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

Middleware for Control Kernel Implementation in Embedded Control Systems

A. Fernández*, M. Vallés, A. Crespo**, P. Albertos **, J. Simó ****

*Departamento de Automática y Computación,
Instituto Superior Politécnico José Antonio Echevarría, La Habana, Cuba
(e-mail: adel@electrica.cujae.edu.cu)
** Instituto Automática e Informática Industrial
Universidad Politécnica de Valencia, Valencia, Spain. P.O.Box 22012 E-46071
(e-mail: mvalles, pedro@isa.upv.es; alfons,jsimo@disca.upv.es)

Abstract: Control tasks require a number of activities, not all of them with the same relevance and priority. The critical subtasks constitute what is denoted as the control kernel. The design of ECS should be structured, with a kernel unit providing the most basic features. Most of these activities, such as data acquisition or control action delivering, are common to a number of control loops implemented in the same CPU. In this paper, the architecture of a middleware (MW) for essential control activities to ensure economic, safe and reliable operation is discussed. It is specially designed for control purposes, interacting with the peripherals (sensors, actuators and communication channels), with the OS, and exchanging information with the bunch of control algorithm implemented in the application area. As part of the facilities of the middleware, some issues for improving in a transparent manner these characteristics are considered. Finally, an example of controller implementation by using this middleware is presented.

1. INTRODUCTION

Embedded Control Systems operate under highly changeable and uncertain environments with strong resource constraints. Control tasks require a number of activities, not all of them with the same relevance and priority. They are RT tasks of reactive nature, delivering control actions at prescheduled time instants. The critical subtasks constitute what is denoted as the control kernel.

Most of these activities, such as data acquisition (ADQ) or control action delivering (CAD), are common to a number of control loops implemented in the same CPU. Moreover, classical CPU operation implies a sequential behaviour.

Thus, in order to save resources and to be as fast as possible, some basic actions should be implemented at a very low level (close to the OS), with high priority.

Control kernel activities have been discussed in previous papers (Albertos *et al.*, 2007), and different options for its implementation have been also presented (Albertos *et al.*, 2006). One of them is to establish a middleware specially designed for control purposes and interacting with the peripherals (sensors, actuators and communication channels), with the OS, and exchanging information with the bunch of the control algorithm implemented in the application area.

The paper is structured as follows: first, the main features of the ECS as well as the facilities linked to the control kernel are reviewed. Then, the priorities of the control subtasks and their interaction are discussed. Their implementation in a middleware area is then presented. Some alternatives to guarantee the safe operation of the control are proposed. In particular, special attention is paid to the local control structure, to guarantee the control action delivering in the event of resources shortage: computation time, data availability or emergency operation. Finally, these concepts are applied to the control of a robot with a flexible joint.

2. EMBEDDED CONTROL SYSTEMS

The strong increasing presence of embedded systems (ES) in products and services creates huge opportunities for the future in different areas such as industrial control systems, avionics, health care, environment, security, mechanics, … (Chinook, 2004). Thus, there is a growing scientific interest on conceptual and practical tools for their development (Dreamteach, 2002). In particular, their use in control applications is becoming very popular.

RT control applications on ES require the best use of the available computation resources. Among the main advantages they offer are the reduced price and size, broadening the scope of possible applications: mass-production systems due to the cost reduction and specific accurate applications for their reduced size and high performances. But the most important problem is the limited computational capabilities they can use because it is well known that, in general, short sampling periods and non-delayed control actions allow for better control performances.

So, one of the most important issues related with ES in control applications are related with the reliable and optimal use of their computational resources and what the resource shortage involves in the design and implementation of the

control algorithms. So, it seems interesting to extract the main characteristics of the ES applicable to ECS and the problems they pose from the point of view of interaction between computers, communication and control.

The basic characteristics of ECS are compact and reduced size, autonomy, reconfigurability, safety, fault-tolerant and missing data operation. As for any control application, RT issues are crucial to guarantee the system response under any operating conditions.

Specifically, at the control applications, the use of embedded systems generates problems related to their implementation, the computational load and resources sharing and the control performance degrading. A detailed description of these can be found at (Albertos, *et al.*, 2005).

3. CONTROL TASK MODEL

In most control applications it is quite common that various processes (multiprocess), with many control variables and sensors (multivariable), involving many control loops (multiloop) and operating at different sampling rates (multirate) exist. If these applications are implemented over a monoprocessor computational system, several tasks will be implemented sharing the same processor and so, depending on the priorities and the scheduling used, a variable delay (jitter) appears affecting the nominal execution times.

Several real-time task models have been proposed for, or motivated by, control applications. Among them, the most commonly used is the hard real-time model. However, adaptive task model, subtask models, and weakly hard models are to some extent also relevant in this context. In discrete-event based control, though, truly hard deadlines are more common. These controllers have to respond to external events within certain deadlines by providing the correct output actions.

Subtask scheduling models are well suited for control. A control algorithm can naturally be divided into two or more parts. In this way the input-output latency caused by task execution can be minimized. Alternatively, the sampling and the output are also considered as separate subtasks, possibly being separately scheduled.

Several sub-task scheduling models and offset-based scheduling models have been proposed in the scheduling community. These include the multi-frame model (Baruah *et al*. 1997) and the serially executed subtask model (González-Harbour *et al*., 1994). The objective is not to achieve better control performance but to enhance the schedulability of the task set under fixed-priority (FP) scheduling.

In (Crespo *et al.*, 1999) and (Albertos *et al*., 2000) a subtask scheduling of control tasks has been proposed. Each task is decomposed into three parts with different priorities: the input operation (medium priority), the control computation (low priority), and the output operation (high priority). The goal of the scheduling design is to minimize the input-output jitter and improve the control performances by reducing the variable delay of all tasks. In (Balbastre *et al.*, 2000) the partitioning method was developed for EDF scheduling policy. The advantages of the appropriate variable delay reduction related to a control parameter such as the control effort were reported in (Albertos *et al.*, 2000).

In this paper this task decomposition is also used and the priorities for each one are assigned taken into account that, from the point of view of a control application, sending a control action at the corresponding instant is the most important task to be done. Thus, the final task should have the highest priority, followed by the initial task, afterwards the mandatory one and finally the optional one.

4. MIDDLEWARE ARCHITECTURE

4.1 Middleware objectives

The generally accepted definition of Middleware is computer software that connects software components or applications. It is used most often to support complex and distributed applications.

Some works have been reported related with the implementation of middleware for distributed real-time and embedded computing. The main objective is to develop architectures allowing the use of different communication networks and computational platforms. Most of them are used for the case of embedded mobile devices ((Cassinis, 2007) (Lakhotia *et al.*, 2006) (Wu *et al*., 2007)) but some of them are related with control applications where some sensors and actuators information is shared over a network ((Baliga, 2005), (Wang *et al.*, 2007) (OMG DDS standard)). Several kinds of middleware are proposed as composition of building blocks (Schmidt et al. 2002). ACE (ACE Project) provides communication services and portability across different operating systems and hardware platforms. Real-time CORBA provides efficient and predictable middleware structures and services, and adaptive QoS management policies. An example of open source implementation of RT-CORBA is TAO (Gill et al. 2002). PolyORB (Thomas et al. 2004) as TAO, are geared toward providing predictable timing of end-to-end method invocations. Also, additional works on PolyORB provide mechanism to high integrity real-time systems (Zamorano et al. 2007). In this paper, a middleware for control applications is proposed.

The control application can be implemented either on a single computational element with different computational capacity and resources availability (from a PC to a microcontroller), or on a distributed computational network with different communication protocols (CAN, Ethernet, wireless). So, the final execution platform could be very different depending on which part of the control application is being implemented.

The general objective of this middleware is to support all these possibilities.

So, the general objectives of the middleware are:

- To provide a flexible interface for handling the acquired information. This includes dealing with the registration and deregistration of sensors for a control application at runtime. In this way, reconfigurability on changing environments can be ensured. The same for the actuation elements.

- From the point of view of the control system, it is needed that the system is being controlled in all situations. So, each element of the control system has to be capable of, autonomously, control its part of the control system.

- To be robust from the control perspective, some supervisory functions are needed for detecting and adapting to changing situations or, more basically, for detecting miss functionalities of sensors, actuators, networks, etc …

As the final platform over which the controller will be implemented can be very different in terms of computing resources, a layered structure is needed in such a way that the basic requirements will be accomplished at the most elementary part of a control system. This middleware has not been designed to provide a complete support of object oriented distribution. It has been focused to provide specific management mechanisms to control applications. These functionalities, defined in the Control Kernel (Albertos et al. 2006), are not supported by general purpose middleware (ACE, TAO, PolyORB, etc.), the main advantage being the specific control services included in the middleware.

This paper has focused on defining the basic layer that every element of the control system should have. It is supposed that there is a single computational element with local access to the sensor and actuators elements.

4.2 Middleware architecture

The basic architecture of control tasks executions has been described in previous papers (Albertos *et al.*, 2007), and is given according to the Figure 4.2.1

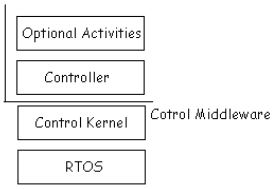

Figure 4.2.1. Decomposition levels

The middleware should use the maximum capacities of a RTOS in order to execute control applications. It should group together the common functions of the control tasks to guarantee the correct use of the available resources at each instant of time and to send the best control action to the process. Be aware that control algorithm is not part of MW because it is particular of the process to be controlled.

The interaction between the MW and the controllers executed in a mono-processor environment are shown in figure 4.2.2.

Using the services of RTOS, the MW can acquire the measurements of the physical sensors and deliver the control action to the physical actuators, both located outside.

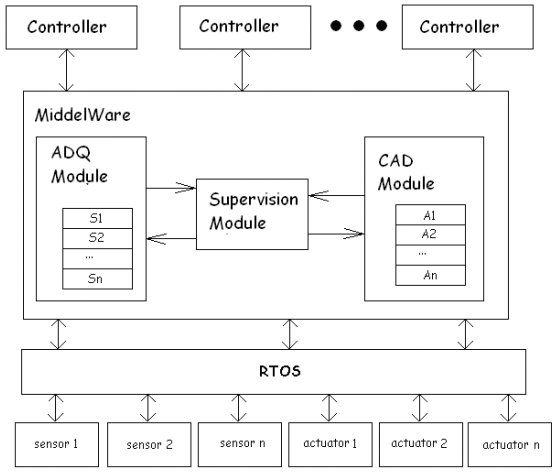

Figure 4.2.2, Middleware architecture

Following the previously proposed scheme of control tasks model, inside the MW there is a module to coordinate the process of acquisition of data, and another one to coordinate the delivered control actions, independently of to which controller it belongs to. S1, S2 … Sn and A1, A2 … An, are the memory blocks to store data and configurations corresponding to the sensors and actuators.

The supervisor module verifies the sensors and actuators state, the available resources and takes the decision of the correct action to be delivered to each controlled process.

The interactions between the MW and the controllers can be seen in figures 4.2.3 a and b. Three parts are considered: data acquisition, control action delivering and supervision.

Data acquisition (figure 4.2.3 a)

Each physical sensor has a structure inside the ADQ module to keep the parameters of each external variable. This structure will be named S = {T, B, C, E}, where T is the sampling period, B is the buffer to store the n last values, C the controller to which belongs the measure and E the sensor state. The last one is a very important parameter to decide which action will be delivered. It can be in one of three possible states:

- *Fail:* the acquisition signal is not received from physical sensor, this signal is identified like noise, using an alarm, or the signal corresponds to the limit of the scale.
- *Event:* the same value is received n times.
- *No_fail:* none of the previous options happen and the operation is normal.

The ADQ module has one subsystem to keep the measurement state and to determine the acquisition quality. To evaluate the quality this module computes the concept of data acquisition interval DAI (Crespo *et al.*, 1999). It expresses the variable time delay in acquire a measurement from sensors due to the processor sharing. This module calculates for each measurement the maximum DAI, the average DAI and its variance.

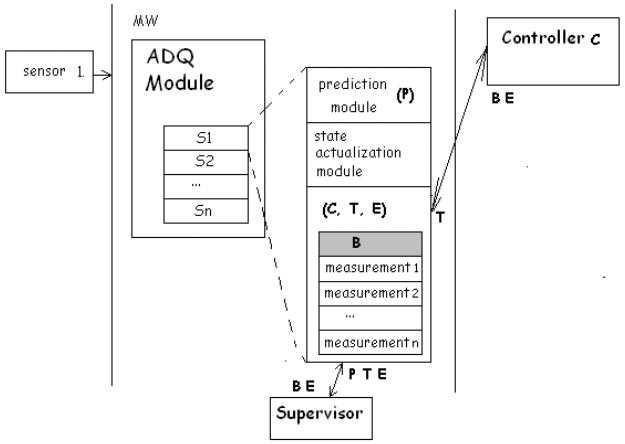

Figure 4.2.3 a, Acquisition module.

Control action delivering (figure 4.2.3 b)

Each physical actuator has a structure inside CAD module to keep the parameters of each control action. This structure will be named A= {T, O, B, R, C, E}, where T is the sampling period, O is the offset, in absolute time, between delivering of the action and acquisition of data, B is the buffer to store the n last values, R is the buffer to store the n future references values, C is the controller to which the action belongs and E is the actuator state. The last one is also a very important parameter to decide which action will be delivered and, again, it can be in one of three possible states:

- *Fail:* it is detected that the action has not been delivered
- *Event:* the computed action is delayed, and the stored actions are old
- *No_fail:* none of the previous options happen and the operation is normal.

The CAD module has one subsystem to keep the actuator state and to determine the delivering actions quality. To evaluate the quality, this module computes the concept of control action interval, CAI (Crespo *et al.*, 1999). It expresses the variable time delay in delivering the control action due to the processor sharing. This module calculates for each action the maximum CAI, the average CAI and its variance.

Supervisor

This module makes the decisions in the MW. To carry out the supervision it should know the state of the sensors and actuators corresponding to one controller; the control state and the available resources.

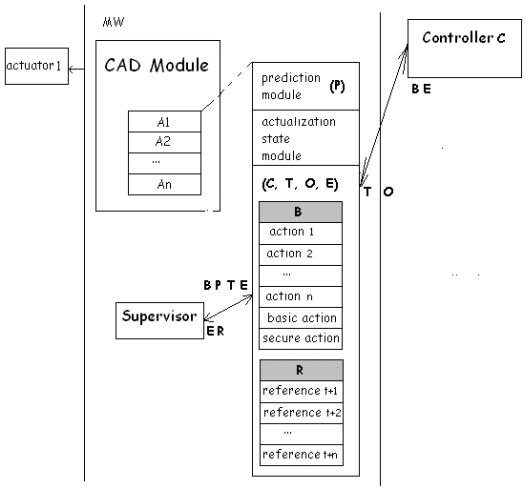

Figure 4.2.3 b delivering action module

Every controller can be associated to *s* sensors and *a* actuators, the sensors will be classified in one main sensor (ms) and several secondary sensors (ss). The sensor indicated as *ms* is the one that measures that variable that allows computing a basic control action on the process.

The possible decisions taken by the supervisor are:

1. If *ms* is in failed state (no transitory) it proceeds to deliver the secure control action.
2. If *ss* is in failed state (no transitory) it proceeds to compute and deliver the basic control action.

4.3. Quality and Robustness of the basic controller

Besides the previously commented functions for ensuring the right functionality of the controller based on the developed middleware, some functions have been added in order to improve the quality and robustness of the control actions delivered by the controller. These ones are the following:

In some situations, the control actions could be evaluated by a complex (optimal) algorithm that could be implemented remotely or locally but subjected to the computational resources availability. If the control algorithm has not provided the next ones, these could be projected from the present ones.

If the input signals do not vary, it is supposed that the system is in a stable state, so the present control action is projected for the future and the controller is put in a sleep mode. In this way, computational resources as well as battery are saved. This module is activated a short time before changing the signal to the failure state; their main purpose is to assure a correct operation of the system in transitory failure situations. See the prediction module in figure 4.2.3 a.

Another possibility, also related with this, is to enlarge the period of the actuator. The period for the acquisition is kept for detecting variations at the input signals but the buffer is filled only with the data for the same period of the actuator

(the last one or a mean of those between two samples of the actuator). This module is activated a short time before changing the actuator to the failure state; their main purpose is to assure a correct operation of the system in transitory failure situations. For permanent failures the basic action algorithm or the secure action registered to the controller will be invoked. See the prediction module in figure 4.2.3 b.

The basic control action implemented up to now in the MW is a proportional control that is registered with the controller. The secure action should be taken according to the controlled system; it can be to close a valve, to stop a motor, etc.

The middleware interface is responsible for the interaction between the controllers and the sensors/actuators. The controller must ensure the computing of the N future control actions and must send them to the buffer B of the related actuator at each sample interval. It must also get direct information about the state of the acquisition process. It must be able to ask for a state change over the MW, depending on the quality of ADQ, AC and the available resources. It must be also able to ask for a change on the sampling period and offset for the acquisition and the AC and it must ensure that the N future references will be sent, allowing MW to compute the basic control.

5. EXAMPLE

The MW has been designed and implemented as a functions library. The user must make the calls to the necessary functions, according to the following order:

1. MW Configuration.
2. Sensors, actuators and controller registrations.
3. Start control procedure.
4. Stop control procedure.

In order to illustrate how to use this library we took the same example mentioned in (Albertos *et al.*, 2007), a robot with a unique flexible join. The dynamical equations are:

$$J\ddot{q}_2 - K_s q_1 = \tau$$
$$D\ddot{q}_1 + D\ddot{q}_2 + K_s q_1 = 0 \qquad (5.1)$$

where q_2, q_1 are the positions and J y D are the inertia of the actuator and the terminal element, respectively. In this setting, the torque (τ) applied to the actuator generates a movement that is transmitted to the terminal element by a spring with elastic constant K_s.

From these differential equations, it is possible to model the flexible robot's behaviour in the space of states. The following matrix was used to control the system with a sample period T=0.01 seg.

$$[K1\ K2] = [-5.5032\ 0.06223\ 0.053\ 7.17334] \qquad (5.2)$$

To build the controller the user must write a function that evaluate these equations and send to the MW the computed action. The general form is the following one

```
void user_controller(void)
{
        value_q1 = read_data_sensor(idq1);
        value_q2 = read_data_sensor(idq2);
        evaluation of the equation 7.2
        u = calculate_acción(value_q1, value_q2);
        send_action(idu, u);
}
```

In order to carry out the supervision, the MW assumes the variable q_1 as the main sensor (ms) while q_2 is considered as the secondary sensor (ss). This should be defined at the time the user registers the sensors, actuators and controllers. The parameters used in the example are:

- t=10 ms: sample period to acquire data and deliver control actions
- p1=1, p2=2: priority with respect to others actuators or sensors registered.
- o=3 ms: offset in time between the data acquisition and deliver control action.
- k=1.5: proportional constant, in the basic control action.
- s= 0 volt: secure action.
- buffersize = 100: ADQ and CAD buffer size.
- wt =10 T: Integer determining the windows time, must be a multiple of the period, to verify failure conditions.
- h= 0.1 rad: hysteresis value to return to normal conditions
- d=0.2 rad: maximum variation allowed in measure value, between *t* and *t+1*.
- cd=10: number of times that a maximum change of value *d* is tolerated before throwing the message: fail.
- s= [6 0 -6] rad: 3 real elements vector that register the instrument scale limit.
- cs=10 number of times that a scale limit value *s[i]* is tolerated before throwing the message: fail.
- r=100: number of times that a same value is repeated without throwing the message: event.

To use the middleware function, the user has to configure the used resources in the following way:

```
procedure configuration {
//allocating resources and failure handlers
   idq1=register_sensor(t,p1,buffersize);
   idq2=register_sensor(t,p2,buffersize);
   register_fail_detection(idq1,wt, h, cd, d, sc, p , r);
   register_fail_detection(idq2,wt, h, cd, d, sc, p , r);
   idu=register_actuator(t,o,p1,buffersize);
 register_controller(user_controller, [idq1 idq2] ,idu);
// start control
  start_controllers();
}

procedure finish {
// free the resources allocated
  stop_controllers();
}
```

The graph shown in the Figure 5.1 is taken directly from a process controlled with the MW. Between t=10s and t=25s, the system is stable. In this interval, the control action period can be increased, only if there are not changes in the

measures. In t=26 the supervisor must return to the previous period.

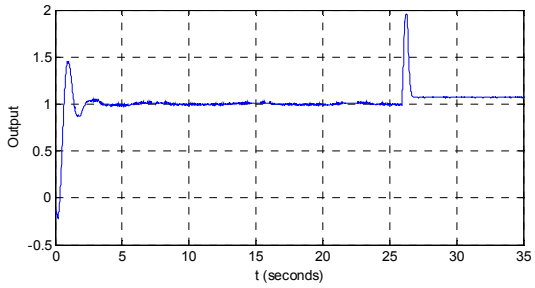

Figure 5.1 change of simple periods

The measurements of the process should be kept in two buffers, one with data got at the current rate and another one at the previous one, to guarantee that the transition process in t=26s is transparent.

6. CONCLUSIONS

A middleware for control applications has been presented where the basic functionality for an elementary node of a control application has been defined. The case of a local controller has been considered in order to clarify the structure at the most basic level. The case of distributed applications will be considered in the next development based on the proposed architecture.

Some functions for ensuring the right performing of the controller even in autonomic working mode have been also introduced. These ones have been complemented with others that introduce some quality and robustness considerations.

Finally, an example for the control of a robot with a flexible joint has been presented where the developed middleware has been used. It has been described the way that the controller will be implemented over the middleware and some experiments has been shown for illustrating how the middleware works when a sensor does not work.

REFERENCES

Albertos, P., A. Crespo, M. Vallés and I. Ripoll, P. Balbastre (2000). RT Control Scheduling to Reduce Control Performance Degrading. *1 Proc. 39th IEEE Conference on Decision and Control.* Sydney (Australia).

Albertos, P., A. Crespo, M. Vallés and I. Ripoll (2005). Embedded control systems: some issues and solutions. *16th IFAC World Congress*, Praga (República Checa)

Albertos P., A. Crespo , José Simó. Control Kernel: a Key concept in Embedded Control Systems. IFAC Conf. on Mechatronics. Heidelberg, 2006.

Albertos, P., M. Vallés, A. Cuenca and A. Valera. (2007). Essential control in Embedded Control Systems. IFAC Symp. On Cost Oriented Automation. Havana. Cuba.

Crespo, A., I. Ripoll, P. Albertos (1999). Reducing delays in RT control: The control action interval. *In Proceedings of the 14th IFAC World Congress*, pp. 257-262.

Balbastre, P., I. Ripoll, A. Crespo (2000). Control task delay reduction under static and dynamic scheduling policies". *In Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications*.

Baliga, G., P. R. Kumar (2005). A Middleware for Control Over Networks. Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. pp 482-487 ISBN: 0-7803-9567-0

Baruah, S.K., D. Chen and A.K. Mok (1997). Jitter concerns in periodic task systems. *18th Real-Time Systems Symp.*

Cassinis, R., F. Tampalini (2007). AMIRoLoS an active marker internet-based robot localization system. *Robotics and Autonomous Systems*, 55(4), pp.306-315

Chinook.webpage: http://www.cs.washington.edu/research/chinook/links.html

Dreamtech Software Team, (2002). Programming for Embedded Systems: Cracking the CodeTM. J. Wiley.

Gill C, Gokhale A, Natarajan B, Schmidt DC, Wang N. "TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems", IEEE Distributed Systems Online, 3(2), February 2002.

Lakhotia, A., S. Golconda, A. Maida, P. Mejia, A. Putambeker, G. Seetharaman, S. Wilson (2006). CajunBot: Architecture and algorithms. *Journal of Field Robotics*, 23(8), pp. 555-578.

OMG DDS standard. OMG standard specification of Data Distribution Service for Real-time Systems specification, version 1.2. http://www.omg.org/technology/documents/formal/data_distribution.htm

Schmidt DC, ACE project. http://www.cs.wustl.edu/~schmidt/ACE.html

Schmidt, D. C. "Middleware for real-time and embedded systems," Communications of the ACM, vol. 45, pp. 43-48, Jun 2002.

Vergnaud, T., J. Hugues, L. Pautet, and F. Kordon. PolyORB: a schizophrenic middleware to build versatile reliable distributed applications. In Proc. of the 9th Int. Conf. on Reliable Software Technologies Ada-Europe 2004 (RST'04), volume LNCS 3063, pages 106 - 119, Palma de Mallorca, Spain, Jun 2004. Springer Verlag.

Wang, XR., YM. Chen, CY. Lu, X. Koutsoukos (2007). FC-ORB: A robust distributed real-time embedded middleware with end-to end utilization control. *Journal of Systems and Software*, 80 (7), pp. 938-950

Wu, ZH., Q. Wu, H. Cheng, G. Pan, MD. Zhao, J. Sun (2007) CudWare: A semantic and adaptive middleware platform for smart vehicle space. *IEEE Transactions on Intelligent Transportation Systems,* 8(1), pp. 121-132

Zamorano J.,de la Puente J.A., Hugues J., Vardanega T.. Run-time mechanisms for property preservation in high-integrity real-time systems. In NICTA, editor, Proceedings of OSPERT 2007 - Workshop on Operating Systems Platforms for Embedded Real-Time applications, pages 65-68, Pisa, Italy, Jul 2007.