The Self Triggered Task Model for Real-Time Control Systems

Manel Velasco, Pau Martí and Josep M. Fuertes
Dept. of Automatic Control and Computer Engineering
Technical University of Catalonia
Pau Gargallo 5, 08028 Barcelona, Spain
{manel.velasco, pau.marti, josep.m.fuertes}@.upc.es

Abstract- **In this paper we present a control-based model for control tasks that allows each control task to trigger itself optimizing computing resources and control performance. Using this model, at each control task instance execution, the executing instance informs the scheduler when the next instance should be executed. The next instance execution point in time is dynamically obtained as a function of the utilization factor and control performance. Preliminary results show that control activities, at run time, are able to define self-execution patterns that dynamically balance optimal levels of control performance and resource utilization.**

I. INTRODUCTION

In real-time control systems, the objective of control activities (to control processes) and the objective of scheduling policies (to meet deadlines) are accomplished separately. This may derive in sub-optimal designs in terms of both control performance and resource utilization.

On one hand, control activities optimize control performance regardless the computational demands of other tasks. In control design, a discrete-time controller is designed assuming a constant sampling period. In terms of task execution, that means that at run time the controller will execute demanding a constant processing capacity. Therefore, in the design process, it is not usually taken into account the possibility that the controller could take advantage of the processing capacity that may be released by other tasks. That is, the controller design does not allow increasing the execution rate to exploit available resources.

On the other hand, scheduling techniques optimize the use of resources regardless the dynamics of the control application. For instance, a periodic control tasks may not require the designed execution rate (processing capacity) if the controlled plant is in equilibrium. When a plant is in equilibrium, the contribution of each control task instance execution can be considered as useless. Its processing capacity could have been used by other tasks with higher processing capacity demands.

To overcome these problems, we present a control-based model for control tasks in which computing resources and control performance are jointly considered. The model allows each control task to trigger itself: at each control task instance execution, the executing instance informs the scheduler when the next instance should be executed, thus adjusting at run time its timing constraints. The next instance execution point in time is dynamically obtained as a function of the utilization factor (global parameter) and control performance (local parameter)[1]. Consequently, we could say that each control task acts as a co-scheduler, helping the scheduler at the scheduling decisions. Figure 1 illustrates the operation of the whole system.

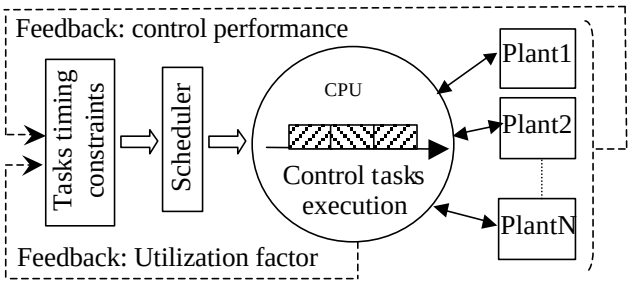

Figure 1. System operation model

Preliminary results show that control activities, at run time, are able to define self-execution patterns that dynamically balance optimal levels of control performance and resource utilization.

The rest of the paper is organized as follows. In section II we discuss the state of the art. The problem formulation is given in section III. In section IV the self-triggered tasks model is developed. Preliminary simulations results are presented in section V. Finally, in section VI we conclude and point to future research work.

II. STATE OF THE ART

The model we present resembles the model presented in [3]. They propose to use feedback information from the controlled plants to take scheduling decisions. Specifically, all control tasks periods are proportionally enlarged or shorted at a given time instant as a function of the utilization factor. Therefore, they do not allow the exchange of processing capacity if the control application requires higher execution rates of specific tasks, as we do.

The later can be achieved using the elastic model of [2]. In such model, the elastic coefficient of each task allows the scheduler change the task execution rate within specified ranges. The elastic coefficients are regarded as fixed parameters to be specified before run-time. The model we propose matches the elastic model if the elastic coefficient of each control task could be treated as a dynamic parameter, being a function of the resource utilization and control performance.

It should be stressed that the work we present derives from [6]. The authors point out that novel methods for control task scheduling in which scheduling decisions should depend on control performance and resource utilization are needed.

Some similarities may be identified between our model and event-based systems. In event-based systems the sampling period takes random values. The sampling period for our model varies, but there exist a slightly difference:

[1] The utilization factor of the system is obtained taken into account all tasks in the system. Therefore, it is a global parameter and affects all tasks. The control performance is obtained by each task from the corresponding controlled plant. Therefore, it is a local parameter and affects each task.

in event-based systems the next execution point in time is unknown; the suggested model in this paper deals with known future periods because they are a result of the model execution.

III. PROBLEM FORMULATION

For control tasks, the task period is given by the sampling period (h_i) that has to balance the desired control performance and the feasible computational demand. The sampling period h_i can be selected form a range of values (see [1] for further reading on the sampling period selection). Each possible choice has advantages and disadvantages. In short, a short period allows a quick reaction in front of perturbations (which is positive from a control point of view), but increases the processor's load (which in negative from a resource utilization point of view). Using long periods decreases the processor's load but may give poor response in front of perturbations.

This demands models that can dynamically accommodate different values for the control task period. The model we present allows control tasks to have varying values for the period. The exact value for the period (at each control task instance execution) is dynamically adjusted depending on the controlled system status and the CPU load.

IV. SELF-TRIGGERED TASK MODEL

The main idea of the model we present is to use the common models that are used in the analysis and design of control systems. Concretely, we propose to use an extended state-space representation. State-space models allow us to describe the future response of a system, given the present state (characterized by the state variables), the excitation inputs and the equations describing its dynamics (see [1] for further reading on discrete-time state space models).

The extension we suggest is to incorporate as new state variables the task period and the utilization factor. Therefore, we will be mixing the control behavior (already represented in the state space model) with the execution rate of the task and the processing demand. In the following subsections, step-by-step, we develop the model.

A. Basic model

Let us think on a closed loop system formed by a ball and beam, which is the plant to be controlled, and a control task that has to be executed on a processor and has to control the ball and beam. The ball and beam system has a motor that balances (by rotating movements) a beam in order to keep the ball (that can rotate freely along the beam) in the desired beam position [1]. The objective of the controller is to actuate on the motor to locate the ball in the desired position. To do so, at each sampling time, the controller takes the value of the position of the ball and the angle of the beam and generates the new angle for the beam that derives in the corresponding actuation on the motor. A linear discrete-time invariant state-space model [1] of the ball and beam is given in (1)

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} \frac{h^2}{2} \\ 2 \cdot h \end{bmatrix} U \qquad (1)$$

In equation (1), x_k and y_k (which are the state variables) represent the position of the ball and the beam angle at the k sampling instant. The first matrix (2x2 dimension), called system matrix, describes the dynamics of the ball and beam. The second matrix (2x1 dimensions), called input coefficient matrix, links the inputs U with the system dynamics. In both matrices, h is the sampling period. U is the available vector of inputs; in our case it is the tension (1x1 dimensions) that we provide to the motor. The input can adopt positive and negative values, allowing the motor of the beam to rotate to both sides.

Note that in the state space representation of the ball and beam, at each sampling instant, x_k and y_k vary according to the system dynamics and the input. However, h, the sampling period, which appears on the matrices as a result of the discretization process, has a constant value that has been chosen at the controller design stage. Recall that (3) is a discrete-time model obtained via discretization of the continuous-time model. Therefore, h has nothing to do with the system state, although it influences its dynamics.

Recall that the state of the system can be directly related to control performance. For instance, a simple rule could be *the smaller the norm of the state vector, the better the controlled system performance.* In terms of the ball and beam: the smaller the deviation of the beam with respect to the horizontal position and the smaller the distance of the ball with respect to the desired location, the better the performance. Therefore, at each task instance execution, the added state variable determines the next task instance execution point in time as a function of control performance.

B. First model modification

As we stressed in section III, we want a model able to accommodate different values for the sampling period (i.e., the task period). The first extension we propose for the previous model allows us to have varying sampling periods according the controlled system dynamics. To do so, we extend the state representation of the system with a new state variable, the task period, h_k, as represented in (2)[2]. Note that for the system in (2) a new control law giving the appropriate sequence of values for the input U is needed.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h_{k+1} & 0 \\ 0 & 1 & 0 \\ a & b & w \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \\ h_k \end{bmatrix} + \begin{bmatrix} \frac{h_{k+1}^2}{2} \\ 2h_{k+1} \\ 0 \end{bmatrix} \cdot U \qquad (2)$$

In (2), at each task instance execution, the task period will be changed according to the state of the system given by x_k, y_k and the new state variable h_k. The dependency of this new variable with the others system variables is given

[2] Note that h_{k+1} (and not h_k) appears inside of the system and input matrices. This is due to the solution of the system equations. In the non-extended model, the sampling period of the system and input matrices has no index (k+1 or k) because it is constant (h at the k instant and h at the k+1 instant have the same value). In the present model, since h_k is a system variable that varies form instance execution to instance execution, it is necessary to distinguish in the matrices which is the appropriate k index.

by parameters *a*, ß and *?* . Let's discuss some properties of the extended model, depending on values of *a*, ß and *?* :

- If *a,ß=0* and *? =1*, then, for each k, $h_{k+1}=h_k$, and the system may be considered as the original given in (1). The control law that will give the sequence of inputs U can be obtained by classical controller design methods.
- If *a,ß=0* and and 0<*?* <1, the sampling period will be decreased at each instance execution, tending to 0, thus leading to a non real system.
- If *a,ß=0* and and *?* ≥1, the sampling period will be increased at each instance execution, tending to ∞, thus violating the Shannon's sampling theorem.
- If *a,ß?0* and zero and 0<*?* <1, we have a system with a variable period, each one depending on the previous system state. In particular, each h_{k+1} value for the next task period depends on the previous one (*?* h_k), with smooth transitions in period variations. The main problem of this combination is that the state space model becomes nonlinear (that is, a small change in the inputs may result in chaotic outputs), as we outline later in this section. Therefore, finding the adequate control law giving the appropriate sequence of inputs U will be a more difficult task (see for example [4]), if feasible.
- If *a,ß?0* and *? =0*, we get a variable period system depending only on the original state variables. Consequently, the more quickly these variables move (angle and position), the faster the period changes, thus loosing the smooth transitions found in the previous case. This may result in values for the sampling periods out of the permissible ranges (as we outlined in section III).
- If *a,ß?0* and *?* ≥1, the evolution of the system will depend on the specific chosen values for *a*, ß and *?* , which require a deeper analysis, out of the scope of this paper.

From the extended model given by (5), three elements should be highlighted. Firstly, the system is nonlinear, since h is a state variable and it also appears multiplying to other state variables. Secondly, it would be possible to obtain negative values for the task period from the actual extended state space model. Considering only a theoretical view, this possibility means that the system should return to the past in order to modify already taken decisions. But this is clearly non-programmable in a real system. We could solve this problem by using the absolute value for the *h* in (2) at each task instance execution. This will guarantee that h will be always positive. Finally, it also have to be stressed that the system matrix in (2) includes an h_{k+1} at the k instant, which is an inconsistency. However, this can be easily solved by substituting the h_{k+1} value for the expression $h_{k+1}=ax_k+ßy_k+?h_k$, which is already known at the k instant.

C. Second model modification

Looking at the final model obtained in the previous section, two difficulties, beyond having a nonlinear model, can be identified. First, the absolute value operator makes the mathematical tractability of the model complex, because it implies using two symmetric models, one for positive values of h and the other for negative values. Second, the possible values that h may take are not bounded, due to the linear relation between h and the original state variables. Note that if the state variables take huge values, h will rapidly increase (and vice versa).

To solve the previous problems, we suggest to bound the possible h values by introducing an appropriate function of the state variables, called *h-function*, instead of having a simple linear relation. In addition, taking advantage of the use of the h-function, we incorporate the utilization factor in the model, as a measure of the processing capacity. Recall that up to now, the model only related the varying period of the task with the original state variables (as a measure of control performance). In this new extension of the model (adding the h-function), expressed in (3), we will relate the period variation to the control performance as well as to the processing capacity.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + y_k \cdot f(x_k, y_k, h_k, z) \\ y_k \\ f(x_k, y_k, h_k, z) \end{bmatrix} + \begin{bmatrix} \frac{f(x_k, y_k, h_k, z)^2}{2} \\ 2f(x_k, y_k, h_k, z) \\ 0 \end{bmatrix} \cdot U \qquad (3)$$

In (3), *z* represents the utilization factor of the processor at the k instant and the h-function is given by *f(.)*. This new model is obtained as a natural extension of the previous one. In the previous section h_{k+1} was obtained as lineal combination of the other state variables. In the new extension h_{k+1} is obtained by an appropriate function of the state of the controlled system and the CPU load. Note that the goal of the h-function is to allow the task period to take values from a bounded range. This allows a grater variety of possibilities in the selection of how h changes at each moment. For instance, the same system with two different h-functions will result in very different behaviors in terms of CPU load and control performance. Note that choosing a specific h-function could facilitate system schedulability (this could be done either offline or even online) as well as improve control performance.

It is important to point out that for the state space model given by (3), the analysis and design of a control law can be a complex task. However, it is possible to design control laws that guarantee the complete stability of the system around a desired working point. These techniques range from the system linearization [4] to the complex techniques of feedback linearization [7].

D. The selection of the h-function

The selection of the h-function determines controlled system performance and CPU load. The most natural way for selecting the h-function is to translate into a mathematical function the following desired rule: as the controlled system gets closer to the desired working point (equilibrium), the period should be as larger as possible, keeping Shannon limit (recall discussion of section III). If a perturbation appears on the system, bringing it away from the equilibrium point, the period should be decreased (to improve control performance) taking into account the available processing capacity. Mathematically, this can be accomplished by the h-function given by (4)

$$h_{k+1} = f(x_k, y_k, h_k, z_k) = e^{-(x_k^2 + y_k^2)} g(z_k) \qquad (4)$$

Note that (4) has a negative exponential shape, with two

main parts. The fist part is the exponential function, which contributes to the h values taking only into account, measures of control performance. It allows each value to smoothly vary form an upper limit to a lower limit, if the exponent of the exponential function is kept positive. That's why we suggest putting the square exponents over x_k and y_k to convert the possible negative values (of the state variables) into positive ones. Note that as we explained in section *A*, in this case, the state variables already are measure of control performance. Otherwise, the exponent should include the operation needed to measure the controlled system performance. The second part, the function $g(z_k)$, allows to correct the next value of h taking into account the processor's utilization factor.

Figure 2 shows possible ranges of h values obtained using the h-function given by (4). The figure has two degrees of freedom: the control performance (horizontal axis) and the utilization factor (which would correspond to a perpendicular axis). The results are given according to the vertical axis, which are the possible values for the task period. When the control performance (understood as the system deviation with respect to the equilibrium point) decreases (the deviation increases), the values for the task period tend to short values, with the aim of quickly correct the deviation. In addition these values are increased as the load increases.

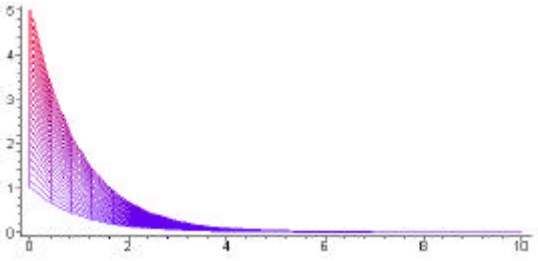

Figure 2. Possible values for a task period

V. SIMULATION RESULTS

Using the extended model (3), at each control task instance execution, the period selection is in consonance with all the elements that are involved in the control of the system and in the scheduling of the task set, thus facilitating the optimization of the whole system in terms of both control performance and resource utilization. The main goal of the presented model is that if several tasks are driven according to this model, the processing capacity can be dynamically balanced among them according to the controlled performance measured, as shown next.

In Figure 3 we show the results of two *ball and beam* control tasks executing on a single processor. The control laws implemented in the two tasks have been calculated by means of linearization techniques. In Figure 3 we can observer 4 lines. The upper ones correspond to the sequences of values for each task period, and the lower ones correspond to the dynamics of each controlled system. The task period values take into account the utilization factor, which is injected as a simulation variable. At the beginning (left side of the figure), both systems are stable at the desired working point, so both have the same value for the execution period.

When a perturbation affects system1, it deviates the system form the desired working point. This perturbation causes an immediate decrease of the task period controlling system 1 and an increase of the task period controlling 2. Therefore, the exchange of the processing capacity among the two control tasks has taken place. Once system 1 is in equilibrium (before the perturbation arrival over system 2), task1 and task2 have again the same period. A similar processing capacity exchange occurs when system2 suffers a perturbation, but in inverse direction.

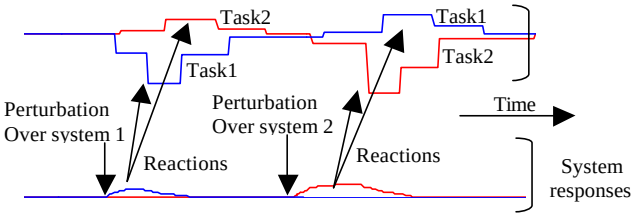

Figure 3. Analysis of responses and periods enlargements.

The delay observed between the perturbation arrival time and the first decrease in the task period is due to two factors. First, there is always an offset among the moment in which the perturbation takes place and the moment in which the task samples the system. In the worst case, this offset could be as big as the task period. Second, after the perturbation arrival time, the error is small and the decrease in the sampling period is not very significant. One period later, the error has increased and the decrease on the sampling period becomes more remarkable.

VI. CONCLUSIONS

In this paper we have presented the self-triggered task model that drives control task executions according to controlled system performance and available processing capacity. Specifically, the model allows control task to adjust their execution rate, acting as a co-scheduler.

As we outlined, the main research issues behind this work is the analysis and design of the controller, which must give the appropriate inputs to drive the whole system with the desired behavior.

REFERENCES

[1] K.J. Åström and B. Wittenmark. *Computer Controlled Systems. Third edition.* Prentice Hall. 1997.

[2] G. Buttatzzo, G. Lipari, M. Caccoamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management". IEEE Trans. on Computers, 51:3, March 2002

[3] A. Cervin, J. Eker, B. Bernhardsson, K.-E. Årzén "Feedback-Feedforward Scheduling of Control Tasks", *Real-Time Systems*, 23:1, 2002.

[4] Isidori, A. *Nonlinear Control Systems*. Spring Verlag, New York, 1989.

[5] Leith, D.J., Shorten, R.N., Leithead, W.E., Mason, O., and Curran P. "Issues in the design of switched linear control system: A benchmark study". Int. Journal of Adaptive Control and Signal Processing. 2003; 17:103-108

[6] P. Marti, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Improving Quality-of-Control using Flexible Timing Constraints: Metric and Scheduling Issues." In Real-Time Systems Symposium, Dec. 2002.

[7] Nijmeier, H. "Nonlinear Dynamical Control systems", Springer-Verlag, 1990.