX.

Martínez, P.J., Coronel, J.O., Blanes, J.F., Simó, J.E., Albero, M., Benet, G. (2007). *"Low-cost Distributed Embedded Control Systems"*. In 4th IFAC Symposium on Mechatronic Systems.

MathWorks Inc (2006) MATLAB®, Simulink®, Real-Time Workshop®. The MathWorks Inc. www.mathworks.com

Microchip (2006) dsPIC™ Digital Signal Controller. *"dsPIC30F Family Reference Manual"* www.microchip.com

Yodaiken, V., Barabanov, M., (1996) "*Real Time Linux",* Linux Journal.