Guaranteed Bounds for the Control Performance Evaluation in Distributed System Architectures

Tobias Bund, Steffen Moser, Steffen Kollmann and Frank Slomka
Institute of Embedded Systems/Real-Time Systems
Faculty of Engineering and Computer Science
Ulm University
89069 Ulm, Germany
Email: {tobias.bund | steffen.moser | steffen.kollmann | frank.slomka}@uni-ulm.de

***Abstract*—Controlling physical systems is a common task of embedded systems. The requirements for a control system are correctness, stability and control performance. This leads to real-time constraints which have to be hold by the implemented controller. We present an approach that supports the control engineer to get guaranteed bounds for the timing behavior of the controller at early design steps. This is done by connecting the Matlab/Simulink environment widely used for controller design to real-time analysis using a mapping between graphs.**

I. Introduction

As the requirements of embedded systems are increasing, we can see more and more complex architectures consisting of a large number of inter-networked electronic control units (ECUs). Taking the automotive industry as an example, we can find up to seventy ECUs in a modern upper-class vehicle. Each ECU consists of various subcomponents like general and application-specific processing units, communication networks and memory architectures. A control unit can be a host for multiple tasks which brings the challenge of finding a suitable task schedule. All these degrees of freedom result in a large design space whereof the system engineer has to choose a solution which must fulfill all requirements that have been specified for the system.

Model driven development is one paradigm that supports the system engineer to handle this task, as it allows to simulate and to analyze the behavior of a system at an early design step.

One commonly required task in embedded systems is controlling a physical system. This results in real-time constraints, which, if violated, can not only have a negative impact on the control performance but may also let the system fail completely. In embedded systems it is not uncommon that the tasks of a controller like gathering sensors values, calculating the actuating signal and driving the actuator are distributed on more than one ECU. This is, for example, the case if one sensor value is needed for more than one technical process. It follows that it is necessary for the system designer to choose solutions of the design space by deciding on which ECU the tasks are mapped and what priority they get on the communication infrastructure. Figure 1 shows a general design flow. The system architecture is designed based on the specification. After then, it is verified regarding different design goals, and changes to the design are done accordingly to the results of the verification. These steps are repeated until the system fulfills its requirements. It is a common way that different verification aspects are explored independently. Consider, for example, the control performance verification. Normally, the controller is verified without any time delays, but this is necessary since the time delay has a direct impact on the control performance as we will see in section VI. Therefore a connection between the control performance and real-time verification is required.

In this paper we give an approach that enhances the model driven design of embedded systems to allow the developer to know in how far a chosen solution in the design space influences the control performance. Therefore the missing connections, depicted as the dashed line in the verification block in figure 1, will be established. To accomplish this, we present a mapping formalism of the compute and delay blocks in controller theory to tasks and communication busses used in real-time analysis. Thereby we show how the commonly used Matlab/Simulink can be connected to a real-time analysis which can be used to analyze the real-time behavior of a complex distributed system with up-to-date algorithms. SimEvents is used to get one model of both, the continuous world of the control plant and the the discrete event simulation in real-time systems.

The paper is structured as follows: In the next section, we

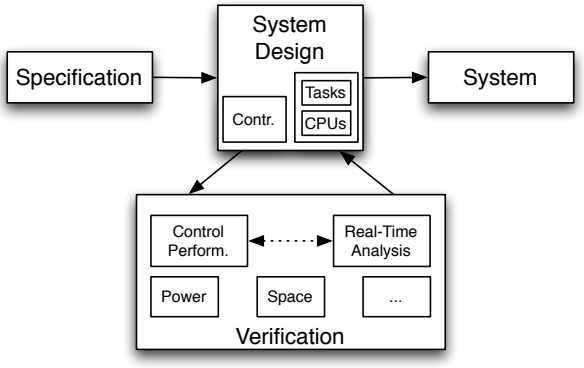

Fig. 1. Exemplary design flow

state the problem and discuss related work. In the third section we show how continuous time-driven and event-driven components can be modeled in Matlab/Simulink using SimEvents. This is followed by the fourth section that describes the real-time analysis. After then we show how we connect controller model and real-time analysis. This is followed by an example.

II. Problem formulation and related work

In classical control theory a dynamic system is described by differential equations. To implement a controller on a computer system, the controller algorithm is performed periodically at discrete time steps. Delays in a control loop have always a destabilizing effect on the system dynamics. To minimize this effect, systems engineers make a strong effort in reducing transport lags. Jitterbug, a Matlab toolbox for real-time control performance analysis [1] simulates the effects of jitter and delays in a control loop and calculates the corresponding cost criteria, which gives a mathematical representation of the control performance.

In many embedded environments, especially in automotive applications, it is necessary to run different tasks on one resource and to transmit information over a bus system. This leads to a delay, caused by the execution times and the scheduling strategy on the resource. Therefore, the control engineer needs to know the amount of delay in the control loop and where it occurs to guarantee a desired control performance. There are approaches to solve this problem by simultions. For example, TrueTime [2], which is implemented as a Matlab/Simulink toolbox, simulates the timely behavior of real-time kernels, executing controller tasks and network protocols to study the influence on networked control loops. In [3] Korte and Slomka present an approach which is based on examining C code by an external real-time simulator after it has been created by a Matlab/Simulink code generating toolbox. As both proposals rely on simulating the real-time behavior, they show the typical coverage problem of simulations, i.e. it cannot be ensured that the worst-case behavior has been observed during the simulation. However, we need to know these bounds to get a guaranteed performance for the control loop. Real-time analysis calculates an upper and a lower bound for the response time of tasks, the worst-case response time and the best-case response time. Those values, back-annotated in the control simulation, can be used to determine an assured control performance. If the performance is not sufficient, improvements in the structure of the controller or adjustments of control parameters can be handled in an early design step.

Aida [4] is a toolset that translates a Simulink model into a data flow diagram which is then used to do the response-time analysis. Compared to Aida, our approach describes a direct formal relation between the controller components and the task level view which makes the flow diagram representation obsolete in our case. To integrate both, the continuous-variable time-driven dynamic system (the plant to be controlled) and the discrete events (for transport and computation) into one model, the Aida toolset appends a layer on top of the simulation. This layer manages the execution time delays and triggers the standard blocks.

In [5], Cassandras and Lafortune describe possible methods to achieve this in general. Digital controllers having a discrete state space can be described as a discrete event system. Systems that combine time-driven with event-driven dynamics are referred to as hybrid systems. One solution to model hybrid systems is the Matlab/Simulink toolbox SimEvents.

In [6], Xu and Wang show how SimEvents can be used for simulating distributed control systems. While they focus on the influence of different routing algorithms on control performance, we will use SimEvents to establish the missing connection between the control simulation and real-time analysis. The usage of SimEvents for this purpose is presented in the next chapter.

III. Controller Model

Technical systems, like an embedded controller operating on discrete time steps are not suitable to be simulated in a continuous time environment. The usage of triggered subsystems is one possibility to solve this problem. The main challenge of modeling control loops is to combine the continuous time base with the discrete time base. Matlab/Simulink is the common standard in the industry to model controllers, that applies a numerical method to solve the set of differential equations represented by a model. The toolbox SimEvents [7] adds a library to the Simulink environment, providing blocks for discrete event simulation. This toolbox is normally used to simulate the behavior of networks, but, as we will show, it can also be used to model a discrete distributed embedded controller.

In SimEvents, entities represent a discrete item that can be transmitted over a network consisting of queues, servers, gates and switches. The information which has to be transported is added to an entity as an attribute of it. For a simple usage and for a better connection to the real-time analysis, we define our control loop on basis of graphs. First we define a distributed embedded controller.

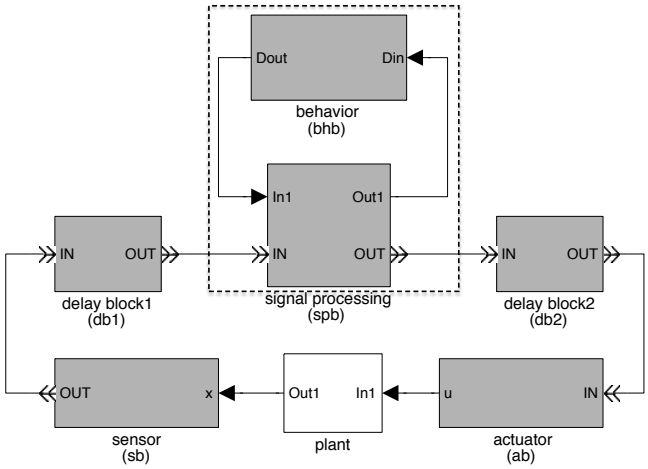

Fig. 2. Content of control loop

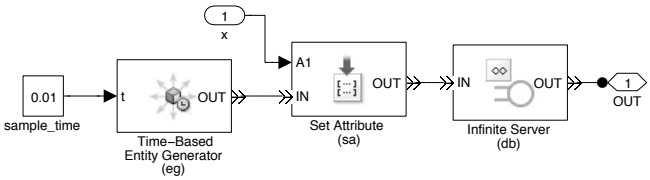

Fig. 3. Content of sensor subsystem (sb)

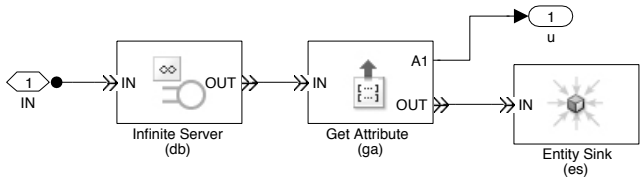

Fig. 5. Content of actuator subsystem (as)

Definition 1. *A distributed embedded control system is a tuple* $dec := (V, E)$ *consisting of* $V := \{sb_1, .., sb_n, ab_1, .., ab_n, spb_1, .., spb_n, db_1, .., db_n\}$ *and a relation* E *where* $E \subseteq V \times V$.

The element sb is the sensor block with a continuous input signal and an event stream of entities as output, the actuator block ab extracts the information from an entity, the signal processing block delays entities and modifies the information of the entities' attributes, delay blocks db specify the time delay of the messages between the elements in sb, ab and spb.

Figure 2 shows a control loop, composed from the basic blocks (grey), control algorithms in the logic block and the plant. Entities are transmitted over signal connections with a doubled arrowhead. Time-continuous signals are transmitted over connections with a single arrowhead. In the next step, we have to define the building blocks of the distributed embedded controller in more detail.

A. Sensor Block

First we define the sensor block which is the entry point for the control loop.

Definition 2. *A sensor block is a tuple* $sb := (V, E)$ *consisting of* $V := (eg, sa, db)$ *and a relation* E *where* $E \subseteq V \times V$.

Consider figure 3 where an example of a sensor block is depicted. Basically, the sensor block has the same functionality as a sample-and-hold circuit.

The main component of the sensor block is the entity generator eg, which generates an entity on every sample time.

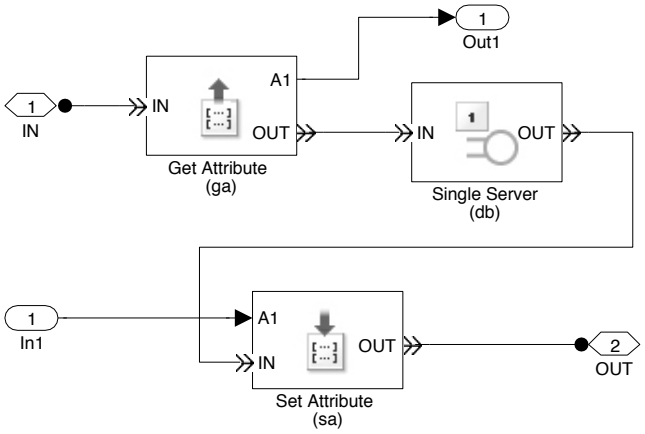

Fig. 4. Content of signal processing subsystem (spb)

The signal x, which is a physical value, is put as an attribute on the entity by the set attribute block sa. To model the delay of the sensor block, e.g. caused by the time needed to convert an analog value in a digital value, a delay block db is used. In SimEvents a server with infinite capacity can be used to model the conversion time.

B. Actuator Block

The actuator block is the counterpart to the sensor block. An example for the actuator block is depicted in figure 5. While the sensor block generates an entity stream and adds information on it, the actuator block extracts the information from the entities by the get attribute block ga. Finally, the entity stream is terminated in an entity sink block es. The time delay of the actuator is modeled by a delay block db. Based on this we can define the actuator block as follows:

Definition 3. *An actuator block is a tuple* $ab := (V, E)$ *consisting of* $V := (db, ga, es)$ *and a relation* E *where* $E \subseteq V \times V$.

C. Signal Processing Block

The signal processing block models the computation unit of the control system. In this basic block, the signal processing in general, for example, filtering and the control algorithm as in our example is realized. The basic block consists of a predefined part which models the time delay and an application-specific block bhb. This behavior block has to be filled by the system engineer. We define the signal processing block as follows:

Definition 4. *A signal processing block is a two tuple* $spb := (V, E)$ *consisting of* $V := (ga, db, sa, bhb)$ *and a relation* E *where* $E \subseteq V \times V$.

In the predefined part, the information is separated from the entities with the get attribute block ga and forwarded to the behavior block bhb, represented by a discrete event subsystem block in SimEvents. While a new actuating signal is computed, the entity is hold in the delay block db which is a single server block in SimEvents. The service time of this block simulates the effect of the delay time of the information flow in the system and thus can hold only one entity. After the service time is elapsed, the new attribute is set and the entity is released by a set attribute block sa. Consider the example depicted in figure 4.

D. Delay Block

A delay block db is an atomic block and represents the time delay caused by the system architecture where the controller is mapped to. In the view of a control engineer, the different effects on transport component are reduced to a guaranteed delay interval. Hence, the delay blocks contain only an infinite server that holds a random number of entities for the service time, or single server components holding only one entity. For the mapping to the real-time analysis we define that $V_{db} := \{db \in V_{dec} \cup V_{sb} \cup V_{ab} \cup V_{spb} | db \text{ is a delay block}\}$ is the set of all delay blocks of the control loop.

IV. Real-Time Analysis

As discussed in section II we need guaranteed bounds for the timing behavior of control systems to evaluate the performance of a control loop. A possibility to solve this is to use schedulability analysis approaches because these deliver for each task in a system a guaranteed bound for the best-case and worst-case response time. Many approaches have been developed in this domain. The base for the real-time analysis of distributed system was introduced by Tindell and Clark in [8]. Since then many improvements have been conducted to obtain more realistic response times of the tasks. Recently, Kollmann et al. proposed in [9] a holistic real-time analysis with an expressive event model. Based on this we will give here the main ideas of the real-time analysis. We start with the definition of a real-time task.

Definition 5. *A task τ from a task set Γ mapped on a resource κ is a tuple $\tau := (c^+, c^-, \rho, b, \Theta^+, \dot{\Theta}^+)$. Thereby, c^+ defines the worst-case execution time, c^- the best-case execution time, ρ the priority for the scheduling, b the blocking time, Θ^+ the maximum incoming stimulation triggering the task and $\dot{\Theta}^+$ the maximum outgoing stimulation generated by the task.*

The worst-case execution time describes the maximum execution demand of a task's job for a resource κ. The best-case execution time is the corresponding counterpart. Due to space limitation we are not able to consider all possible schedulers and therefore we consider here only schedulers with fixed-priorities. Hence, each task must have an unique priority for the scheduler to determine the execution order of the tasks $\tau_i \in \Gamma$ mapped on the same resource κ. The blocking time b_τ determines the maximum delay for a task τ produced by tasks having a lower priority. The incoming stimulation describes the maximum number of events or jobs for a task τ which can be demanded in a specific time interval. The outgoing stimulation describes the maximum number of events which can be generated by a task τ in a specific time interval.

From this we can define two functions necessary for the real-time analysis of fixed-priority systems.

Definition 6. *The event function $\eta(\Delta t, \Theta^+)$ denotes for an incoming stimulation Θ^+ the maximum number of events which can occur in an interval Δt.*

The pseudo-inverse function is defined as follows:

Definition 7. *The interval function $\delta(n, \Theta^+)$ denotes for an incoming stimulation Θ^+ the minimum interval in which n events can occur.*

A detailed definition of these functions in conjunction with a concrete event model is given in [9]. From these definitions we can formulate how the worst-case response time of a task τ can be determined scheduled non-preemptively by fixed-priorities.

Lemma 1. *The worst-case response time of a task is bounded by:*

$$
\begin{aligned}
r^+(\tau) &= \max_{k \in [1,\ldots,m]} \{r^+(k,\tau) - \delta(k, \Theta^+_\tau)\} \\
m &= \max_{k \in \mathbb{N}} \{k | \delta(k, \Theta^+_\tau) \leq \\
&\quad \min\{\Delta t | \Delta t = b_\tau + \sum_{\tau_i \in \Gamma_{hp} \cup \tau} \eta(\Delta t, \tau_i) \cdot c^+_{\tau_i}\}\} \\
r^+(k,\tau) &= \min\{\Delta t | \Delta t = b_\tau + (k-1) \cdot c^+_\tau \\
&\quad + \sum_{\tau_i \in \Gamma_{hp}} \eta(\Delta t, \tau_i) \cdot c^+_{\tau_i}\} + c^+_\tau
\end{aligned}
$$

Proof: The proof is given in [10]. ∎

The idea of this analysis is the following. The equation for $r^+(k,\tau)$ determines for the $k-th$ job the maximum completion time. The maximum completion time is the interval from the request of the first event $\delta(1, \Theta^+_\tau)$ up to the point in time where the $k-th$ job has finished its execution. For this the maximum blocking time of the task b_τ and the maximum execution demands of $(k-1)$-jobs of the task $(k-1)c^+_\tau$ are added. Then the maximum interference of all higher priority tasks $\tau_i \in \Gamma_{hp}$ are added. This equation can be solved by a fixed-point iteration starting with the value of $b_\tau + (k-1)c^+_\tau$. After that, c^+_τ is added for the execution time of the $k-th$ job as we assume a non-preemptive system. To compute the concrete response time of a job the request time must be subtracted from the completion time $r^+(k,\tau) - \delta(k, \Theta^+_\tau)$. The maximum over all jobs in $[1, \ldots, m]$ is the worst-case response time. For a detailed description of the response-time analysis see [10] and [11]. For the best-case response time $r^-(\tau)$, c^-_τ is always a bound. How the whole analysis for distributed systems works and how the outgoing stimulations of a task can be computed is described in [8] and [9], for example. For the real-time analysis the tasks must be connected. This can be solved by defining a graph.

Definition 8. *A real-time graph is a two tuple $rtg := (V, E)$ consisting of $V := \{\tau_1, \ldots, \tau_n\}$ and a relation E where $E \subseteq V \times V$.*

V. Interconnection Of Controller Model and Real-Time Analysis

From the previous sections we know that the delay of the controller blocks and the messages between the controller blocks are modeled by delay blocks. The aim of the real-time analysis is to determine the best-case and worst-case delay of these delay blocks. This is equal to the best-case and worst-case response time of a task in a real-time system. Therefore

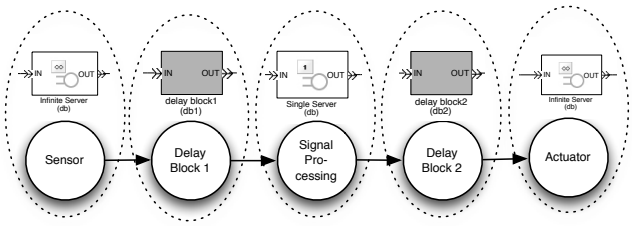

Fig. 6. The controller as graph representation for the real-time analysis

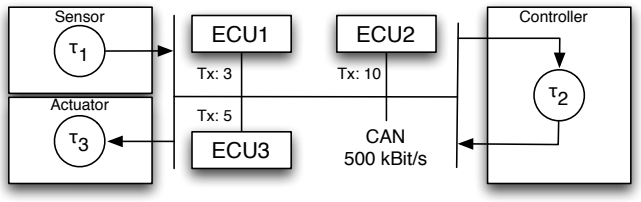

Fig. 7. Hardware architecture

the delay block can be mapped to a task in a real-time system. For this task the best-case and worst-case response time is determined and therefore the timing of the delay block. Since both approaches are defined on graphs, we need only a bijective mapping between the graphs to interconnect the two approaches. Therefore we define:

Definition 9. *Let $db \in V_{db}$ and $\tau \in V_{rtg}$. Then $f : V_{db} \to V_{rtg}$ is a bijective mapping, so that $f(db)$ returns of real-time task and $f^{-1}(\tau)$ returns the corresponding delay block.*

In order to clarify our procedure, the controller loop in figure 2 is mapped to a real-time graph as depicted in figure 6. With the methods of the graph theory it is quite easy to connect the two approaches without any transformation. To the best of our knowledge this is not considered by previous work.

Via this mapping it is possible to exchange information between the control model and the real-time analysis. For example, the initial event streams can be directly derived from the Simulink model, because SimEvents works also on discrete events. The worst-case and best-case execution times depend on the hardware where the tasks are mapped to. To get these times it is possible to use the code generation of Simulink to get the source code which must be executed on the hardware. Via a worst-case execution time analysis as proposed in [12] it is possible to get bounds for the worst-case and best-case execution times.

VI. Example

As a sample application, the system in figure 2 is regarded. The motivation is to control a oscillatory plant over the Controller Area Network (CAN) with an additional load. The system that needs to be controlled is a PT_2 plant with transfer function

$$F(s) = \frac{K}{T_2^2 s^2 + T_1 s + 1} \tag{1}$$

and parameters $T_2 = 0.1$, $T_1 = 0.05$ and gain $K = 10$. The sample rate in the sensor block is chosen to $T_s = 0.01s$ and the controller algorithm in the logic block is constructed as an empirical determined discrete PID controller.

Figure 7 shows the hardware architecture. Additional to the sensor, actuator and the controller, three ECUs are connected to the CAN bus. For simplicity, the sensor, actuator and controller tasks have a fixed best-case and worst-case response time of $50\,\mu s$. The CAN bus has a speed of $500\,kBit/s$ and each message has a size of 8 bytes. Sensor and controller block transmit one message whereas the number of messages transmitted by ECU1, ECU2 and ECU3 are described in figure 7 by Tx. Twenty messages in total are sent over the bus. All messages from ECU1 to ECU3 are sent strictly periodically. The smallest period is $20\,ms$ and the greatest $100\,ms$.

In this example we consider the performance of the control loop by shifting the priorities of the sensor message and the controller message. For this we explore the following cases: Assume, the CAN bus has twenty unique priorities. The sensor message starts with the highest priority and in each step the priority is decreased to the next lower priority up to the second lowest priority. The controller message has always one priority lower than the sensor message. The remaining priorities are distributed to the remaining messages. In each step the best- and worst-case response times are calculated and the influence on the control loop performance is computed. Table I gives an overview of the properties of the CAN messages. The best-case and worst-case response times for the sensor message and the controller message are also described.

The knowledge about the worst-case response times feeds the delay blocks with the corresponding information. Since not only the bounds, defined by $r^+(\tau)$ and $r^-(\tau)$ have a destabilizing effect on the control performance, a varying response time between these bounds can lead to at least the same performance losses. In this example, the jitter is modeled as an equally distributed service time for the delay blocks.

The step response for a step from zero to one regarding the command variable at $t = 1\,s$ is drawn in figure 8, which shows the transient response for different delays. The differences between the step response with best-case response time and no delay are negligible, whereas in a simulation with worst-case response time a distinct oscillating behavior is shown. In the case of a jitter between the best-case and the worst-

CAN priority	size	$r^-(\tau)$	$r^+(\tau)$
1	8 byte	$240\mu s$	480 μs
2	8 byte	$240\mu s$	720 μs
⋮	⋮	⋮	⋮
18	8 byte	$240\mu s$	4560 μs
19	8 byte	$240\mu s$	4800 μs
20	8 byte	$240\mu s$	4800 μs

TABLE I
Properties of the CAN messages

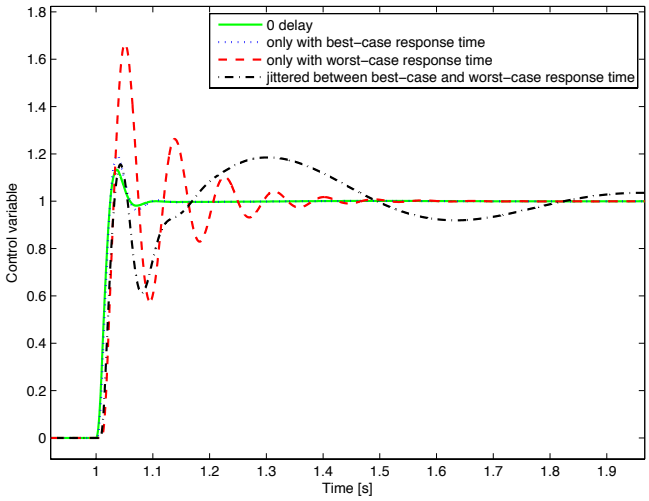

Fig. 8. Step response of the control loop

case response time, the step response for the system has less overshoot, but leads to a longer oscillation.

For further analysis, a quadratic cost criteria is defined, where a high cost results in a poor control performance. A common cost criteria is the integral of the squared error, defined as:

$$J = \int_0^\infty x_d^2(t)\, dt \tag{2}$$

with $x_d(t)$ as control difference. Figure 9 plots the defined cost criteria over CAN priority for the previously described step response of the controlled system. As depicted in Figure 9, the curves for the worst-case response time and in the case of a jitter show a disproportionate rise, whereas the slight curve at priority 18 is caused by the equivalent transport delay for priority 19 and 20 (see Table 1). Note that the choice of the jitter distribution function affects the resulting cost.

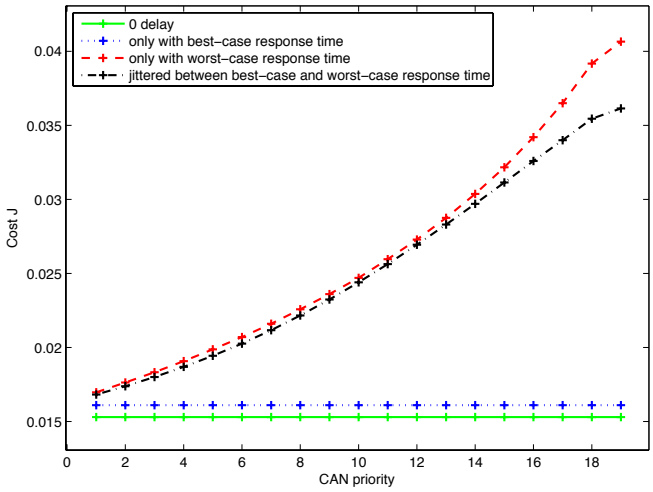

Fig. 9. Cost criteria by decreasing CAN priority of the sensor message

At this point, the controller can be redesigned to reach a better performance for the new knowledge of the delay in the system. This leads to an iterative process in improving the overall performance of the system, as described above.

VII. Conclusion and Outlook

In this paper we presented an approach to connect a control model to real-time analysis by giving a formal mapping between the model components of a controller and tasks of a distributed real-time system. This helps the control engineer to simulate the control performance based on guaranteed timing bounds that are derived mathematically. In the example we have shown in how far control performance is dependant on the system's real-time behavior. Here, not only the the worst- and best-case response times are of interest, but also the amount of jitter of the reponse time.

In this paper we have shown how guaranteed bounds for the timing behavior can be derived and how these can be included in the control performance evaluation. Thereby an open question is how the worst-case performance of the controller can be constructed by these bounds.

The introduced method gives us the opportunity to consider also recently developed control strategies like the event-triggered control.

References

[1] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NV, Dec. 2002.

[2] A. Cervin and K.-E. Årzén, "TrueTime: Simulation tool for performance analysis of real-time embedded systems," in *Model-Based Design for Embedded Systems*, G. Nicolescu and P. J. Mosterman, Eds. CRC Press, Nov. 2009.

[3] M. Korte and F. Slomka, "C-based system development of asynchronous distributed systems. proceedings of the forum on design languages," in *Proceedings of the Forum on Design Languages, ECSI*, Barcelona, Spain, Sep. 2007.

[4] O. Redell, J. El-khoury, and M. Törngren, "The aida tool-set for design and implementation analysis of distributed real-time control systems," in *J. of Microprocessors and Microsystems*. Elsevier, 2004, vol. Vol 28/4 pp 163-182.

[5] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems Second Edition*. Springer, 2008.

[6] X. Xu and Z. Wang, "Networked modeling and simulation based on simevents," in *Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing, ICSC*, Beijing, China, Oct. 2008.

[7] Mathworks. (2010, Aug.) Simevents 3.1. [Online]. Available: http://www.mathworks.com/products/simevents/

[8] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, pp. 117–134, 1994.

[9] S. Kollmann, V. Pollex, and F. Slomka, "Holisitc real-time analysis with an expressive event model," in *proceedings of the 13th Workshop of Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2010.

[10] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," INRIA, Tech. Rep., 1996.

[11] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the 11th IEEE Real-Time Systems Symposium*, December 1990, pp. 201–209.

[12] C. Ferdinand, "Worst case execution time prediction by static program analysis," in *18th International Parallel and Distributed Processing Symposium, IPDPS*, Santa Fe, New Mexico, Apr. 2004.