# CONTROL KERNEL: A KEY CONCEPT IN EMBEDDED CONTROL SYSTEMS

**P. Albertos**, **A. Crespo** , **José Simó**

*Instituto de Automática e Informática Industrial*
*Universidad Politï¿½nica de Valencia*
*P.O. Box. 22012, E-46071, Valencia, Spain.*
*e-mail: pedro@aii.upv.es, alfons@aii.upv.es, jsimo@aii.upv.es*

Abstract: Embedded control systems are becoming ubiquitous in control applications. They combine the properties of computer embedded system with newly designed complex controllers where flexible, safe and reconfigurable operations are required. Some common and general features are always required in any control system, independently of the hardware implementing platform. The new concept of the control kernel of an application is introduced, making easier the design of general purpose applications.

Keywords: Embedded control systems, real-time constraints, control kernel, control performances.

## 1. INTRODUCTION

Real-time and embedded control systems (ECS) operate in constrained environments in which computer resources (memory and processing power) are limited. They often need to provide their services within strict time deadlines to the applications. Dealing with real-time control applications, they must provide an adequate reaction to the process and the environment to guarantee a safe and reliable operation.

In mechatronic devices the control is always an embedded part of the system, fully interacting with the rest of components and performing more and more critical activities. ECS constitute an enabling technology for future mechatronic development where the ECS provides the basis for introducing new functionalities in an integrated framework. That also means the inter-device communication and information exchange, leading to a networked distributed control system. But the device reliability strongly relies on that of the control component. This is an aspect of particular relevance in safety critical applications such as aircrafts, vehicles, mobile robots or medical equipment.

Although being autonomous subsystems, they must be off-line accessible and programmable, reusable and modular, to keep as low as possible the total cost of the design. That means, they should perform as many as possible activities regardless the implementation platform used for a specific application.

This leads to the concept of the application kernel. This concept is well known in the computer engineering field. In this environment, the kernel is the part of the operating system that provides the most basic services to the application. The kernel of a real-time operating system (RTOS) provides an "abstraction layer" that hides from application the hardware details of the processor upon which the application software will run. This allows to

[1] This work has been partially granted by the CICYT project number DPI2002-0443

develop the application regardless the hardware to be later on implemented in.

In a similar way, the control kernel concept can be interpreted as the basic services to be provided in order to perform the control application. Obviously, the kernel should provide enough flexibility to get the best performance of the rest of components. This also requires to define which are the basic and prioritized control activities, in any control application.

There are a number of basic assumptions about the control implementation at the control design stage:

- 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 algorithm is fixed and well defined

- Alternative controllers, if there exist, are independent each other

- Power supply is guaranteed

It is difficult to guarantee the fulfilment of these assumptions in ECS.

The main goal of this paper is to emphasize about the relevance of some control activities, their inclusion in the so-called *control kernel*, and the advantages in developing ECS based on this kernel. The paper is organized as follows: in the next section, the main characteristics of ECS are summarized.

## 2. EMBEDDED CONTROL SYSTEMS

The cost reduction in the hardware implementation of digital control systems has two main effects. On the one hand, the field of applications is enlarged as the cost of the control system becomes affordable with respect to the cost of the controlled process. On the other hand, the control device is reduced on size leading to a limitation on the available resources. As a result, the control subsystem is integrated into the whole process. The control is embedded and the main features of this kind of systems should be taken into account. An ECS should be considered as a part of the device and its computational facilities are transparent to the user, if not included in its specifications.

Among the main features of the ECS, the following ones are the most relevant from the control viewpoint:

- Autonomy, to work independently of the workstation.
- Fault-tolerance, to be able to work in a degraded or faulty mode.
- Missing data operation, to cope with the absence of external information.
- Reconfigurability, to be adapted to changing conditions. This requires a set of control options as well as the transfer mechanism among these options.
- Hybrid control, combining continuous time control and decision making systems.
- Variable time delays in the system operation as well as in the computation time.
- Safety, to avoid damages to people in the vicinity.

In industry, most of the ECS are based on microcontrollers and Programmable Logic Controllers, (PLC's). Microcontrollers provide most of the basic features to implement basic control systems (processor, input/output, converters) but normally they provide low computation level and dedicated range of application. They often have no operating system, or a specialized embedded code (often a real-time operating system), or the programmer is assigned to port one of these to the new system. Small kernels with specific services are used for these kind of systems to develop several tasks. Its capacity of adaptation and reconfiguration are very limited and its use is constrained to well very known and simple control systems (printers, modems).

The need of a higher computation power can be improved by the use of DSPs. These microcomputers whose hardware, software, and instruction sets are optimized for high-speed numeric processing applications play an essential role for processing digital data in real time. But again, their facilities to implement control structures are rather limited.

Most of the industry control applications use PLCs which provide a wide range of input/output and communication protocols. However, the programming languages have not evolved as in other software technology and there are many different ladder diagram languages to develop applications. In general, these languages provide poorly programming structures and the kernels do not provide most of the features offered by the real-time operating systems. Some initiatives (PLCOpen), focused around the standard IEC 61131-3, are trying to adopt a norm in the design and operation of the programming interface.

However, the new ECSs are characterized by growing software complexity and functionalities where embedded software dominates the development cost and schedule. The old way of developing software for each embedded project from scratch is giving way to the need to reuse software, and build on existing software wherever possible.

Also the embedded (real-time) operating systems are providing more and more new services to fulfil the new needs. Examples of these embedded operating systems are: embedded Linux (several distributions), Windows CE, VxWorks, QNX, OS-9, etc. The diversity of operating environments and platforms poses a real challenge in deploying software across multiple platforms and configurations. The use of embedded operating systems providing POSIX interface facilitates the portability and reusability of the applications.

Other than the previously cited control requirements, from the digital implementation viewpoint, ECSs must handle

- Multitasking environment in the one-single-computer-based control.
- Hard real time constraints in the control implementation.
- Resources optimisation (CPU, memory, power supply, hardware redundancy)

These features require dealing with some issues in designing and implementing the control, as the same resource must be shared between different tasks. As a result of this competition for the CPU use, the timing of the tasks is not fully determined and the possible variable time delays should be taken into account. Alternative control algorithms should be ready to get the control of the process. Working in a changeable environment, the control goals and options may change and the control algorithms should be adequate to new scenarios. Working conditions, such as priority, allocated time and memory or signals availability may change. Thus, complexity, structure and basic properties of the control system should change. The synchronicity of signals cannot be ensured anymore. From the safety point of view, the control applications should be validated and certified. Any embedded control system should be proved to be reliable and safety operation should be ensured.

Many research groups are working on these issues, see for instance (Cervin 2000, 2003, Baruah 1999, Abeni 2000, Koutsoukos 2000, Goebel 2004, Seto 1996, Albertos 2000).

## 3. CONTROL ACTIVITIES

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. Moreover, to run a control application in a safe mode, the following activities should be executed:

- A safe control action should be delivered at any required time to the process. This signal may be the result of a detailed computation or simply a safe back-up command such as: *keep unchanged*.

- A supervisory control must overlook the control 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*, *Robust*, *Adaptive*, etc

- Some data should be recorded, displayed, stored, updated

- Communication links with other activities should be activated

Not all activities have the same priority. It is evident that if there is no reaction to the process the computed control action becomes irrelevant. In this sense, the following priority of actions can be established: Deliver the control action, Get the current essential data, Compute the control action, Select the control algorithm, Determine the control mode (models and goals), Access to the full external data, Supervise the behavior, Update the internal information and Evaluate the control performance.

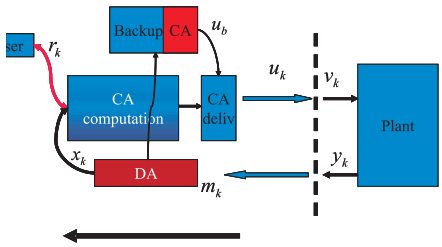


Fig. 1. OS structure

With respect to the Figure 1, the most important activity is

- Ensure control action (CA) delivering
  - Safe back-up
  - Back-up based on previous (stored) data (process situation)

That is, there is a signal sent to the process. This signal 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.

- Data acquisition of essential data, and then
- 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

- Transfering to a new operation mode
  - Alarm treatment
  - Change to a new control mode (structure and parameters)
  - deliver a proper (back-up) control action

Under normal operation, without resources constraints,

- Get the full set of required data
- Compute the current control action and deliver it
- Control structures evaluation and selection
- Communication facilities with the environment, other ECS and the operator
- Coordination facilities

Let us review the concept of kernel and suggest which of these actions should be at the basic level (highest priority).

## 4. THE OS KERNEL

Typically, the RTOS kernel supplies five main categories of basic services to application software, as seen in Figure 2.

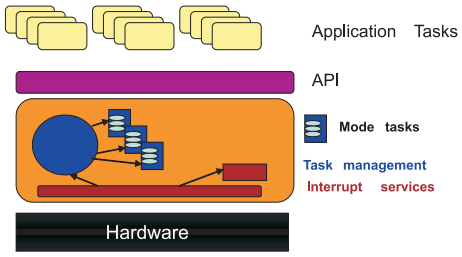


Fig. 2. OS structure

The most basic services are related to the Interrupt and Time management. Among other basic categories of kernel services is the Task Management. This set of services allows the application to create and execute several threads or tasks capturing different concurrent activities of the application (periodic or aperiodic activities). Services in this category include the ability to launch tasks and assign priorities to them. The main RTOS service in this category is the scheduling of tasks as the embedded system is in operation. The Task Scheduler controls the execution of application tasks.

The second category of kernel services is the Intertask Communication and Synchronization. These services make possible the exchange of information between tasks. They also make it possible for tasks to coordinate, so that they can productively cooperate with one another. I/O Management services permit to the applications sending or receiving data from external devices.

In addition to the kernel services, many RTOSs offer a number of optional add-on operating system components for higher-level services such as file system organization, network communication, network management, user-interface graphics, etc. Although many of these add-on components are much larger and much more complex than the RTOS kernel, they rely on the presence of the RTOS kernel and take advantage of its basic services. Each of these add-on components are included in an embedded system only if its services are needed for implementing the embedded application, in order to keep program memory consumption to a minimum.

### 4.1 *Specific control requirements*

Considering the control application, several additional services, not included in the normal ones provided by the kernel, should be taken into account. These services are mainly related to the delivering of the control action back-up, the organization of the application in different modes and the fault tolerant management.

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 preassignment 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 in the next time instants, if they are not updated at the next sampling period. This sequence should be stored and used, if no option to get better suggestions is available.

Task mode management is related to the organization of tasks in modes of functioning. Several tasks included in a mode cooperate in the system control when some external conditions are done. For instance, in a normal operation of a furnace 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 (**?**). Task mode management could be included in the kernel as basic services or at the application level as a control thread.

Fault tolerant management is related to the detection and management of abnormal situations. The error detection is a service that could be considered basic and should be provided by the kernel. Error management is more related to the application which knows in detail how the error or fault has to be handled. Fault tolerance involve both questions, at the kernel level, the detection of faults and, depending on the fault nature, its management or its propagation to the application.

Different type of faults can be considered (**?**):

- Transient faults: they starts at a particular time, remains during some periods and disappears. Examples of such faults occurs in hardware components (communications).
- Permanent faults: they starts at a particular time and remains until they are repaired.
- Intermittent faults: they are transient faults that appear from time to time.

The operating system should provide a basic management for all these failures trying to perform the operation in the same or an alternative device several times. If the fault is not solved, the fault has to be managed at the application level.

From the application domain, failure modes can be identified :

- Time failure: the service is delivered at the wrong time. Time failures can be detected by the operating system (deadline missing, several deadline missing in a period window, excessive output delay in a control task, etc.) and can be managed (emit an event change), tolerated or propagated to the application.
- Value failure: the value associated with a service is in error. The detection of these failures belongs to the kernel when the value causes operation failures in expression evaluation (overflow values, division by zero, etc.) or the application when the values are domain. Constraint or value errors have to be evaluated at the application level which knows the value range or limits. Handlers of these situations have to be defined at application level.

## 5. CONTROL KERNEL

In real time embedded systems, an operating system kernel should be provided to handle the basic requirements of the whole system even working in the emergency conditions (Artist 2003). And it has been discussed that, for control applications, some extra requirements should be included in the OS kernel, to ensure the safety of the control operation.

Similarly, from a pure control viewpoint, a control solution should be implemented to ensure the safe behaviour of the system even in the case of resources shortage. This implies that, other than the designed global control algorithm to fulfill the requirements, a simplified control algorithm based on a reduced order model of the plant, and a reliable operation should be implemented as an alternative. These basic properties should be captured by a kernel representation of the system in order to apply an essential control. Thus, if this is the case, the system will run under this control until more resources (options) become available. Among other emergency conditions, the following two circumstances are analysed:

- The time allocated for control computation is reduced and a safe operation (with degraded performances) is foreseen. Thus, the sampling period may be enlarged and only the slow part of the plant model could be considered.
- Under an emergency condition, a quick reaction of the control is expected. The sampling period is reduced and the slow behaviour of the plant may be considered as unchanged.

In both cases a reduced model of the plant is required and a change in the model parameters should be implemented. A similar reasoning will be applied if dealing with the controller model. There are many model reduction techniques but the real-time transfer between equivalent representations should be investigated.

As previously discussed, changes in the operating conditions require changes in the controller. Under any working condition, the system must keep some basic properties. If this is not the case, the emergency routines should take care of the whole system moving it to a safe, even shutdown, situation. These basic properties should be captured by a kernel representation of the system in order to apply an essential control. This control should be able to ensure the system stability under a shortage of resources. Thus, in this case the system will fall under this essential control, until more resources (options) become available.

Generally speaking, the kernel representation implies a reduced order model of the system and a mechanism to transfer from a normal, extended model to the kernel and vice versa. For that purpose, a simple algorithm to transfer between models, also recovering the involved data, should be provided. The transferring approaches require scheduling support to allow mode changes in the system. A mode change is initiated whenever the systems detects a change in the environment or in the internal state that must drive it from one operating mode to another allowing the use of reduced models and the transfer between models.

Several protocols for mode changes can be found in the literature. A survey on mode change protocols can be found in (Real and Crespo 2004). The main requirements to be achieved by the protocols are: schedulability, periodicity, promptness and consistency.

The kernel concept must consider to cope with essential goals connected to safety in the operation as well as different resources' shortage, such as lost of data or time limitation. Let us discuss the model reduction issue under time constraints.

### 5.1 *Transfer between models*

Assume a standard (Albertos, 2004) state space representation of a discrete time plant such as:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = \begin{bmatrix} A_{11} & A_{11} \\ A_{11} & A_{11} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \quad (1)$$

$$y_k = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k$$

where $x_{1,k}$ represents the "fast" components of the state vector and $x_{2,k}$ the "slow" ones. Under some operating conditions constraints, a partial model is enough to compute the appropriate control action.

Deleting the slow dynamics implies to assume $x_{2,k+1} = x_{2,k}$ in (1). That is:

$$x_{1,k+1} = A_1 x_{1,k} + \bar{B}_1 u_k \quad (2)$$

$$y_k = \bar{C}_1 x_{1,k} + \bar{D}_1 u_k \quad (3)$$

where

$$\bar{A}_1 = A_{11} + A_{12}(I - A_{22})^{-1} A_{21}$$
$$\bar{B}_1 = A_{12}(I - A_{22})^{-1} B2 + B_1$$
$$\bar{C}_1 = C_1 + C_2(I - A_{22})^{-1} A_{21}$$
$$\bar{D}_1 = C_2(I - A_{22})^{-1} B_2$$

In this case, to go back to the full model (1) at time $k$, the state vector should be updated to:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k^+ = \begin{bmatrix} I \\ (I - A_{22})^{-1} A_{21} \end{bmatrix} x_{1,k} + \begin{bmatrix} O \\ (I - A_{22})^{-1} B_2 \end{bmatrix} u_k$$

On the other hand, looking for a safe slow model, the assumption of an "immediate" change in the fast modes is equivalent to assume that $x_{1,k}$ is an input to the reduced model. That is:

$$x_{2,k+1} = A_{22} x_{2,k} + A_{21} x_{1,k} + B_2 u_k \quad (4)$$

$$y_k = C_2 x_{2,k} + C_1 x_{1,k}$$

In this case, to go back to the full model and initialise the full state vector, the state variables should be measured or estimated through the output vector.

### 5.2 *Changes in the sampling period*

The DT model (1) of a continuous time plant corresponds to a nominal sampling period, T. If, for any reason, the control tasks are re-scheduled, their periodicity will be changed. It is well known (Albertos, 2000) that, for control purposes, faster sampling rates allow for better control performances than slower ones. So, according to the CPU use, the sampling rate of the control algorithms (and, in the same way that of the plant models used to compute the controller) should be adjusted.

Changes in the discretization rate of a controller requires changes in both, the controller parameters and the set of data handled by the control algorithm, in order to get bumpless commutation and a reduced degrading of control performances. Some options, as previously mentioned, are described in (Albertos, 2003b).

### 5.3 *Scheduling support*

Both previous approaches require scheduling support to allow mode changes in the system. A mode change is initiated whenever the systems detects a change in the environment or in the internal state that must drive it from one operating mode to another allowing the use of reduced models and the transfer between models. Several protocols for mode changes can be found in the literature. A survey on mode change protocols can be found in (Real, 2004). The main requirements to be achieved by the protocols are: schedulability, periodicity, promptness and consistency.

### 5.4 *Control kernel implementation*

The control kernel can be implemented in two different ways:

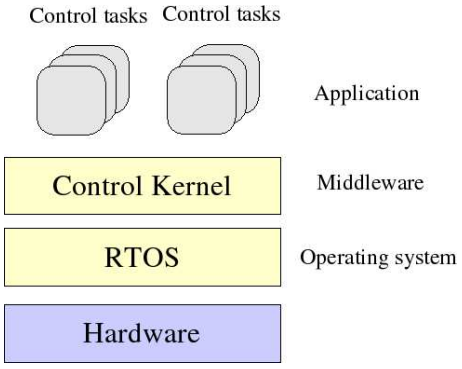


Fig. 3. Control kernel middleware

The control kernel includes two threads that perform the data acquisition and output delivery at the defined time.

- Data acquisition thread (DAT): it is in charge of the data read from external devices. It is a unique periodic thread which works as a input server at different rates. Each control task in the application defines the acquisition rate, the device in which it is interested and the event to notify the task that a new data has been acquired. When a new data for any task is acquired, this DAT raises the event. In order to avoid, the execution of the task with the same values than the previous sample, a threshold can be defined to consider significant changes in the data acquired. If the new data is in threshold band, the information is updated at the DAT level, but no event is generated.

- Output action thread (OAT): It performs the ssending of the calculated variables to the external actuators. As the DAT, it is a unique periodic thread that sends periodically calculated actions. The OAT definition includes the updating rate, a maximum fixed delay to deliver the control action and a safe operation value. If the data is available before the end of the delay, the OAT sends the received value, if not safe or bachup action is delivered.

The specification of the control kernel is presented in the next figure (figure 4

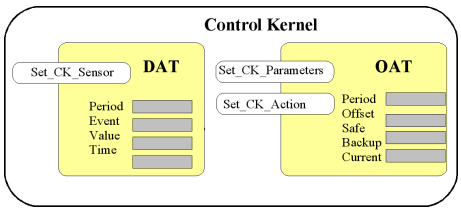


Fig. 4. Control kernel specification

There are two differentiated modules: DAT and OAT. The Dat module defines one function in order to specify the sensing characteristics. On the other hand, the OAT module defines two functions: one for the definition of the actuation and the secod one to provide the action value from the control task.

The relation between this module and the the control tasks is shown in figure 5

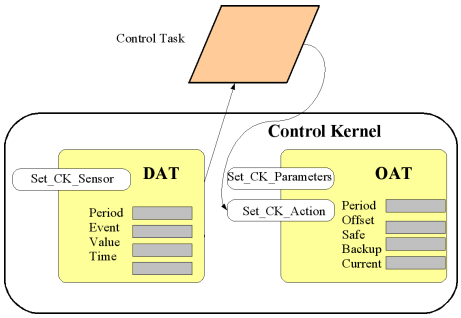


Fig. 5. Control kernel specification

A control task can be implemented in the following way:

```
task Control_task is
    Period, Offset : Time_Span;
    Next : Time;
    New_data : Sensor_Data;
    Action, default_action: Data_output;

begin
                    --define the control kernel behaviour
   Set_CK_Parameters(Period, Offset, safe_action);
                    --define the data acquisition behaviour
   Set_CK_Sensor(Period, Value, Threshold, Event);
   loop
    accept event(New_data) do
      Compute_Control_Action(Cntrller, New_Data, Action);
      Set_CK_Action(action);
    end accept;
   end loop;
end Control_Task;
```

(1) As part of the operating system: Additionally to the services provided by the operating system, new functionalities required by the control kernel should be implemented. In this case, the control kernel will be part of the operating system and the API should include new calls to define the control parameters and to connect them with the application. This implementation has strong drawbacks, at least at this early phase of the control kernel implementation. Moreover, even if it is provided as a component to be selected at the operating system level during the configuration phase of the embedded system support prior the deployment, the inclusion at this level can introduce more complexities than needed.

(2) As a specific middleware for control applications: In this case, the operating system maintains the same functionalities and the control kernel is implemented as a library which is linked with the application. The control kernel functionalities are included in this library allowing to isolate them from the operating system and the applications.

Considering these reasons, we propose to implement the control kernel as a middleware for control applications. Figure 3 shows a scheme of this proposal.

## 6. CONCLUSIONS

Dealing with computer-based control solutions, under the event of resources shortage, a number of basic activities should be performed to ensure a safe operation of the system under control. They conform the *control kernel*. Some of these activities are application independent and some others are specific for the application, but all them should be run with very high priority.

The control kernel is composed of:

- the OS kernel activities
- the back-up, mode change and basic fault treatment
- the simplest alternative controls to achieve basic control performance under extreme working conditions.

As a result, the highest priority tasks, as well those essential to keep the controlled system operating in a safe way must be detected and arranged to be run under any conditions. The control kernel includes all these activities.

The ideas here summarizes are the matter of further research from both the theoretical and practical viewpoints (Albertos *et al.*, 1999), (**?**).

## 7. REFERENCES

Albertos, P., R. Sanchis and A. Sala (1999). Output prediction under scarce data operation control applications.. *Automatica* **35**(8), 1671–1681.