# Control Co-design: Algorithms and Their Implementation[*]

Pedro Albertos[1,**], Alfons Crespo[1], José Simó[1], and Adel Fernández[2]

[1] Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia
P.O. Box. 22012, E-46071, Valencia, Spain
{pedro,alfons,jsimo}@aii.upv.es
[2] Departamento de Automática y Computación
Inst. Superior Politécnico José Antonio Echeverría, La Habana, Cuba
adel@electrica.cujae.edu.cu

**Abstract.** This paper deals with the new approach in the design of hard real-time control applications where the control requirements as well as computing and communication constraints should be jointly taken into account to design a control application. Resource distribution and limitation, safety requirements and autonomy lead to the need of the so called co-design, where the integral problem of the design of the control structure, algorithm and its implementation should be tackled together. Along these lines, after a motivation, the interlacing between both design issues is analyzed and new concepts and architectures are proposed.

**Keywords:** real-time constraints, control safety, control structure, embedded control systems, networked control systems, event-based control systems, control kernel, control effort, time delays.

## 1 Introduction

Traditionally, control designers and software engineers work separately. The former conceive the control algorithms based on the required performance and the process knowledge, regardless of their subsequent implementation [10], whereas the software engineers deal with the control code without looking after the impact of the code execution in the control performance [7]. But their activities are interlaced and both designs should be jointly treated, mainly if the control tasks share resources with some other activities and these resources are limited. The real-time (RT) control design and implementation should be reviewed from both perspectives [14], [28] and a co-design approach should be adopted.

To this end, global requirements on control applications in time critical environments, such as automotive, aerospace or flight control, where multiple

[*] This project has been partially granted by Consellería de Educación under PROMETEO project number 2009-0268 as well as CICYT project SIDIRELI: DPI2008-06737-C02-01/02.

[**] Currently on leave at the Industrial Control Centre, University of Strathclyde, Glasgow, granted by the Ministerio de Educación (PR2009-0268).

interactive control loops are implemented, should be reviewed to extract their main requirements. Special attention should be devoted to new and widespread control scenarios where the controller is not anymore implemented in a dedicated computer without resources constraints, but sharing and competing for computing, storage and communication facilities with several other tasks. Embedded control systems [3], networked control systems [16] and event-based control systems [9] challenge the design of the control and its implementation where architectural issues play a relevant role in the controlled system performance [21]. Main issues in control co-design [8], other than control algorithms themselves, are the communication (networking) issues, the hybrid behavior, RT constraints in computing and scheduling, multi-mode operation and safety constraints. There is a need of support for RT activities, device drivers access, fault tolerance and distribution [20] with special emphasis on the minimal use of hardware and software. In this context, new software development models and middlewares ([27], [11]) are proposed to deal with quality of service of control performance as well as computing, communications and power resources availability. The ultimate goal of this technology is to allow the separation of complex control systems design from the RT tasks dynamic reconfiguration. Some key concepts interacting with both, the control performance and the control implementation, such as the control effort [5] or the control kernel [2] are emphasized and some general directions in the co-design are summarized.

Let us consider some examples:

**Automotive.** The so-called electronic car should rather be termed as automatic control car. A number of devices are located everywhere in different parts of the car to measure speed, temperature, flow, lightness, sliding... Initially most information provided by these devices was used for monitoring, to be displayed to the driver. At present instead, all these measurements are

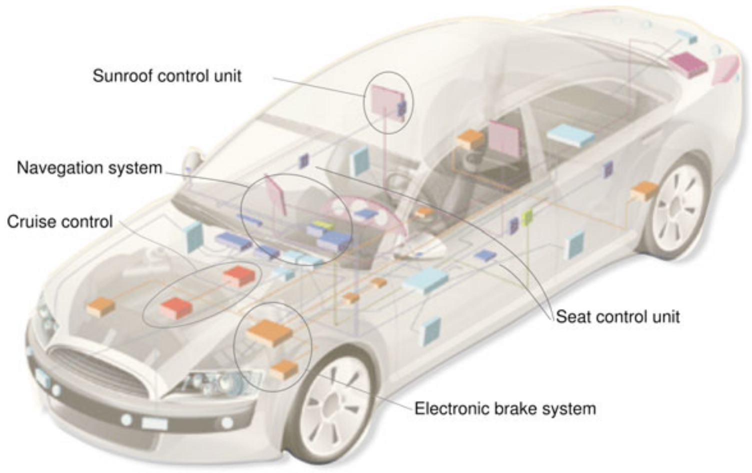

**Fig. 1.** A car with multiple distributed sensors

increasingly used to control many processes. They include servo braking (even brake-by-wire), computer assisted steering, traction control systems, active suspension, besides all the climate control, and invisible engine control, fuel efficiency and pollution control systems and the trusted cruise control. Some cars include collision avoidance systems and automated parallel parking. The information is distributed, and so is the actuation combining data from different sensors to elaborate the control actions (see Figure 1). And there are several local control units but there is a central control unit to coordinate all the activities. Anti-sliding control, speed control, engine control... do not work independently and according to the time and resources availability, the "best possible" control action is sent to the actuators.

The control, computer and communication network infrastructure in a car is substantial. At the present state of the art it may be viable to consider an internal wireless network rather than a wired network, purely from a fuel efficiency point of view (less cables = less weight = more fuel efficiency).

**Aerospace and flight control.** The continuous improvement in aerospace control should rely on integrated control systems seeking a variety of goals: flight control, control of power units, engine controls, utility systems, data fusion and concentration... All this requires a modular approach to design with upgradeable units and reusable components providing a fast and reliable operation, fault tolerance, reconfigurability and minimal resource requirements. Working in a harsh environment, the control must operate under different scenarios with variable availability of signals, power and general resources. In this context, special tools to rapid development and testing of the control solutions are required: design requirements translation at different levels, control design, code generation and testing.

Platforms to develop and test embedded control solutions have been proposed elsewhere [12]. We have built a very modular system involving the use of simulation facilities (Matlab®), RT development tools (Linux and Partikle[1]) and simple communication facilities to design and test control strategies for helicopters.

**Industrial systems.** Applications in the process industry are not in general as time demanding as the previous ones, but the number of signals, the distribution of subprocesses, sensors and actuators and the need to reduce the wiring and increase the reliability and safety of the applications also ask for a joint design of the control and its implementation. Networked and coordinated control are crucial issues in the process industry.

[1] PaRTiKle is a small footprint RT operating system designed for use in embedded systems. PaRTiKle [24] is especially well suited, but not limited, to work in combination with XtratuM technology. XtratuM [18] is a bare-metal hypervisor for RT systems that provides spatial and temporal isolation to many operating systems running as partitions. Several partitions can run simultaneously under RT constraints on top of the hypervisor. It is especially useful when a strong isolation of critical code is required.

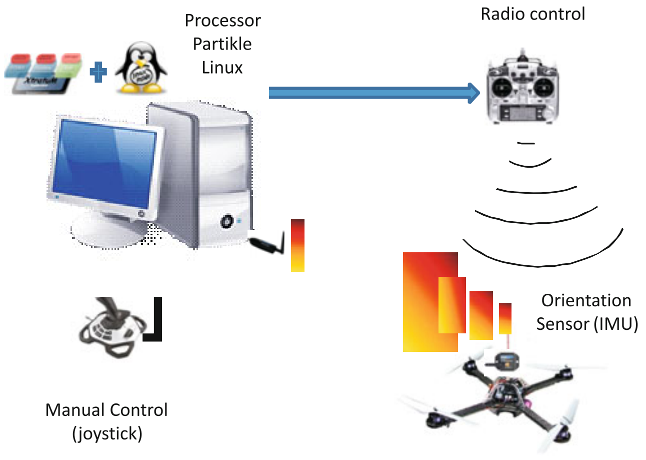

**Fig. 2.** A platform for prototypes development

The paper is structured as follows. In the next section, the computing requirements for control applications are reviewed. Then, the control algorithm requirements for an efficient RT implementation are discussed. New control scenarios with a space distributed operation have been developed recently. This is the topic of the next section, where networked systems and control over the network raise new challenges. Merging altogether, the need of simultaneously design the control and its implementation, the co-design, appears as the logical frame for the design of most RT control applications in the near future.

## 2 Computing Requirements for Control Applications

A standard periodic control loop implementation written in Ada 95 has the scheme shown in figure 3. The code is composed of two parts: specification and implementation. In the specification part the parameters associated with the task (initial-time, period and phase) are defined, while in the implementation part an infinite loop that executes at each period the actions related to the control (get external values, calculate the action value, send it, and compute global state) is included. This sequence may be changed or some tasks can be skipped according to the availability of resources.

This code corresponds to a classical control scheme where the computation of the control action can be undertaken using traditional techniques. However, more and more applications require complex computation and the use of exhaustive algorithms (unbounded algorithms) that can compromise the responsiveness of the system. If this is the case, the computation time should be split into a fixed (mandatory) part and an optional part (improvement).

The main requirements on SW/HW to run a control application are:

```
task Level_Control (initial_time, period, phase: TIME);

task body Level_Control is
level : Sensor_Value;
action : Actuator_Value;
reg : Regulator;
next_period : TIME;       -- period task attribute

begin
Define_Regulator(reg, par1, par2, ...);
....
delay until (initial_time + phase);
next_period := initial_time + phase;
loop
   level := get_level_sensor();
   Regulator_evaluate(reg, level, action);
   -- operations to improve the regulator results
   send_actuator(action);
   -- operations to evaluate the global state
   --operations to prepare the data for the next sampling
   -- operations of updating the global data base
   delay until next_period;
   next_period := next_period + period;
end loop;
...
end Level_Task;
```

**Fig. 3.** Standard control-loop implementation in Ada 95

– To have a quick an secure dispatch of a control action.
– To get a basic "picture" of the current situation.
– To compute a simple and fast control action to be improved if resources are available.
– To switch to the appropriate control mode, based on the resources availability.
– To reconfigure under detected faults.

To always provide a control action to the process, there should be a basic task organizing the back-up pile of control actions, by either pre-assignment or as a result of a batch computation. For instance, in model predictive control the algorithm computes a sequence of control actions to be applied at the next time instants, unless they are updated at the next sampling period. This default backlog should be stored and used, if no option to get better suggestions is available.

Not all the signals being treated have the same relevance (see next section). Thus, the system should give priority to those signals providing the fundamental knowledge about the current situation and the actions to be computed and delivered. In this sense, on-line rescheduling should allow to give the highest priority to these signals and their related treatment. This will ensure a safe operation of the control system.

In order to keep the process under control at any moment, a control action should be delivered at due time to the process. To that end, simple and fast control algorithms, probably not too performing but providing an adequate response, should be implemented with high priority, to be executed at the time required by the controlled process. If there is more time or, in general, more resources, a better action could be evaluated and the simplest one would be discarded.

Task mode management is related to the organization of tasks in different modes of operation. Several tasks included in a mode cooperate in the system control when some external conditions stand. For instance, under normal operation of a cruise navigation control, tasks involved in this mode take control of the different control loops, visualization and monitoring. If the system state changes and the system has to be managed in a different way, a mode-change event is raised and the operating system should stop the tasks involved in the previous mode and start those involved in the new mode. Tasks can be included in several modes with the same or different timing constraints. A mode change protocol is the method to implement this task switch. Protocols have to be efficient guaranteeing the system schedulability during the change phase [13], [17], [26].

Fault tolerance management is related to the detection and management of abnormal situations, such as missing data, emergency control or components fault. Error detection is a service that should be regarded as a basic functionality to achieve fault tolerance. Error management should advise in detail how the error or fault has to be handled, but it is more application dependent. Fault tolerance involves both questions, the detection of faults and, depending on the fault nature, its management or its propagation to the application.

RT control applications are paradigmatic applications where hard RT issues should be taken into account. They are reactive systems, gathering information from the environment, processing it and providing some actions in due time. And the information processing is done by sharing many resources with some other applications which also can be as demanding as them. Moreover, these applications may run in uncertain scenarios where operating conditions may change.

New trends in the control implementation regards infrastructure abstraction by means of middleware (Mw) components. Communication Systems such as DDS (Data Distribution System) [23],[22] provide the abstraction of communication details driving the data flow by specified QoS (Quality of Service) requirements. Data-centric approaches like DDS are specially well suited to develop event-based control systems (see Section 4.3). Using a publish/subscribe model, control applications can optimize the bandwidth usage but the control theory required to develop this kind of applications needs further development and maturity. In a similar way, the execution environment can be provided by middleware by using virtual machine technologies, i.e. RT Java (RTSJ).

Objects and agents can be deployed in a distributed execution platform also with the support of specific middleware like ORBit, PolyORB, RT-Corba or ACE+TAO[2]. In a similar way some research is focused on the definition of a

[2] These open source middlewares can be downloaded from the corresponding web pages, like [25].

middleware abstraction of basic control services: the Control Kernel (see Section 5). This abstraction provides an execution environment to deploy control applications.

In summary, let us consider the final picture of a software infrastructure supporting control applications as a bundle of interacting middlewares (Real-Time and Control Kernel) with the following desirable features:

– Real-time Mw: Data-centric and event-based communications driven by QoS and QoC (Quality of Control), specific time meta-information (time-stamps, temporal fire-walls, actual delays), Real-time Object Brokers, Redundancy support, Execution introspection in terms of QoC, delays and resources involved (bandwidth, computing time, power).
– Control Kernel Mw: Sampling, acting, signals holding, controller's switching management, default and "emergency" actions support, QoC management.
– Operating System: Real-time scheduling with dynamic load support, power management, fine-grained timing.

## 3 Control Algorithmic Requirements for RT Implementations

The execution of a control algorithm is not a unique activity and its parts may have different treatment, also depending on the environment. Let us discuss in some detail the issues involved in the RT implementation of the control algorithm.

### 3.1 Control Activities

In classic control design there are a number of basic assumptions about the control implementation [6], [7]:

– The data acquisition system is providing the required data.
– The actuators' drivers timely deliver the control actions.
– The CPU computes on-time the control action.
– The data required to perform the computations are stored in the memory.
– The sampling pattern is regular (constant, synchronous and uniform for any control task).
– The control goals as well as the control algorithms are fixed and well defined.
– Power supply is guaranteed.

It is difficult to guarantee the fulfilment of these assumptions in autonomous distributed control applications.

In digital control, the code to implement the control algorithm only takes a few lines but there are many other control related activities which are crucial to properly implement the control. The activities carried out in running a control algorithm should be analyzed and those which are more critical should be guaranteed under no matter which conditions the system is operating. Moreover, to run a control application in a safe mode, the following requirements should be fulfilled:

– A safe control action should be delivered at any required time to the process. This action may be the result of a detailed computation or simply a safe back-up command such as: *do nothing*.
– A supervisory control must make executive decisions and propose actions such as: *switch controllers*, *disconnect*, etc.
– A control action should be computed based on gathered data and a predefined algorithm: *On-Off*, *PID*[3], *Robust*, *Adaptive*, etc.
– Some data should be recorded, displayed, stored, updated.
– Communication links with other activities should be provided.

Not all activities have the same importance. It is evident that if there is no reaction to the process the computed control action becomes irrelevant.

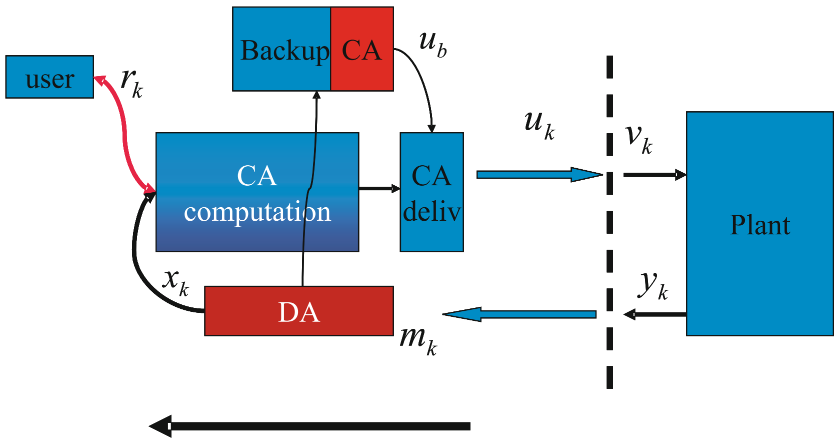

**Fig. 4.** Control activities

The activities shown in figure 4 can be ranked in order of relative importance as follows:

1. Assure the execution of a control action (CA) delivering. At least,
   – a safe back-up, or
   – a back-up based on previous (stored) data (defining the process situation).

   That is, there is always an action to be sent to the process. This action may be just a safe action (disconnect, open, close, etc) or it may depend on the current situation. It could be an emergency control action or, for instance, a previously computed suboptimal action.
2. Data acquisition of essential data, and then

[3] PID stands for Proportional-Integral-Derivative controller. See [4] for a basic overview of control concepts.

3. Compute (and deliver) a safe control action based on current data.
   If fresh data are gathered, the decision can be refined and updated, improving the safe control action or it can be decided to
4. Instigate a change of operation mode. This may require
   - Alarm treatment
   - Change to a new controller (structure and parameters)
   - Deliver a proper (back-up) control action.

Under normal operation, without resources constraints, these activities will be complemented by the following:

- Get the full set of required data and process it.
- Compute the current control action and deliver it.
- Evaluate and select the most appropriate control structure (i.e. which are the variables and the controllers involved in computing the control action).
- Communicate with the environment, other systems and/or the operator.
- Coordinate with higher decision levels.

Other than the control action delivering, the mode changing protocol and the fault treatment, a number of local decisions should be taken with a high priority:

- Detect missing data. Missing data should be replaced by estimated data. If it is not possible, a mode change should be initiated, in order to use a control structure not requiring the missing data.
- Evaluate control performance. This will allow the monitoring of the control task to decide if the control is properly working or some extra action should be taken.
- Determine the control action to be issued. Alternative controllers may compute different control actions. It should be decided which action is the most convenient to be applied and delivered to the process.
- Change the operating mode. As a result of any of the previous decisions.
- Compute back-up signals (CA and outputs). Have some alternatives ready in case there is some incident in the next sampling periods.

## 3.2 Models, Signals and Controllers

Most control algorithms are designed by using model-based approaches. That is, the controller design procedure uses in some way an abstraction, a model of the process to be controlled. Obviously, the more complex the process model is the more complex the controller results. And a complex controller implementation requires more computing resources, availability of a larger number of signals and usually a longer computation time.

In hard RT control applications alternative controllers based on different models should be available so as to use the most appropriate according to the current resource availability.

In the control action computation, some signals are fundamental and some others are complementary. Also, there may be different representations of the same variable depending on its accuracy, estimation procedure or time print. The quality of the signals will also determine their use. Missing data are usually replaced by old data, and are updated by extrapolation or prediction algorithms.

To compute the control action different controllers may be used. Each controller is characterized by its complexity (the number of operations required for its computation), the information required to run it (parameters and involved variables) and its computing time. The system must be able to decide the controller to be used as well as the required pre- and post- computing treatment of the involved signals.

### 3.3 Control Action Relevance: The Control Effort

In the sequential execution of multitasking activities as a result of simultaneously controlling several variables, that is dealing with multiloop control, there are inherent delays and jitter between the sampling of process signals and the delivering of control actions.

In digital control, the control loop is open in between two updates of the control input. Thus both delays and jitter influence the control performance. This influence[4] depends on the changes the controller produces in the controlled plant dynamics. In fact, if there is a soft control the absence of control action is not necessarily significant, but if we pretend to strongly change the plant behavior by the control action, then any delay will modify the result. In general, the stronger the change in dynamics is the worse the effect of any unaccounted delay. There are different ways to determine the relevance of a signal. In a qualitative way, the most relevant signal is that whose failure provokes the highest performance degradation in the system.

In this framework, it is interesting to introduce the concept of *control effort* as a measure of the change in the plant dynamics the control action produces[5]. Assume a multivariable process, with $p$ measurements being stabilized/controlled by means of $m$ control actions. Among these variables $(m+p)$ the most relevant one is the one that leads to a less stable behavior, if disconnected.

Altogether, the signals to be more accurately computed are those involving the greater control effort but also related to the more relevant controlled signals.

### 3.4 Control Algorithm

As a result, the design of the control algorithm must be able to adapt parameters and data of the controller as well as to reconfigure the control structure based on:

– New jitters and delays due to the rescheduling.
– Changes in the sampling period, due to changes in the resources availability.
– Model reduction of the process and/or controller, also due to changes in the resources and specially in the event of missing data.

[4] In some special cases this influence may be positive [1] but, in general, the control performance is degraded.

– Power availability, to change to simplest controllers if a lack of energy is expected in the next future.

Moreover, for many autonomous RT control applications, changes in operation mode and environment should be accounted for in the equations of the control algorithm.

There are several approaches to control algorithm design. First, to try to maximize the determinism of the control algorithm computation, avoiding looping and options. This will result in a perfectly schedulable algorithm. If some flexibility is introduced, then a robust design allowing to cope with different scenarios would be required. The price we pay for that is a conservative and less performant control. Next step is to introduce a dynamic (active) robustness by changing the frame according to the situation (i.e., gain scheduling). The best performing approach, from the control viewpoint, but also the more demanding from the computing viewpoint is to implement feedback scheduling [15], that is, to schedule the different activities based on current measurements taken on the process.

## 4 Challenging Control Scenarios

As already mentioned, new control scenarios have emerged. In particular, dealing with resource constraints (embedded control systems), distributed resources (networked control systems) and non uniform/periodic sampling (event-driven control). Let us discuss the main features of these new scenarios.

### 4.1 Embedded Control Systems

Most of the warnings and concerns discussed above apply to the design of Embedded Control Systems (ECS). RT control applications on embedded systems require the best use of the available computation resources. The main advantages they offer include 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 performance. But the most important problem is the limited computational capabilities they can use. Short sampling periods and non-delayed control actions which warrant better control performance cannot be ensured.

Hence, one of the most important issues is related with the reliable and optimal use of the computational resources and what the resource shortage involves in the design and implementation of the control algorithms. For these applications, it is not always possible to implement the control by using general purpose operating systems because of the particular requirements in terms of delays and jitter limitation. Thus, the control computations should be implemented as RT tasks being executed under a specific RT scheduling policy. In this sense, the basic common features of ECS can be summarized as: compact and reduced size, autonomy, reconfigurability, safety, fault tolerance and ability to work under missing data operation conditions. In the end, one CPU, with its own power supply, must control a number of variables in an uncertain environment. The problems to be considered are related to implementation, workload and resources sharing, and control performance degradation.

From the implementation point of view the same resources, and in particular the CPU, must be shared between different tasks. As a result of this competition for the CPU, time delays and jitter affect the activation times of tasks, which in turn has an effect on the performance of the control algorithm. Working in a changeable environment, the control goals and options may change and the control algorithms should be adequate to new scenarios. Thus, alternative control algorithms should be ready to get the control of the process. The changeable scheduling and data availability determines that variable delays should be considered, [14]. The synchronicity of signals cannot be ensured anymore. Any embedded control system should be proved to be reliable and safe operation should be ensured.

From a computational point of view low-cost algorithms should be designed to use as little computation time and memory as possible, and an easy update of information should be provided to allow for the shortest time in controller changes. Control algorithms should be split in mandatory and optional tasks, the later being only run if time is available. Switching between different control scenarios requires to consider supervisory levels, and resource saving claims for memory saving (storing only what is necessary) and optimal data transfer. On the other hand, for safety reasons hardware redundancy may be implemented if its cost, size and involved complexity is affordable and fault detection, isolation and system reconfigurability should be provided to allow an autonomous behavior.

ECS algorithms should be designed by using the most appropriate process model, with different levels of detail. Thus model reduction techniques should be considered to simplify either the plant model or the designed controller. As already mentioned, supervisory control is required to decide about the most suitable operation mode. This implies the design of hybrid controllers, taking care of the transfer between modes of operation. It is well known that, in general, the performance of the digital control degrades if the sampling period increases. Thus, if resources are available, the control tasks periods should be reduced and this implies changes in the controller parameters as well as in the stored information. One mandatory feature of ECS is a safe operation. Thus, control algorithms should provide always a control action, even under adverse conditions such as data missing, faults or lack of computation time to compute the most convenient control action.

Also, as discussed in the sequel, non regular sampling should be considered.

## 4.2 Networked Control Systems

A new order of complexity arises when the the control system is spread over a network of computing-sensing-acting devices (nodes) linked through different networking technologies.

Each node may house different devices, see figure 5, acting locally or through the network. In this scenario the RT analysis must be extended to take into account the

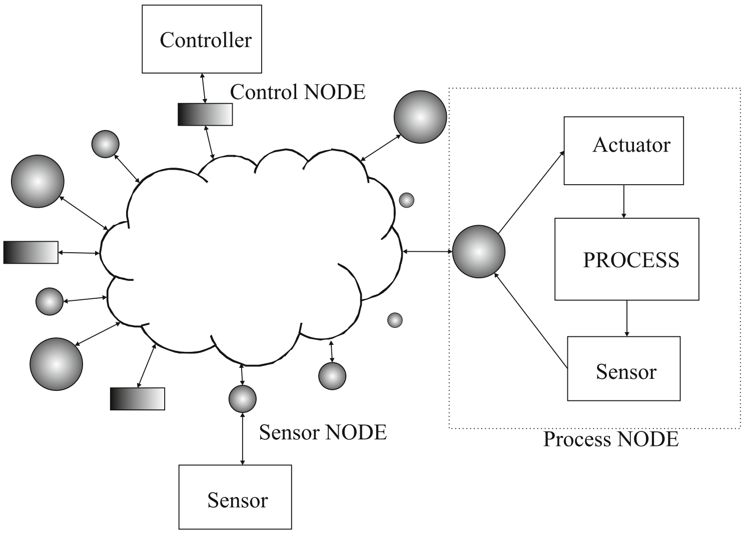

**Fig. 5.** Networked processes, processors, sensors and actuators

delays introduced by the communication infrastructure. Additionally, if some RT communication infrastructure is available, a scheduling plan for the involved buses should be obtained to ensure delay bounds. Given the specification of control tasks and the capabilities of computing and communication infrastructures, the off-line scheduling analysis should follow the next rough steps:

1. Identification of control tasks and mapping between tasks and processing nodes. After this mapping, the worst case execution time (WCET) of each task can be computed as resulting from the target processing architecture.
2. Identification of shared information and bus scheduling. Once the bus access protocols and scheduling have been determined the worst case for communication delays can be obtained.
3. Tasks scheduling for each node. If some task cannot meet its deadline, return to Step 1 and reassign tasks to nodes.

This approach has some weak aspects:

- The final implementation of the entire control system is too rigid. Tasks are assigned to nodes in the way that the movement of tasks among nodes or the inclusion/rejection of tasks from nodes should be foreseen at the design stage and modeled as "operation modes". Each operation mode provides a different scheduling scenario that must be analyzed off-line following the above mentioned three steps. Moreover, tasks belonging to each operation mode should be preloaded in affinity nodes in order to be ready to run if the operation mode changes. This fact usually leads to a memory over-consumption that impacts the hardware requirements. The design complexity can be assumed when the operation modes are well defined and the number of them is not too high taken into account the design resources and perhaps business bounds for design cost. This design complexity becomes intractable when the physical system to be controlled is highly dynamic and reconfigurable. In these cases a hierarchical decomposition of the problem must be made with, again, a consequent increase in design complexity.

Although from the computational point of view, the problems derived from the “operational modes” decomposition can be overcome, the main criticism to this approach comes from the control engineering field. The main problem here is the “mode switching” procedure. Engineers can ensure a stable performance of the controllers running in each mode but it is hard to ensure stability during the switching time, even more if the switching is random. New components should be included in the architecture to manage the mode switching in closed loop.

- As for the memory, the communication and processing specifications should be defined by the peak of load found among all operation modes (and perhaps influenced by the switching strategy). This leads to an increase of the cost of the computing infrastructure. This cannot be avoided but regarding the general low resource usage, systems should integrate into the scheduling components power management techniques as fine grained as possible in order to spend energy only when needed. This is especially important in nodes using wireless technology and battery powered.

- The maximum delay for a data transaction over the communication infrastructure provided by the off-line analysis is enough to feed schedulability analysis and verify that control deadlines are satisfied. But, again from the control engineering point of view, this maximum delay cannot be used to align samples in time and correct the control action by taking into account explicitly the real delay in the controller computations. In this context, communication technologies providing time meta-information together with data are very valuable.

## 4.3 Event-Driven Control

Regular sampled data systems are very popular. First because their mathematical treatment is well established and there are many control analysis and design techniques for them. Also, because digital control has been traditionally implemented in computer-based systems with a time driven operation. These two conditions match perfectly and excellent results are reported in the literature. Problems arise if there is lack of resources. Regular sampling implies the repeating of the control cycle at every period. It does not matter if it is needed or not, and it does not take into account if the sampling frequency is the one the control performance requires. The first change in this scenario is when the processed signals have different power spectrum. In this case they do not need to be sampled at the same rate. Then, multirate systems become a necessity, optimizing the use of the sensors, actuators and computing units. But we can go further. We can compute the control action just when it is needed. This is the basis of the so called *event-driven* control systems.

As summarized in the work of K. Åström [9], regular sampled data are unfeasible when dealing with distributed, asynchronous and multi time scale systems. In these cases, the control should be updated anytime an event is detected,

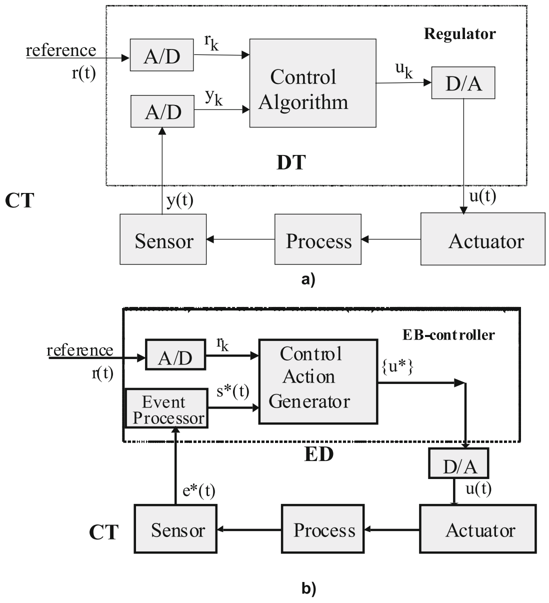

**Fig. 6.** Basic control loop: a) regular sampled data; b) event-driven

analyzed and a new control action is generated. In this conceptual case the control structure is different, as depicted in figure 6.

The sensor can be a local virtual sensor, processing the measurements and detecting an event signal $e^*(t)$ to be sent to the controller. The event processor determines $(s^*(t))$ what should be done and fires the control action generator. This block is a kind of "generalized hold" generating the series of signals $u^*$ (it could be just a permanent one) to be sent to the actuator as far as no new event appears. In this way, the communication load is highly reduced and it only happens upon the occurrence of an event. As discussed in the next section, both the event detection and the storage of the future control actions can be done at a lower (the kernel) level.

There is a lot of work to do in this area and new tools to analyze the stability and performance of event-driven controllers should be developed. What is undoubtable is that this is the right approach to save resources without degrading performance.

## 5 Control Co-design: The Control Kernel

To fulfill the basic and common features of software applications, the kernel of a RT operating system (RTOS) provides an "abstraction layer" that hides from the application the hardware details of the target processor on which it runs. This permits the development of portable applications. Moreover, it ensures the execution of the most important activities.

In a similar way, the control kernel concept [2] can be interpreted as the basic services to be provided in order that the control application can be safely performed. This also requires to define which are the basic and most important control activities, in any control application, as outlined in Section 3.1.

The control kernel will be run with high priority. Its main goals are [2]:

1. To provide a flexible interface to manipulate the gathered information (including the registration and elimination of sensors) from a control application at execution time. In this way, it is possible to reconfigure the application under changing environments.
2. To provide a flexible interface to manipulate actuator devices, including their registration and elimination.
3. To keep control over the system in all situations. Each control subsystem must be able to autonomously control its corresponding part within the system.
4. The control kernel should be robust and adaptable to changing situations. Furthermore, it should detect fault conditions in sensors, actuators, communication networks, etc.

## 5.1 Main Features of the Control Kernel

As the processors operation has a sequential behavior, some basic activities should be implemented at a low level and with high priority to guarantee the availability of the resources they need and a timely response. Furthermore, as the ultimate platform where the controller will be implemented can change in the sense of computational resources, it is necessary to use a layered structure to accomplish the basic requirements of the control system.

The control kernel should be able to interact with the environment (sensors, actuators, communication channels), the OS and exchange information with the control algorithm implemented at application level. Also, it should cluster all common jobs of the control activities related to any variable, such as the data acquisition and the delivering of the control action [19].

Figure 7 shows these interactions, where the following abbreviations stand for:

- ADQ: Samples of variables.
- REF: Control references.
- ESM: Outputs, references and inputs estimation.
- CO: Control commands, including
  - Controller commuting
  - Change of controllers' parameters.
- SCA: Sending of control actions.
- SDI: State and diagnostic of inputs.

Taking into account the elements analyzed in section 3.1 the proposed architecture should have a *supervisor component* at the control kernel level. It will permit to apply each control strategy at the needed time and to carry out the following actions:

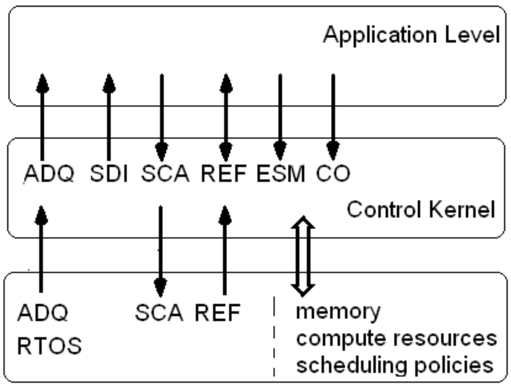

**Fig. 7.** Layers and interactions of the control kernel

1. To check the control variable measurements.
2. To check the control performance index.
3. To switch controllers.
4. To change the parameters of the controllers.
5. To send the estimated control actions.

The control kernel should have access to a set of alternative control actions from where it is possible to select and send the most appropriate one to the process. The algorithms corresponding to the most complex control strategies (i.e. optimal control, adaptive control, predictive control and intelligent control) should be run at the application level because they need a larger processing capacity. Other control strategies such as PID (all variants) or simple state feedback control will be allocated at the control kernel level and they will be identified as basic controllers. It is worth clarifying that the full control algorithm is not a part of the control kernel because it depends on the process to be controlled. This makes it possible to assure that the control kernel behaves deterministically because it is possible to determine with good precision the time required to compute the control action.

## 5.2 A Distributed Control Kernel Implementation

The control kernel should be conceived in a modular way, allowing a distributed implementation as well as different levels of complexity depending on the application. To this end, a distributed embedded control model can be defined as composed by two node types: *Light nodes* and *Service nodes* (see figure 8). 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 processors with limited computing and networking capabilities but complete I/O features.

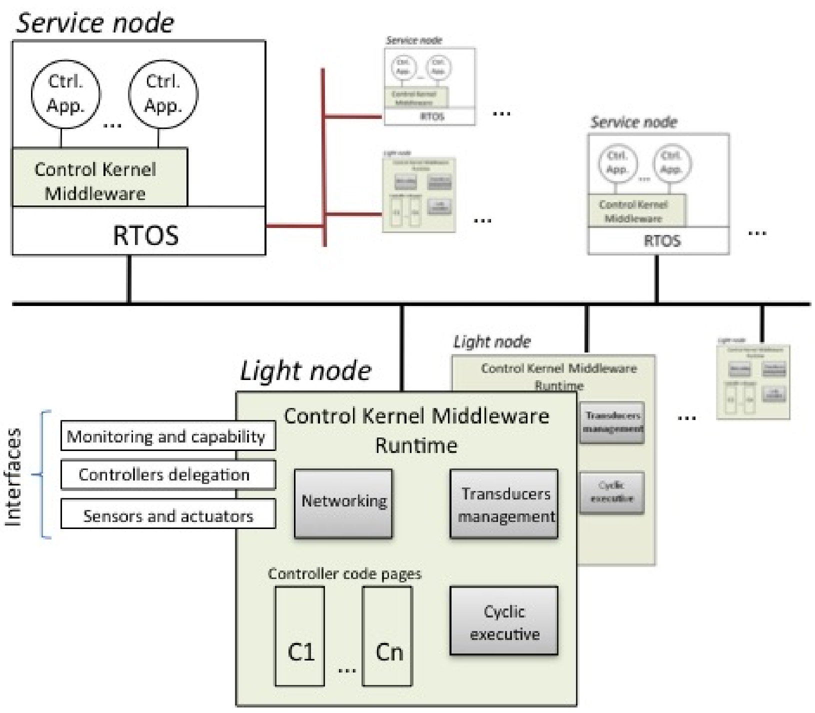

**Fig. 8.** Architecture of the distributed control kernel

Control applications run in service nodes on top of a fully featured *Control Kernel Middleware* (CKM). This middleware provides for abstractions and functionalities related to RT execution of control tasks, access to sensors and actuators, and communications management. The programming model of the 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 has access to the communications space of the task.

Light nodes are a cost-effective solution to have some computing resources as close as possible to each actuator. This is mandatory in order to reduce the non-determinism 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 the CKM Runtime include network interfacing to sensors and actuators and controller code pages upload. A light node can be used as a simple slave component to interface the DCS or it can run local 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

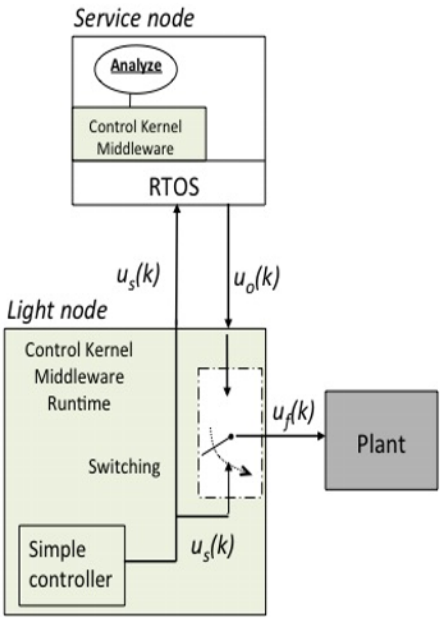

**Fig. 9.** Light nodes ensure that always exist a control action to be sent to the process ($u_o(k)$ or $u_s(k)$)

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 optimizing control activities and light nodes can run activities to drive the system to a safe state or run a simple algorithm that guarantees a minimum of stability in the system at any time.

Let us consider the model depicted in figure 9. Two nodes are defined for this control kernel structure:

- 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. It ensures that there always exists 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.

When $u_o(k)$ is not detected or it is wrong, immediately the signal $u_f(k)$ is switched to a safe signal $u_s(k)$. 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 the light node to execute other controller.

## 6 Discussion and Conclusions

In this work a biased analysis of the co-design of control algorithms and their implementation has been presented, mainly from a control viewpoint.

Initially, the requirements from both the control and computing perspective have been reviewed and some relevant application areas have been visited to emphasize the embedded character of many applications as well as the need for development platforms capable of speeding up the design and validation of integrated and complex RT control applications.

To fully develop a demanding RT control application, other than the end user requirements the constraints imposed by the implementation environment should be taken into account to get the best possible performance and, in any case, a safe operation. This co-design approach is always convenient but it becomes necessary if dealing with new control developments where resources are scarce, operating conditions are very changeable and there is a great uncertainty in the achievable results. In this sense, embedded control systems, with high autonomy and resources constraints, and networked control systems, with distributed and shared resources, are emblematic frameworks. And these settings are pervading most control applications.

The first consequence of this analysis is the re-structuring of the control algorithm implementation. It is not anymore a cyclic loop to be repeated periodically. The relevance of activities and signals should be pointed out. The most fundamental activities should be scheduled at a very low level and priority should be given to the most relevant signals. This leads to the concept of control effort and evokes the need for a basic layer, the control kernel, to execute the fundamental activities.

To save resources, all the activities leading to compute the control action should be only performed if needed, and in many cases there is a waste of resources to compute a control action being the same as the previous one or easily predicted from it. In this framework, event-based control systems offer a way to go. Unfortunately, the control theory required to design event-controlled plants is still in its dawn, and much effort should be devoted to reach the maturity that regular sampled-data-based approaches offer to the designers. Also from the communications viewpoint, event-based behavior is quite natural and new developments of communication systems, like DDS, go in this direction.

Let us conclude by emphasizing the concept of control kernel. This layer, implemented as a middleware in the control application, allows for greater reliability, reusability, transportability and safety in the design and operation of RT control applications.

## References

1. Albertos, P.: Phase-conditionally stable systems. Systems & Control Letters 55, 803–808 (2006)
2. Albertos, P., Crespo, A., Simó, J.: Control Kernel: a Key concept in Embedded Control Systems. In: IFAC Conf. on Mechatronics (2006)
3. Albertos, P., Crespo, A., Vallés, M., Ripoll, I.: Embedded control systems: some issues and solutions. In: 16th IFAC World Congress (2005)
4. Albertos, P., Mareels, I.: Feedback and Control for Everyone. Springer, Heidelberg (2010)
5. Albertos, P., Olivares, M.: Time Delay Limitations in Control Implementation. In: European Control Conference, Karlsrue, Germany (1999)
6. Albertos, P., Vallés, M., Cuenca, A., Valera, A.: Essential control in Embedded Control Systems. In: IFAC Symp. On Cost Oriented Automation, Havana, Cuba (2007)
7. Årzén, K.-E., Cervin, A.: Control and Embedded Computing: Survey of Research Directions. In: Proc. 16th IFAC World Congress, Prague, Czech Republic (2005)
8. Årzén, K.-E., Cervin, A., Eker, J., Sha, L.: An Introduction to Control and Scheduling Co-Design. In: Proceedings of the 39th IEEE Conference on Decision and Control, Sydney, Australia (2000)
9. Åström, K.J.: Event based control In Analysis and Design of Nonlinear Control Systems. In: Honour of Alberto Isidori, vol. 147, pp. 127–147. Springer, Heidelberg (2007)
10. Åström, K.J., Wittenmark, B.: Computer controlled systems. Prentice Hall, Englewood Cliffs (1997)
11. Baliga, G., Kumar, R.: A Middleware for Control Over Networks. In: Decision and Control. European Control Conference, pp. 482–487 (2005)
12. Banning, R., Roesch, M., Morgan, A.: Improved Embedded Flight Control System Design Oricess using Integrated System Design/code Generation Tools, pp. 134–138. IEEE, Los Alamitos (1994)
13. Bertrand, D., Déplanche, A., Faucou, S., Roux, O.H.: A Study of the AADL Mode Change Protocol. In: Proceedings of the 13th IEEE international Conference on on Engineering of Complex Computer Systems. ICECCS, pp. 288–293. IEEE Computer Society, Washington (2008)
14. Cervin, A.: Integrated Control and Real-Time Scheduling. PhD Thesis. Departament of Automatic Control, Lund Institute of Technology, Lund, Sweden (2003)
15. Cervin, A., Ecker, J., Bernhardsson, B., Årzén, K.-E.: Feedback feedforward scheduling of control tasks. Real Time Systems 23(6), 25–53 (2002)
16. Chow, M.-Y., Tipsuwan, Y.: Network-Based Control Systems: a Tutorial. In: The 27th Annual Conference of the IEEE Industrial Electronics Society, pp. 1593–1600 (2001)
17. Crespo, A., Albertos, P., Vallés, M., Lluesma, M., Simó, J.: Schedulability issues in complex embedded control systems. In: IEEE International Symposium on Computer-Aided Control Systems Design, Munich, Germany (2006)
18. Crespo, A., Ripoll, I., Masmano, M., Arberet, P., Metge, J.J.: XtratuM an Open Source Hypervisor for TSP Embedded Systems in Aerospace. Data Systems In Aerospace DASIA, Istanbul, Turkey (2009)
19. Fernández, A., Vallés, M., Crespo, A., Albertos, P., Simó, J.: Middleware for Control Kernel Implementation in Embedded Control Systems. In: The 17th IFAC World Congress, Seoul, Korea (2008)

20. Kopetz, H.: Real-time systems: design principles for distributed embedded applications. Kluwer Academic Publishers, Dordrecht (1997)
21. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC (2008)
22. OMG: Data Distribution Service for Real-Time Systems, v1.1. Document formal/2005-12-04 (2005)
23. Pardo-Castellote, G.: OMG Data-Distribution Service: architectural overview. In: Proceedings of 23rd International Conference on Distributed Computing Systems Workshops, Providence, USA, vol. 19-22, pp. 200–206 (2003)
24. Peiró, S., Masmano, M., Ripoll, I., Crespo, A.: PaRTiKle OS, a Replacement for the Core of RTLinux-GPL. In: 9th Real Time Llinux Workshop, Linz, Austria, Real-Time Systems Group, Polytechnic University of Valencia, p. 6 (2007)
25. PolyORB.: Distributed applications and middleware, `http://polyorb.ow2.org/`
26. Real, J., Crespo, A.: Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. Real-Time Systems 26(2), 161–197 (2004)
27. Schantz, R.E., Loyall, J.P., Rodrigues, C., Schmidt, D.C., Krishnamurthy, Y., Pyarali, I.: Flexible and Adaptive QoS Control for Distributed Real-Time and Embedded Middleware. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, Springer, Heidelberg (2003)
28. Xia, F., Sun, Y.: Control-Scheduling Codesign: A Perspective on Integrating Control and Computing. NO.S. 1, 1352 (2006)