Feedback Control of Cyber-Physical Systems with Multi Resource Dependencies and Model Uncertainties

Lindberg, Mikael; Årzén, Karl-Erik

Published in:
[Host publication title missing]

DOI:
10.1109/RTSS.2010.14

2010

Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties

Mikael Lindberg
Department of Automatic Control
Lund University
lindberg@control.lth.se

Karl-Erik Årzén
Department of Automatic Control
Lund University
karlerik@control.lth.se

***Abstract*—The problem of modeling and controlling resources in a system with interaction between hardware and software is considered. A model encompassing both hardware and software dynamics is developed together with an online estimation scheme in order reduce dependence on a-priori information. A control structure is presented in order to control performance under constrained resource situations and to reduce effects of estimation errors and disturbances. The approach is applied to a conversational video case and evaluated through simulations.**

I. Introduction

The main characteristic of Cyber-Physical Systems (CPS) is the need to simultaneously consider both the computing and communication processing and the physical system, and their interactions. During recent years a number of challenging applications have emerged within areas such as smart power grids, intelligent homes, environmental monitoring, and transportation systems, see [21] [22] [13] [26]. Many of these applications base their functionality on wireless communication, either in the form of sensor/actuator networks or based on mobile cellular communication systems.

In addition for mobile communications to being an enabling technology for CPS, the components of a mobile communications, i.e., the cellular phones themselves, also can be viewed cyber-physical systems, such as illustrated in Figure 1. For the modern, typically multi-core, processors employed in smart phones efficient thermal management is essential in order to not overheat the system. The chip temperature dynamics is governed by a continuous-time differential equation model which takes as an input the power generated by the CPUs, which in term is decided based on the CPU utilization. An efficient solution for this requires an integrated or holistic approach, i.e., a CPS approach, encompassing both the temperature dynamics and the computations.

Another central characteristic of modern cellular phones is the focus on multimedia applications. Multiple synchronized audio and video streams are required in order to, e.g., support conversational video applications. Multimedia streaming applications are rate-based applications where the execution time each cycle typically changes stochastically depending, e.g., on the type of video frame being processed or due to cache effects.. Correct synchronization between the streams, e.g., between the video and audio stream is essential from a usability perspective.

The uncertainty in execution times is just one out of several sources of uncertainty in these types of systems. The temperature dynamics may vary drastically depending whether the phone is exposed to the sun or not. The limited heat tolerance of the processor imposes restrictions on resource availability in a way that needs to be considered for mobile devices, which working conditions can be hard to predict. The classical approach in real-time systems when subject to dynamic variations in resource consumption is to do a worst-case design. However, in addition to the problem of deriving the worst-case numbers, the resulting over-provisioning of resources is in general not acceptable in these types of mass-market products.

Feedback and estimation are generally very good ways to handle uncertainties in physical systems. This also holds for resource-related uncertainties in embedded real-time computing system. Using feedback the allocation of resources is based on measurement of actual resource consumption and resource availability. This results in a resource management system that dynamically adapts the behavior of the system. Dynamic resource management is often combined with bandwidth server or reservation techniques such as, e.g., constant bandwidth servers (CBS) [2] that enforce the abstraction of virtual processors in which an application is guaranteed a certain share of the total CPU resource.

A challenge for CPS is the development of a unified modeling formalism that covers both physical phenomena and computations. Although, a plethora of different hybrid models have been proposed, they are often overly complex for the purpose of dynamic resource management. What is required is a abstract notion of resources and transformations of resources.

In this paper a feedback-based resource management method is proposed that unifies thermal management and control of media tasks. A unified model is used that is based on the dynamics of flows, both physical and virtual. Unlike previous approaches the feedback is based on metrics that

relate to the application performance, in this case the synchronization error between the video and the audio and the encoding latency, rather than just the CPU utilization. It is shown how the use of such metrics enables the attenuation of transient phenomena, despite being in a resource constrained situation.

A. Related Work

Resource management for embedded systems has its roots in scheduling theory. Many traditional works use Constant Bandwidth Servers [2] or Proportional Share (PS) scheduling [23] to enforce resource reservations. Such mechanisms are today widely available for commodity operating systems, such as Linux [19] [5], but assigning shares still requires knowledge about resource consumption. The approach taken in this paper is that resources are allocated based on run-time metrics. Such schemes have been proposed by e.g. [4], [1], and [17]. Unlike these works however, this paper considers systems where the components have dependencies, both in physical and virtual resources.

[7] studies the coupling between thermal dynamics and software performance, but does this based on the utilization bound, not application performance.

For resource constrained situations, feedforward allocation is often proposed, with Q-RAM [20] being one of the more complete formulations. However, this is mainly an off-line method and relies on a-priori knowledge about execution times. This paper uses a nominal feedforward allocation, but based on on-line estimated dynamics and also applies feedback to reduce effects of estimation errors and disturbances. Thiele proposes the use of Real-Time Calculus for resource management in embedded systems [24], but focuses on analysis instead of control. Other works of interest are [8], [12], [25] and [18].

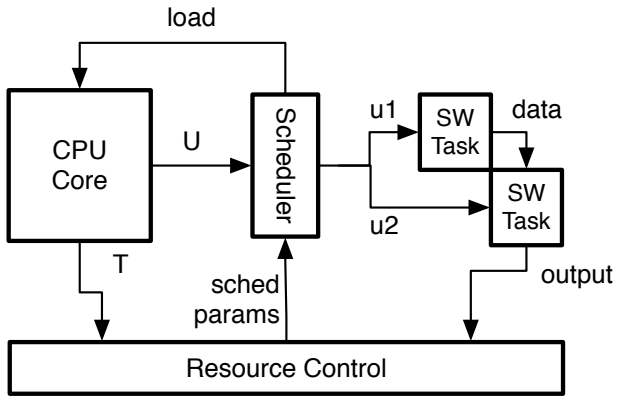

Figure 1. A common CPS, consisting of a CPU and a set of software components. The CPU can be utilized to a degree U that depends on the temperature T. The utilization should then be allocated to the software component so that the output of the system is optimized in some sense.

Recently event based methods have gathered attention in the control society, with some examples in [11], [10], [16], because of the way it can improve on the trade-off between control performance and cost of control action. Event based dynamics is used in this paper, not to reduce the cost of control, but to introduce information that is not possible to describe in flow dynamics. It should be noted that there are similarities between the problem solved in [9] and the synchronization problem discussed below due to the use of event dynamics.

B. Outline of the paper

First the problem of modeling both physical and software dynamics is discussed in Section II, where the cyclic task model is introduced together with a model of the CPU thermal dynamics. This section also introduces the example system, a conversational video pipeline, and the relevant performance metrics. The control structure used is discussed in Section III and the simulation environment and simulation results are then presented in Section IV. A discussion on future directions follows in Section V and then some concluding remarks in Section VI.

II. System Model

One important objective of this paper is to model a system where there is noticeable interaction between the software components and the hardware components. The approach taken is to see it as a system which performance is governed by the flow of resources. *Resource* is here taken to mean a quantity, bounded and non-negative, that through a system component is converted into another resource. The system performance is then expressed in terms of this transformation.

A *resource flow* is the exchange of a resource between two components of the system. In a CPS, a flow can be physical (e.g. heat or power) or virtual (computations or data). In this work physical flows are considered $\mathcal{L}_2$ functions while virtual flows are either $\mathcal{L}_2$ or a sequence of Dirac-spikes, with finite density. This is similar to the notion of discrete event signals described in [14]. Formally, let $\mathcal{U}$ denote the set of generalized functions on $\mathbb{R}$ such that if $u(t) \in \mathcal{U}$ and, $t_1, t_2 \in \mathbb{R}$ then

$$\int_{t_1}^{t_2} u(t)dt \tag{1}$$

exists and has the sign of $t_2 - t_1$. The rationale behind resource flows as Dirac-spikes is that many important virtual resources are generated in an event-like manner. As an example, let $n_{CPU}(t)$ denote the number of completed instructions in a CPU and $n_{task}(t)$ the number of times a sporadic task running on that CPU has executed. Obviously, $n_{task}(t)$ depends on $n_{CPU}(t)$. In order to analyze the performance of the task, e.g. to check if it completes cycle n before a certain time, it is necessary to study how $n_{task}(t)$

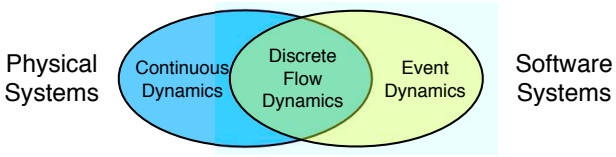

Figure 2. A diagram showing how the domains of physical and software systems have their own unique dynamics, but share the discrete flow dynamics.

and $n_{CPU}(t)$ evolve over time but due to their stair case nature, if t_0 denotes some specific time,

$$\lim_{h\to 0} \frac{n_{task}(t_0) - n_{task}(t_0 - h)}{h} \tag{2}$$

is either 0 or undefined. Therefore the dynamics of a system involving virtual flows must be expressed in discrete time, here called a *discrete flow dynamic*.

Forming such a description for physical systems can be done through sampling. Note however that while the sampled system description says nothing about the system behavior between sample points, signals like $n_{CPU}(t)$ and $n_{task}(t)$ are defined for all t. Because of this, it is possible to talk about exactly when the changes in n occur. These instantaneous changes, referred to as events (see e.g. [10]), contain important information about the system that can be exploited in control.

To summarize, the dynamics of a physical system is often modeled through differential equations. These can be discretized or sampled to give a discrete time model. Software system are usually modeled through event based dynamics, but these can be re-formulated as discrete flow dynamics, comparable with the discretized continuous model. This gives a common model domain with which to express the entire system dynamics. Figure 2 show how the two domains and the dynamic descriptions relate.

A. Cyclic Task model

To derive both discrete flow and event dynamics of a software task, this paper introduces a variant of the periodic task model. In order to accurately express the event dynamics, the exact completion time of each job is important and therefore explicitly part of what will be referred to as the *cyclic task model*.

A task τ_i is cyclic if it repeatedly performs a set of computations that output the same form of results, with the next cycle beginning immediately when the previous ends. The output is a data resource, e.g. encoded video frames, sensor readings or computed control signal. A cyclic task will block if and only if it is starved of CPU-time. The cycle execution time for cycle n is denoted $C_i(n)$ and is stochastic. For estimation purposes, it will be assumed that $C_i(n)$

- has a strictly positive lower bound and

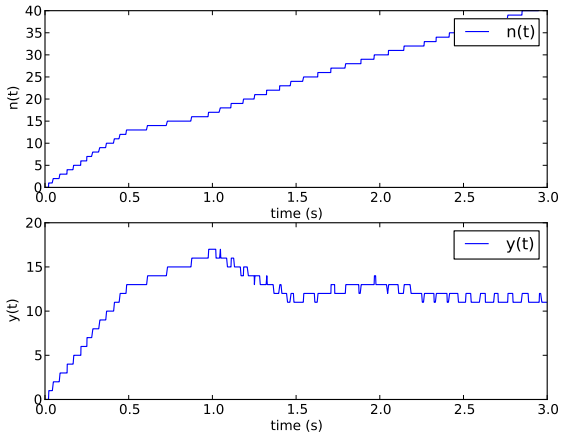

Figure 3. $n_i(t)$ and $y_i(t)$ for one of the tasks in the simulations. Note that these are not sampled curves, despite their jaggedness. Both signals are defined for each t

- constitutes a weakly stationary stochastic process with $E\{C_i\} < h$

From this it follows that if a task is started at t_0 and executes until t_1, the expected number of completed cycles becomes

$$E\{\frac{t_1 - t_0}{C_i}\} \approx \frac{t_1 - t_0}{E\{C_i\}} \tag{3}$$

Let $u_i(t)$ denote the fraction of CPU-time allocated to τ_i through some resource reservation scheme, such as CBS or PS, over the interval $h = t_1 - t_0$. Let $n_i(t)$ denote the number of completed cycles for τ_i at time t. (3) can then be used to approximate the dynamics of $n_i(t)$ as

$$n_i(t+h) - n_i(t) \approx \frac{1}{E\{C_i\}} h u_i(t) = h k_i u_i(t) \tag{4}$$

or in words, n_i evolves approximately like a discrete time integrator with the unknown gain k_i and is driven by the virtual resource flow $u_i(t)$. $\frac{n_i(t) - n_i(t-h)}{h}$ is denoted $y_i(t)$ and is referred to as the *execution rate* of τ_i. $y_i(t)$ constitutes a discrete flow. Figure 3 shows an example of $n_i(t)$ and the corresponding $y_i(t)$ for one of the tasks used in the simulations.

The other aspect of the cyclic task model is the event dynamics, i.e. exactly when the contributions to the flow occur. This has no counterpart in physical flows as contributions are spread out continuously over time. For software tasks on the other hand, the completion time is often an important performance metric. In the cyclic task model, the completion time for cycle n is denoted $x_i(n)$ and modeled as

$$x_i(n+1) = x_i(n) + \frac{C_i(n)}{u_i(x_i(n))} \tag{5}$$

if $u_i(x_i(n))$ is assumed to be constant over the interval $[x_i(n), x_i(n+1))$.

While this could be used for traditional deadline driven scheduling (e.g. keeping $x_i(n+1) < x_i(n) + D_i$)) it is also possible to control other and sometimes more interesting metrics, such as the synchronization between two non-uniform sequences.

B. Multi-resource dependencies

It is desirable to model tasks that require multiple resources to execute. In this paper this is done by assuming that if a task lacks a supply of any one required resource, it will block until it has enough. This is a generalization of the behavior for the dependency on CPU-time introduced previously. It is assumed that a task has the capability of accumulating incoming flows, e.g. in FIFO queues for data or execution time deficit accounting in the scheduler for cpu-time. It follows that the task execution rate is limited by the rate at which resources are made available to it. Formally, if τ_i is dependent on the flows $u_0, ..., u_N$

$$y_i = \min(k_{i,0}u_0, ..., k_{i,N}u_N) \tag{6}$$

would describe its execution rate, where $k_{i,j}$ denotes the dependency from u_j to y_i. From (6) it follows that an allocation strategy should try to keep the incoming flows equal to minimize over-provisioning. Furthermore, it points towards two important objectives for maximizing performance of these systems

1) calculating a steady state flow that maximizes the relevant performance metric

2) control transient effects that cause blocking

C. CPU dynamics

In order to study how the availability of the CPU influences the software and how the software load affects the availability of the CPU, this paper uses a model for the thermal dynamics based on [6]. The CPU must not become too hot and if there is no active cooling, the temperature must be controlled through the power. It is assumed that the system studied in this work consists of a computing platform with one CPU equipped with a thermal sensor. The model of the dynamics from CPU power P to CPU temperature T is

$$\dot{T} = a(T_a - T) + bP + d \tag{7}$$

where a and b are constants depending on the thermal resistance and heat capacity of the processor and T_a the ambient temperature. d is a disturbance term which will be assumed have slow dynamics, such as heat generated by direct sunlight or by being placed on a heated surface. For of-the-shelves CPUs, a and b are in the order of 10^{-4} and 10^{-3} respectively (see e.g. [7]), making the dynamics relatively slow. It is therefore assumed that it possible to filter out measurement noise, which is therefore omitted from the model.

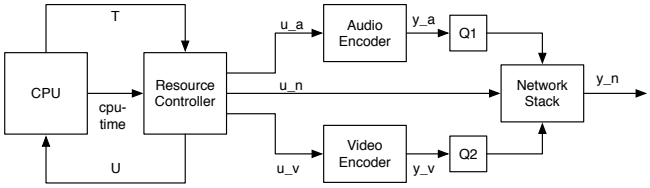

Figure 4. An overview of the conversational video pipeline

The relationship between CPU load U and P is then modeled as

$$P = P_{idle} + U(P_{max} - P_{idle}) \tag{8}$$

Sampling the combination of 7 and 8 can then be done under zero-order-hold assumptions.

D. Example system

The prototypical system to be considered is a conversational video pipeline as displayed in Figure 4. The software part consists of three tasks, an audio encoder, a video decoder and a network stack. The encoders are assumed to have private access to capture hardware and it is also assumed that they are capable of variable rate execution. The encoders are connected to the network through FIFO-queues, Q1 and Q2. In order to send a network packet, the stack requires one frame of audio and video.

In order to evaluate the performance of this system, the following metrics are defined

1) Sync error: It is disturbing to the human eye when video and audio is out of sync and therefore it is natural to consider the difference in encoding timestamp between the corresponding audio and video frames. If both tasks are assumed to be cyclic and $x_a(n)$ and $x_v(n)$ denotes the encoding timestamps for audio and video frames respectively, then

$$\begin{aligned} x_a(n+1) &= x_a(n) + \frac{C_a(n)}{u_a(x_a(n))} \\ x_v(n+1) &= x_v(n) + \frac{C_v(n)}{u_v(x_v(n))} \end{aligned} \tag{9}$$

would be the corresponding dynamics. The sync error $e_s(n)$ is then defined as

$$e_s(n) = x_a(n) - x_v(n) \tag{10}$$

2) Latency: Delay in the conversation is also an important quality metric and for this set up it will be the end to end latency. If $q_1(t)$ and $q_2(t)$ are the number of elements at time t in Q1 and Q2 respectively, then let the $e_l(t)$ be defined as

$$e_l(t) = \frac{\frac{C_a(n)}{u_a(x_a(n))} + \frac{C_v(n)}{u_v(x_v(n))}}{2} + (\frac{q_1(t) + q_2(t)}{2} + 1)\frac{C_n(n)}{u_n(x_n(n))} \tag{11}$$

or in words, the computational delay combined with the network backlog in the queues.

3) Queue dynamics: Using the cycle dynamics expressed in (4), the dynamics of q_1 and q_2 is modeled as

$$\begin{aligned} q_1(t+h) &= q_1(t) + h(y_a - y_n) \\ q_2(t+h) &= q_2(t) + h(y_v - y_n) \end{aligned} \tag{12}$$

III. Control design

The challenges in designing a control scheme for the described system lies in the uncertainty in C_i and the thermal dynamics of the CPU. Normally a CPU can only operate properly if the temperature is kept below a certain level but if there no active cooling, as is the case in many embedded platforms, this must be respected through control of CPU power. The options to do this include voltage- and frequency scaling and idling, the last of which will be used in this paper. The main reason for this is that the speed of the CPU directly affects C_i and any on-line estimates of these parameters will then change due to control action. Controlling the power through U and then imposing the limit

$$u_a + u_v + u_n \leq U \tag{13}$$

simplifies the estimation strategy.

The control will be done through assigning a share of the available CPU-time through a resource reservation framework. In the calculations, fluid resource model is assumed and the actual deviations from the ideal share are modeled as part of the stochastic cycle times. This means that the parameters of the reservation framework will influence the noise levels.

If a CBS framework is used, the server periods should be kept well below h or the assumption that

$$\int_t^{t+h} u_i(t)dt \approx h u_i(t) \tag{14}$$

which is used in (4), will not hold. The server period will also influence the sync dynamics and therefore should be kept below the shortest cycle time in the system in order to minimize disturbances caused by the scheduler.

A. Thermal control

Given that the temperature is modeled with first order dynamics with slow disturbances, a PI-controller [3] is a simple and effective choice, though anti-windup measures must be added to handle effects from control signal saturation.

The pure PI-controller is defined as

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau) \tag{15}$$

By discretization of the dynamics using a forward Euler approximation, constraining the control signal U to the interval $[0, 1]$ and adding an anti-windup tracking term to the integral part, the resulting control algorithm, described as pseudo-code, is

```
e := r - T;
V := K*e + I;
U := sat(V, 0, 1);
if (Ti != 0) then
    I := I + h*e/Ti + h/Tr*(U - V);
endif
```

where h is the sample interval.

B. Estimation

When faced with model uncertainties such as the parameters k_i, there is a choice between either estimating them on-line or applying robust feedback. In constrained situations, knowledge of model parameters is often necessary for trade-off decisions. This paper is therefore based on using an estimation scheme and the resulting estimates are then used for both feedforward and feedback design.

Periodic sampling and recursive estimation methods could be applied on a physical plant (see e.g. [15]), but this would be impractical for use with the event dynamics, which deal with non-discretized time. The approach taken instead is simply counting the number of cycles completed in a time window of length t_w and then estimate $\hat{k}_i$ as

$$\hat{k}_i(t) = \frac{n_i(t) - n_i(t - t_w)}{h u_i} \tag{16}$$

This estimate however relies on u_i being constant over t_w, which might not be the case. Though a reservation based scheduling policy is used, if the estimation window applied is not fitted to the reservation period, the actual amount of resource can deviate noticeably. However, as most operating system expose the accumulated CPU-time per process, i.e.

$$\int_0^t u_i(t)dt \tag{17}$$

through system calls it is therefore possible to reformulate the estimator as

$$\hat{k}_i(t) = \frac{n_i(t) - n_i(t - t_w)}{\int_{t-t_w}^t u_i(t)dt} \tag{18}$$

t_w has been set to h in during simulations but is important to note that the parameter estimates are expressed in continuous time.

C. Latency

It follows from (11) and (6) that latency can be controlled through minimizing the queue lengths and then keeping a uniform steady state cycle time across all components. The approach taken in this work is to combine a feed forward control based on the total amount of resource with a feedback *re-allocation* to reduce queue length. In essence this is a form of midrange control.

The nominal feed forward controls are computed by combining (13) with

$$y_a = y_v = y_n \tag{19}$$

which gives

$$\begin{bmatrix} u_a^{nom} \\ u_v^{nom} \\ u_n^{nom} \end{bmatrix} = \begin{bmatrix} k_a^{-1} \\ k_v^{-1} \\ k_n^{-1} \end{bmatrix} \frac{k_a k_v k_n}{k_a + k_v + k_n} U \tag{20}$$

To control the queue lengths, the feed forward controls are then modified with a feedback term u_q. As the control system cannot violate (13), u_q will be applied as

$$\begin{bmatrix} u_a \\ u_v \\ u_n \end{bmatrix} = \begin{bmatrix} u_a^{nom} \\ u_v^{nom} \\ u_n^{nom} \end{bmatrix} + \begin{bmatrix} \frac{k_v}{k_a+k_v} \\ \frac{k_a}{k_a+k_v} \\ -1 \end{bmatrix} u_q \tag{21}$$

subject to the constraints that the resulting controls $u_i \geq 0$.

To calculate $u_q(t)$, the closed loop dynamics of the queues are evaluated. It is assumed that the queues will be of equal length in steady state (see the section on sync error) so the feedback can be designed with any one in mind. Recall

$$q_1(t+h) = q_1(t) + h(y_a - y_n) \tag{22}$$

Assume the objective is to drive the queue length to some predetermined length r (e.g. zero). Let

$$h(y_a - y_n) = K_q(r - q_1(t)) \tag{23}$$

This converges to r for all $0 < K_q < -2$. Substitute $y_a = k_a(u_a^{nom} + u_q \frac{k_v}{k_a+k_v})$ and $y_n = k_n(u_n^{nom} - u_q)$ and solve for u_q to obtain the actual controls.

D. Sync error

If $\frac{C_a}{u_a}$ is approximated by $(k_a u_a)^{-1}$, it follows from the definition (10) that

$$e_s(n+1) = e_s(n) + (k_a u_a)^{-1} - (k_v u_v)^{-1} \tag{24}$$

As there is a finite combined flow of CPU resource to the audio and video encoder, the approach taken here is to introduce u_s as a feedback term, modifying the feedforward allocation so that

$$e_s(n+1) = e_s(n) + (k_a(u_a+u_s))^{-1} - (k_v(u_v-u_s))^{-1} \tag{25}$$

is driven towards zero. While this seems to interfere with the queue control, the difference in time scale between the event-to-event dynamics make its effects on the slower queue controller negligible. In fact, it is seen in Section IV-E that controlling the sync error actually greatly simplifies the queue control. Let

$$(k_a(u_a + u_s))^{-1} - (k_v(u_v - u_s))^{-1} = K_s e_s(n) \tag{26}$$

Under deterministic circumstances the sequence $e_s(n)$ will converge to zero for any $K_s \in (0, -2)$. Given the variations in the cycle execution times, some care should be taken when selecting K_s as the noise can drive the system unstable. As this work is done without a detailed noise model, K_s is chosen conservatively as -0.5. Then solve for u_s under the constraint that $u_a + u_s \geq 0$ and $u_v - u_s \geq 0$.

The resulting control structure is presented in Figure 5.

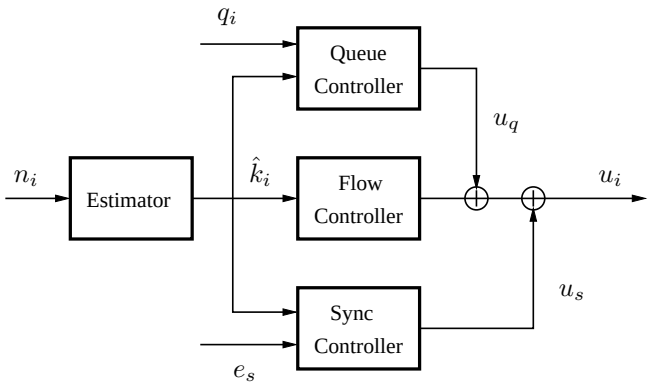

Figure 5. The resulting control structure including estimation, feed forward flow control based on the estimated model parameters and the available CPU resource and feedback based on queue state and sync error

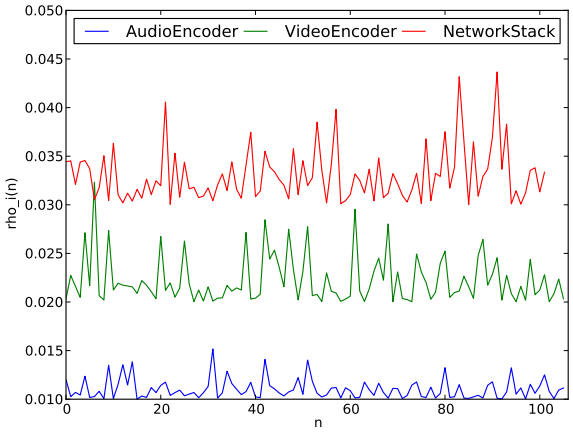

Figure 6. Generated cycle times used in the simulation examples below.

IV. Simulation results

A. Simulation environment

In order to evaluate controls, a simulation environment has been developed in Python. The system dynamics are approximated by discretization with a time step of 1 ms, which incurs quantization on the cycle completion time stamps. This is however assumed to be of little effect as the variation due to noise is orders of magnitude greater for these simulations.

Cycle execution times have been generated as $D_i + X_i(n)$ where D_i is an a-priori unknown constant and $X_i(n) \in \exp(0.1D_i)$. The realizations used for the presented simulation results are shown in Figure 6. The randomness is meant to model both software execution time uncertainty and the stochastic properties of a modern CPU, including the effects of caches, the memory bandwidth gap and deep pipelines. The D_i values were chosen randomly so that the resulting k-parameters would lie between 10 and 100. A real

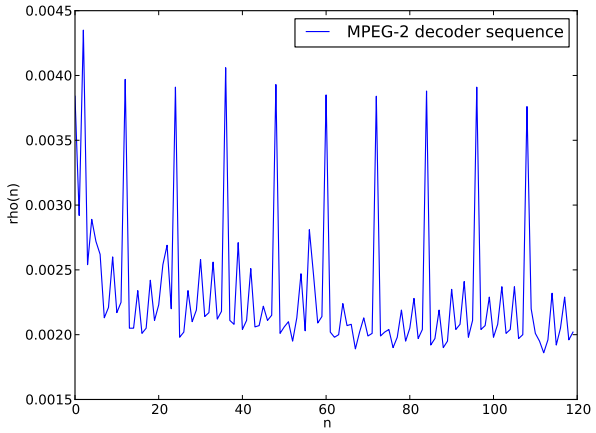

Figure 7. A sequence of execution times for an MPEG-2 decoder working on a video stream. This encoding standard interleaves complete image descriptions with delta descriptions. Decoding a complete image is significantly harder, causing the high peaks.

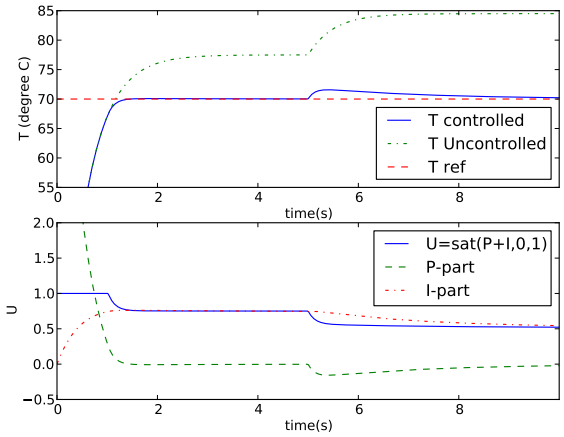

Figure 8. PI control of temperature with a constant disturbance of 15 degrees C / s entering at 5s. The uncontrolled dynamics are shown for comparison.

sequence of cycle times for a video decoder is provided for comparison in Figure 7. h is globally defined as 1.

B. Thermal control

The thermal model of the CPU are based on [7], but to make the effects of the thermal dynamics more visible in the simulations, the parameters have been scaled so the dynamics are faster. This makes the effects of control and disturbances more prominent. In the simulations $a = 2$ and $b = 1.5$. The main purposes of the thermal model are

- to provide a scenario for the varying availability of CPU resource and
- to show how physical models can be combined with the software models.

so switching for a more realistic parameters would not change the design decisions significantly though the thermal controller must then be re-tuned. However, as the focus of this paper is on the application performance as affected by both hardware and software, more advanced temperature control strategies are left for future research.

A scenario where a disturbance enters occurs at 5 seconds is shown in figure 8. This could be a situation where the unit is left in direct sunlight that causes an insolation effect of 15 degrees C / s. The controller keeps the temperature by throttling the available CPU-time.

C. Parameter estimation

Figure 9 shows the estimated execution rates and corresponding k_i-parameter estimates over the simulation. The discontinuous nature of the virtual flows is evident in these plots. Note that it takes some time before the Network Stack starts to execute and this is because of the queue-controller. It throttles the Network Stack while the queues are filled

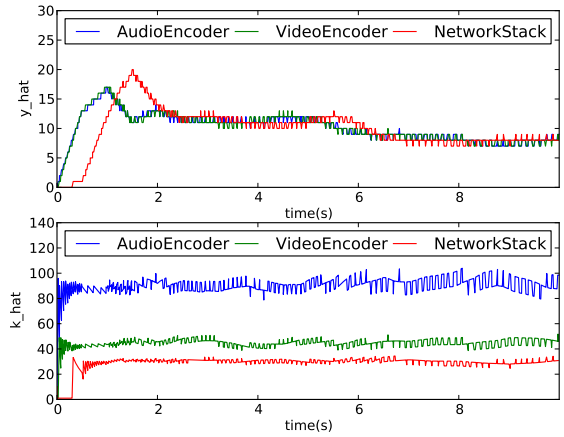

Figure 9. Rate and parameter estimation for all three tasks. Rates are estimated by counting events with a sliding time window with length 1 second. The network stack (red) lags behind in the beginning due to queue control.

and because of this, the k-parameter estimator needs more time to form $\hat{k}_n$. This causes the big overshoot in the queue length before it settles on the desired level.

Even though the actual execution rate changes over time, the estimate mean remains stable while the variance increases at lower rates. This is because a single event being outside or inside the estimation window will affect the estimate more. The window based estimation scheme will break down when the rate drops below 1 cycle per second. Section V-B discusses some alternatives that could be explored in future research.

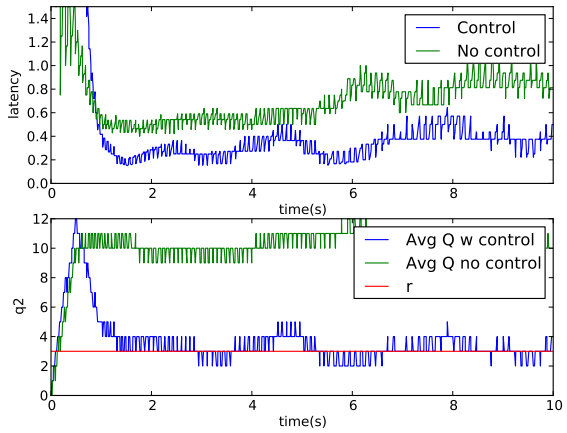

Figure 10. End-to-end latency and average queue length compared with and without control. The uncontrolled case (green) is about 2-3 times worse than what it obtained through control (blue).

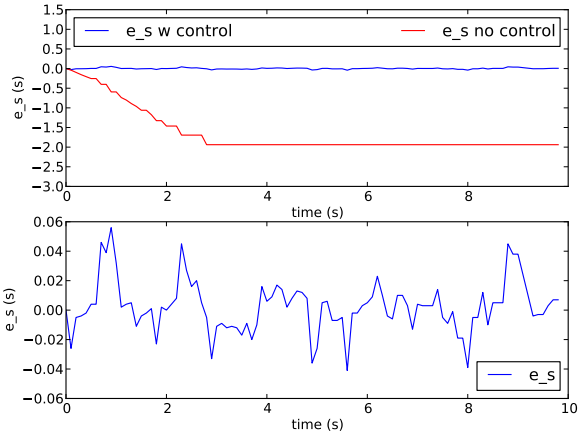

Figure 11. Sync error compared with (blue) and without (red) control. In the uncontrolled case, the encoding pipeline stalls which is why the encoding error seems to remain constant after 2.5 seconds.

D. Latency performance

As there is no information about future demands for the CPU resource, a reasonable strategy is to minimize latency at all times. This is done by utilizing all available CPU-time while respecting the temperature set-point, thereby reducing the execution time for all tasks and by controlling the queues. The reason why the queue controller is not trying to drive the queue lengths to zero is that this could cause blocking in the network stack, which in turn would reduce the accuracy of $\hat{k}_n$. This could be treated by designing a better estimator.

The latency control performance is displayed in Figure 10. A scenario without queue control is provided for comparison and the problem with this is evident. Even though the system reaches steady state, an initial transient is left to cause significantly higher latency.

E. Sync performance

The simulations reveals the importance of the sync controller. Figure 11 shows that the sync error while left uncontrolled will actually drive the system to a stall. The reason for this is that the queue controller uses the average queue length to do the re-allocation. Figure 12 shows that even before the k-parameter estimates converge, the sync controller keeps the audio and video stream tightly together. This means that the queue lengths are actually the same, an assumption which can then be safely used by the queue controller.

Figure 13 shows how the queue lengths diverge quickly and the resulting re-allocation by the queue controller actually starves both audio end video encoder. It would theoretically be possible to form a MIMO-controller to handle both sync and queues in the same control law, but recall that the queue controller is a discrete time system that uses resource

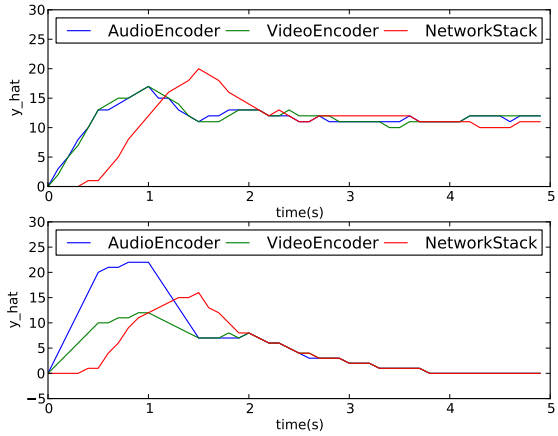

Figure 12. Execution rates compared with and without sync control. The curves have been averaged over a window of 0.1 s to provide better visibility.

flow semantics and therefore cannot in a simple way utilize information about individual events. The sync controller on the other hand operates on the event sequence and thereby have access to the cycle completion time stamps.

V. Discussion and Future Directions

The simulations show that the approach with resource flow semantics, coupled with specific use of cycle completion timestamps can address situations where there is direct interaction between physical and virtual resources. The approach also shows promise as a way to model multi-resource dependencies for dynamic resource allocation. Obviously the limits on performance will come from the quantity of noise in the systems, which limits the accuracy

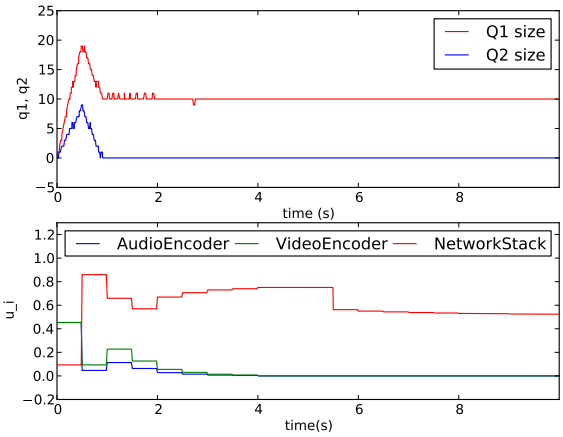

Figure 13. Queue lengths for a scenario with no sync control. The resulting allocation is based on the average queue length $(Q1 + Q2)/2$.

on parameter estimates. The example above shows that even though this system has no slack in resource use at all, it is possible to control performance metrics with relatively simple strategies. Below follow conclusions on some specific topics.

A. Relation between physical and software models

The presented approach lends itself to a way of modeling where there is no significant difference between how we control the physical and software components. Both can be seen as transforming a set of flows into other flows and even though some precision is lost in order to handle the virtual flows, it seems attractive for modeling larger systems. One observation is that the sync problem is very similar to synchronizing servos in robotics or formation control for autonomous vehicles. These are generally resource constrained environments as well, making them attractive future application areas.

B. Parameter estimation

In the simulations, a window based estimation method is used. This is difficult to combine with knowledge about the correlation between execution times, such as displayed in Figure 7. An interesting alternative would be to formulate an estimator working directly on the timestamp series, like the sync controller does. This would require more detailed models of the algorithms and some care when thinking about to express the estimates so that they make sense in the flow semantics.

C. Controller design

The controllers used for this work are all very simple and as there are several constraints in the dynamics, on both state and control, it would be interesting to see how an MPC-style controller could be designed. This is also a direction where load models could be used to reserve resources a head of time, e.g. by reducing CPU temperature in advance so that it could effectively be over-clocked during a time with high demand.

D. Non-constant data resource dependencies

In this work, the data resources are produced and consumed in deterministic quantities. An extension of interest would be where for instance the cycle time is kept constant and the consumption quantities varied.

E. Reconfiguration and startup transients

The start up of the system is handled ad-hoc in this scenario, with $k - parameters$ kept at an initial guess until enough information is gathered. A problem can arise if an initial guess would result in a degenerate allocation causing a system stall. A similar situation could arise if the system is reconfigured at run-time, switching which components are active or redirecting resource flows. It is necessary that formal strategies for preventing such situations from occurring are developed before these types of solutions can be deployed in safety critical environments.

F. Distributed control control

Though the semantics of both physical and software components are similar, a controller that must exists in both worlds can be difficult to build. A distributed formulation where minimal communication between hardware and software could prove necessary.

VI. Conclusions

This paper has presented a way to model dynamics that can be used to express the properties of both software and hardware, which is important to the domain of CPS. It is shown through simulations how such models can be used to do resource allocation in situations with both uncertain dynamics and task to task dependancies. Furthermore the simulations shown that the approach works even in resource constrained environments, as it does not rely on over-provisioning.

VII. Acknowledgements

This research has partially been funded by the VINNOVA/Ericsson project ”Feedback Based Resource Management and Code Generation for Real-time System” and partly by the EU ICT project CHAT (ICT-224428)

References

[1] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, 2003.

[2] L. Abeni, G. Buttazzo, S. Superiore, and S. Anna. Integrating multimedia applications in hard real-time systems. In *In Proceedings of the 19th IEEE Real-time Systems Symposium*, pages 4–13, 1998.

[3] Åström, Karl-Johan and Murray, Richard M. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[4] Cervin, Anton and Eker, Johan and Bernhardsson, Bo and Årzén, Karl-Erik. Feedback–feedforward scheduling of control tasks. *Real-Time Syst.*, 23(1/2):25–53, 2002.

[5] D. Faggioli, M. Trimarchi, and F. Checconi. An implementation of the earliest deadline first algorithm in linux. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1984–1989, New York, NY, USA, 2009. ACM.

[6] R. P. Ferreira and D. Mossé. Thermal faults modeling using a rc model with an application to web farms. In *In Proceedings of RTS*, 2007.

[7] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang. Feedback thermal control for real-time systems. In *Proceedings of the 16th Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*, pages 111–120, Stockholm, Sweden, 2010.

[8] A. Ghosal, T. A. Henzinger, C. M. Kirsch, and M. A. Sanvido. Event-driven programming with logical execution times. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 167–170. Springer Berlin / Heidelberg, 2004.

[9] T. Henningsson and A. Cervin. Comparison of LTI and event-based control for a moving cart with quantized position measurements. In *Proceedings of the European Control Conference*, Budapest, Hungary, Aug. 2009.

[10] T. Henningsson, E. Johannesson, and A. Cervin. Sporadic event-based control of first-order linear stochastic systems. *Automatica*, 44(11):2890 – 2895, 2008.

[11] D. Henriksson, A. Cervin, and K.-E. Årzén. TrueTime: Simulation of control loops under shared computer resources. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, 2002.

[12] T. A. Henzinger, C. M. Kirsch, M. A. Sanvido, and W. Pree. From control models to real-time code using giotto. *IEEE Control Systems Magazine*, 23(1):50–64, 2003.

[13] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. Wiley-Interscience, 2007.

[14] J. Liu. Continuous time and mixed-signal simulation in Ptolemy II. Technical Report UCB/ERL Memorandum M98/74, Dept. of EECS, University of California, Berkeley, CA, USA, 1998.

[15] L. Ljung. *System identification : theory for the user / Lennart Ljung*. Englewood Cliffs, New Jersey : Prentice-Hall, 1987.

[16] J. Lunze and D. Lehmann. A state-feedback approach to event-based control. *Automatica*, 46(1):211 – 215, 2010.

[17] P. Martí, J. M. Fuertes, G. Fohler, and K. Ramamritham. Improving quality-of-control using flexible timing constraints: Metric and scheduling issues. In *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium*, page 91, Washington, DC, USA, 2002. IEEE Computer Society.

[18] M. Mazo, A. Davitian, and P. Tabuada. PESSOA: A tool for embedded controller synthesis. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174, pages 566–569. Springer Berlin / Heidelberg, 2010.

[19] C. S. Pabla. Completely fair scheduler. *Linux J.*, 2009(184):4, 2009.

[20] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium*, page 298, Washington, DC, USA, 1997. IEEE Computer Society.

[21] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang. Cyber-physical systems: A new frontier. In *SUTC '08: Proceedings of the 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pages 1–9, Washington, DC, USA, 2008. IEEE Computer Society.

[22] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 160–171, New York, NY, USA, 2002. ACM.

[23] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium*, page 288, Washington, DC, USA, 1996. IEEE Computer Society.

[24] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, 2006.

[25] G. Weiss and R. Alur. Automata based interfaces for control and scheduling. In A. Bemporad, A. Bicchi, and G. Buttazzo, editors, *Hybrid Systems: Computation and Control*, volume 4416. Springer Berlin / Heidelberg, 2007.

[26] D. R. Yoerger, M. Jakuba, A. M. Bradley, and B. Bingham. Techniques for deep sea near bottom survey using an autonomous underwater vehicle. *Int. J. Rob. Res.*, 26(1):41–54, 2007.