A new delay specification for cyber-physical systems development

Tobias Bund, Frank Slomka
Institute of Embedded Systems and Real-time systems
Ulm University
{tobias.bund | frank.slomka}@uni-ulm.de

ABSTRACT

One application of cyber-physicals systems are embedded control systems. A physical process is stabilized by a digital controller implemented on computation units. The control unit gathers sensor data over a communication network and sends the calculated control value in order to stabilize the physical system. Such distributed control loops are highly sensitive to delays that occur in the control loop. There are different methods and techniques to obtain these delays, including analysis methods or simulations. So far, these delays are reannotated in the functional model, where the control system is validated and if necessary revised.

With the method described in this paper, we make this step invalid. The delay density is an expressive specification for the allowed amount of delay in an interval and defined in the functional design. If the the specification of the delay density can be validated during the platform design, one can be sure to fulfill the required performance. This paper defines the delay density and how it can derived from a functional behavior. Afterwards, real-time analysis methods are used to validate the specification.

1. INTRODUCTION

When developing a digital control system, a control engineer designs the algorithms and also specifies the rate a new control value is calculated. This rate is often known as sampling rate or sampling frequency. In general this sampling rate is strict periodic, which means the gathering of the sensor values and the following calculating is done in equidistant time intervals. As a further requirement, the control engineer defines a deadline, when the calculation of the new control value has to be finished. This deadline is typically chosen for the same duration as the sampling rate.

This is an established method to develop a digital control system, because the control engineer does not need to consider any timing effects introduces by the implemented software and the underlying platform. On the other hand, the software engineer, who implements the control functionally, does not need to know anything about control theory, as he treats the algorithms as tasks. A paradigm, which implements the above described method is the Time-Triggered Architecture (TTA) [?]. The TTA design paradigm separates the design from the implementation.

When developing resource constraint systems, there is an effort to reduce the computation resource. From smaller computation units follow reduced cost, smaller energy consumption and less wasted heat. To obtain a efficient resource usage, multiple tasks share one computation unit. Additional, in the context of distributed control systems, there is a limited amount of communication bandwidth. Therefore multiple message share the same communication network. As a consequence, the execution time of tasks and messages influence each other, resulting in response times greater than their execution times.

A further possibility to decrease the system utilization is to reduce the sample rate. Unfortunately, a small sample rate makes the control system more sensitive to response times [?]. This fact makes it rather difficult to co-design control functionality and control platform for resource constrained distributed systems.

How to connect the platform introduced timings from platform model and a functional model was introduced in [?]. In the work is described how to map functional blocks and their execution order to a platform model. In the platform model, tasks are assigned a scheduler and a priority. Based on a real-time analysis on the platform model, worst-case response times are calculated. These response times are back-annotated to the appropriate place in the functional model. Afterwards, the control system can be validated and if necessary redesigned, taking the response times into account. This procedure is repeated until a sufficient setting of control algorithm, task priorities and platform is found. This design flow is sketched in figure 1. It is clear, that such a design flow produces a highly integrated design of functionality and platform but could be very time-consuming. A further negative aspect of this design flow is, that in an industrial environment several departments with different expertises are involved. That makes it even more time-consuming.

There are other works, which in general practice the same design flow, as the control performance is elaborated, regarding platform introduced timing effects. In [?] [?] and

[?] tools and methods have been developed, that show the effect of different sampling rate and delay on the control performance. There are also stability criteria, taking into account the influence of time-varying delays, also called jitter [?]. In [?] a more detailed design flow was proposed. Average response times are included, to design the control loop performance for states in which it is most commonly. Additionally the worst-case performance for jitter is derived.

As mentioned above, such a design flow is costly and time-consuming. In order to break the iterative design flow, a new expressive delay specification from the functional model is derived. Such a specification describes the timing bounds a platform designer has to fulfill, otherwise the system will not work properly. One simple approach to specify the allowed delay in a control system is done in [?]. In this work, a sample is defined as stable, when it's end-to-end delay (from sensor to actor) is under a certain limit. Stable samples stabilize the control system. On the other hand, a sample is defined as unstable, when the end-to-end delay exceeds the limit. The requirement for a valid platform design is to keep the ratio between unstable samples and stable samples under a certain threshold.

In our opinion, this specification is imperfect. The described specification would allow an arbitrary amount of consecutive unstable samples, if followed by enough stable samples. The missing reference to the number of consecutive unstable samples could cause an unacceptable performance. Further more time delays should be represented as continuous values. The discretization of delays in multiple samples is a pessimistic simplification. Both restrictions are to be improved through our work.

In the following section, we define a new expressive delay specification for digital control systems. The idea behind the specification is to define the maximum allowed delay for an amount of consecutive samples. The result is an abstract, but powerful description, which we call delay density. In the third section, we show how to derive such a delay density from an arbitrary cyber-physical system. Section four presents real-time analysis methods and describes how to validate the specified delay density. In section five an example is shows the performance of our defined delay density. The paper finishes with a conclusion and an outlook on further work.

2. DELAY DENSITY

In this section we define a new and powerful specification for delays in distributed embedded systems, the delay density. The main idea behind our approach is to specify an amount of maximum and minimum allowed delay for a number of consecutive samples. For systems with periodic task stimulation this corresponds with the allowed delay in a specific time interval. This makes it an abstract and universal specification for delays and defines an interface between the design of functionality and platform design.

From here on we refer to a message as a task. Every task has a response time ($r_i(k)$), defined as the time interval from task activation until it finishes its job. Due to interference between tasks, that share a resource, response times vary in a certain range between consecutive tasks k. As we will

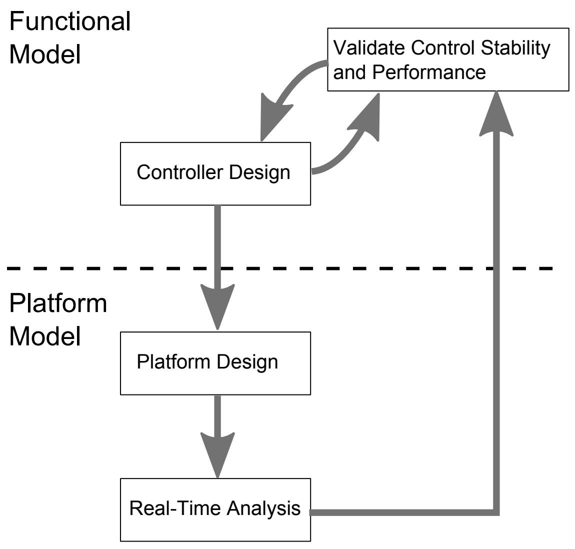

Figure 1: Design flow, which contains elements of the functional and the platform model.

see in section 4, one can obtain a maximum and a minimum response time (r^+, r^-), that bound the response time in an interval. When the response time of all tasks involved in a task chain are summarized, one obtain the end-to-end delay $\delta(k) = \sum r_i(k)$. Figure 2 displays a sequence of events and their corresponding end-to-end delays.

The delay density can be seen as an interface between the functional behavior of a system and the platform design. Depending on which perspective you look at the delay density, it has a different meaning. From the view of a control engineer, the maximum delay density defines the *minimum* delay in an interval, that causes a barely acceptable level of performance. From the perspective of a software engineer or a platform designer, the maximum delay density represents the *maximum* acceptable delay in a specific interval.

In the following a formal definition of the delay density is given from the perspective of the platform design. The maximum delay density ($\zeta^+(\Delta)$) is defined as the maximum sum of Δ successive delays that can occur at any time

$$\zeta^+(\Delta) = \max_{k\in\mathbb{N}} \left\{ \sum_{i=k}^{k-1+\Delta} \delta(i) \right\} \forall \Delta \in \mathbb{N} \tag{1}$$

In a similar manner the minimum delay density ($\zeta^-(\Delta)$) is defined as the minimum sum of successive delays that may occur at any time

$$\zeta^-(\Delta) = \min_{k\in\mathbb{N}} \left\{ \sum_{i=k}^{k-1+\Delta} \delta(i) \right\} \forall \Delta \in \mathbb{N} \tag{2}$$

The delay density is a powerful representation of the allowed delay in the appropriate time interval. Therefore the loose sequence of end-to-end delays is reorder in a way, that maximum and minimum cumulative delay for a number of consecutive samples is displayed.

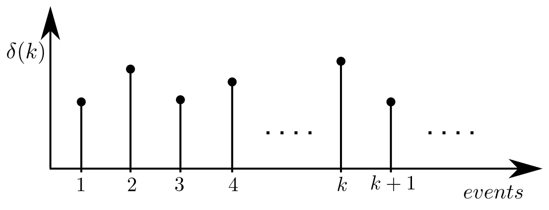

Figure 2: A sequence of event and their corresponding end-to-end delays.

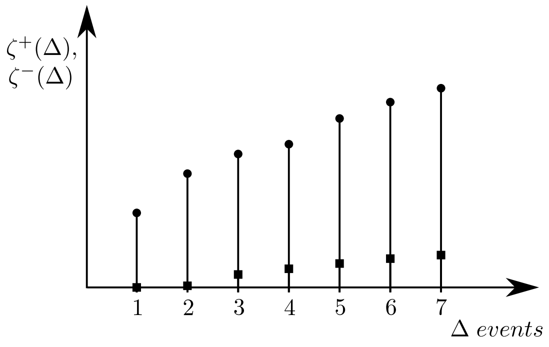

Figure 3: Maximum and minimum delay density.

Figure 3 shows a possible delay density. As a consequence of equations (1) and (2), the minimum and the maximum delay density is always a monotonic increasing function. The first bar ($\zeta(1)$) is defined as the maximum/minimum end-to-end delay that may occur in the system, which is an important parameter in the platform design. Moreover it must be ensured in the system validation that the maximum occurring delay density is under the specified in each value. Different methods, how this can be ensured is described in section 4.

3. DERIVE DELAY DENSITY FROM SYSTEM DYNAMICS

In this section we explain the steps to derive a delay density from system dynamics. First we need a kind of performance metric the system must fulfill. For control systems a minimum control performance is required, at least stability. The most common definitions are:

- as maximum overshoot of the step response
- as phase margin
- as integral of absolute error (IAE) or integral of squared error (ISE)
- as settling time in the step response

From the performance criteria a system designer has to decide the threshold value, which determines the acceptable range. In addition to a performance criteria a functional model of the system is needed. This can be in the form of transfer functions, or build out of functional basic blocks as for example in MATLAB/Simulink [?]. As the last information one need the maximum stimulation that is exposed to the system, in order to react. This could be a change in the reverence value of a control system, or a disturbance on a physical process. Would otherwise the system rest in a stable state, delays would not effect the control performance.

The way the system is stimulated also defines the region of interest (roi). As we are not able to analyze an infinity amount of samples, defined in equations (1) and (2), we have to focus on a relevant time interval, named as roi. For a step in the reverence value or disturbance, the roi includes all samples from the step until the system reaches steady state. Outside the range, delays have no effect on the signal sequence. For periodic stimulation, the roi is the time interval for the duration of a period. This requires a time-invariant system, which is true for most systems.

Without introducing further details in system theory, we present a very simple heuristic, to find the maximum delay density of the system. We construct the delay density stepwise, beginning with the delay for one sample. Therefore we increase the delay of one sample ϵ, while all other samples have a delay of zero. Notice that we search for the maximum delay in the platform designers view. In the perspective of a control designer, this is the minimum delay, for which our defined performance criteria is barely met. When this sample in the roi is found, it consequently defines the first value on the maximum delay density and therefore the maximum delay of one sample $\zeta^+(1) = max\{\delta(k)\}$.

To obtain the second value of the maximum delay density ($\zeta^+(2)$), we again analyze the roi, but with two consecutive samples delayed. One of the samples is delayed by the maximum delay of one sample ($\delta(k) = \zeta(1)$). For the second delayed sample ($\delta(k+1) = \epsilon$), ϵ is increased until performance criterion is no longer satisfied. The search is repeated vice versa, whereby the delay of sample ($delta(k) = \epsilon$) is increased, while the following sample is delayed by ($\delta(k+1) = \zeta(1)$). We receive two results, where we choose the smaller sum of the two delays for the second value of our delay density ($\zeta^+(2)$). A restriction, that bounds this value is,

$$\zeta^+(2) \leq 2 \cdot \zeta^+(1)$$

Otherwise equation (1) would not be satisfied.

The number of constellations we have to analyze increases for each additional value in the delay density. For the third value there are four possible arrangements. The reason is, that we search for constellations depending on already identified values. The values $\zeta^+(1)$ and $\zeta^+(2) - \zeta^+(1)$ remain neighbors. The third value is determined, by placing a delay (ϵ) before or after the group of $\zeta^+(1)$ and $\zeta^+(2) - \zeta^+(1)$. Table reftable:constellation lists all four possible constellations.

$\delta(k)$	$\delta(k+1)$	$\delta(k+2)$
$\zeta^+(1)$	$\zeta^+(2) - \zeta^+(1)$	ϵ
$\zeta^+(2) - \zeta^+(1)$	$\zeta^+(1)$	ϵ
ϵ	$\zeta^+(2) - \zeta^+(1)$	$\zeta^+(1)$
ϵ	$\zeta^+(1)$	$\zeta^+(2) - \zeta^+(1)$

Table 1: Possible constellations of delays that need to be analyzed to obtain $\zeta^+(3)$.

It is easy to recognize that the amount of constellations increases exponential with the number of delayed samples. The computation effort would therefore also increase exponentially. Fortunately, there are restrictions, that bound the problem. First, we only need to consider the amount of samples for the length of the roi. The second restriction is given by the already identified values of the delay density. A sample shall not be delayed longer than the identified maximum delay for one sample

$$\epsilon \leq \zeta^+(1)$$

Furthermore, for three delays, it must apply that $\epsilon \leq \zeta^+(2) - \zeta^+(1)$. The procedure, for a greater number of delayed samples is analogous. For a greater amount of samples, there are also more restrictions.

All these restrictions reduce the search space and retain the heuristic practicable. There exist probably more elegant ways to derive the delay density from the system behavior, including system theory knowledge or a kind of sensitivity analysis. In section 5, our heuristic is applied to a digital control system.

4. REAL-TIME ANALYSIS

The idea behind the delay density is, that it is no longer necessary to reannotate the timing in the functional model of the control system, to validate the control performance. After a delay density is extracted, the platform design and the associated real-time analysis can validate the correct behavior of the system.

There are different methods to obtain the timing of control task, executed on a platform. What they all have in common is, that they determine the response time of a task or a message. The delay, as used in previous sections, is the summed response time of all tasks and messages involved in the control loop.

We differ from analytically calculated response times and from simulated values. Both require information about the tasks and the platform they are executed on. These properties are explained hereafter.

execution time c. The execution time of a task or a message is the time it needs to complete it's job, when it holds the resource exclusively. The execution time is not a static value, as tasks can execute different commands, depending on the input data. There exist tool that determines the worst case execution time, for example the worst case execution analyzer aiT [?].

stimulation $\eta(\Delta t)$. The stimulation is a measure of how often a task or a message is activated in a specific time interval. Depending on the analysis method, there exist different models to describe a task stimulation [?], [?].

task chain and binding. Tasks rely on computations of previous executed tasks. Such dependencies produce task chains, on which the next task is activated, when the previous one has finished its job. The time when a task starts it execution is therefor event-activated. In distributed system architectures, every task is assigned a resource, on which it is executed.

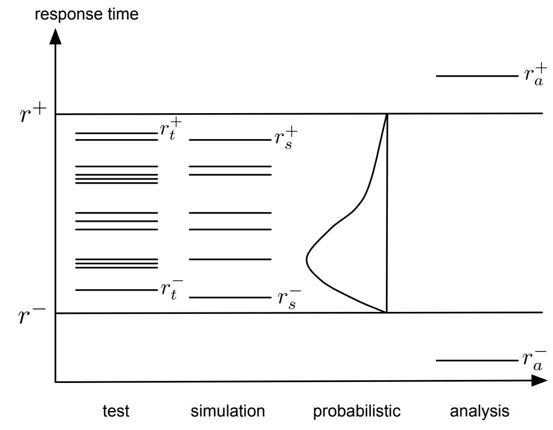

Figure 4: Results of different timing analysis methods.

priority and scheduling. On every resource a task is assigned a priority or a time slot. The operating system has a scheduler, that allocates computation time to a task, depending on its priority and the scheduling strategy.

Analysis methods

Analytic methods calculate from the listed information the maximum and minimum response time of the involved tasks. These are absolute bounds which are not exceeded. These values are theoretical, as it is not sure, if this bound is ever reached in reality. The maximum response times of tasks in the signal path can be summed to build the minimum and maximum end-to-end delay (δ^+, δ^-). If these values are available, the system performance is fulfilled, if the following inequality holds

$$\zeta^+(\Delta) \geq \delta^+ \cdot \Delta, \ \forall \Delta \in \mathbb{N} \tag{3}$$

Simulation methods

Beside real-time analysis, there exists simulation methods. The real-time simulation is an event-triggered simulation, with a stepwise execution. The longer the simulation runs, the more values for task's response times are obtained. The disadvantage of the real time simulation is the coverage problem [?]. The problem describes, that it is not possible to find the worst case in a simulation run. To validate a given delay density we have to proceed in the same way, as if measurements would be available and apply equation (1).

Figure 4 shows the difference of analysis results, simulation results and results obtained from a device under test.

Stochastic methods

New trends in real-time analysis take into account probabilistic parameter variations [?], [?] and [?]. Probabilistic

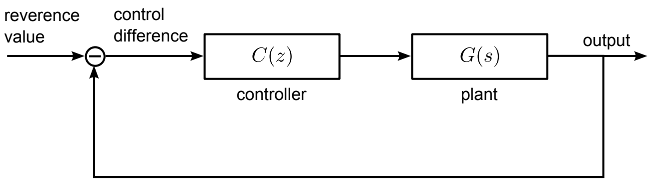

Figure 5: Structure of the control System.

approaches obtain in general less conservative results than the above mentioned real-time analysis. The work of [?] results in a stochastic real-time analysis to obtain typical worst-case response time. In the paper, a method is presented, that calculates "an exact bound of the maximum worst case response time occurrences in a given number of instances". This method is quite related to the delay density specification. By varying the number of instances and the threshold for maximum worst case response time, a maximum delay density can derived.

5. EXPERIMENTAL RESULTS

To show the beneficial effect of the presented methods, we derive the delay density for a real-world digital control system. The system consists of an unstable plant in form of an IT1-element with transfer function $G(s)$

$$G(s) = \frac{K_I}{T_1 \cdot s^2 + s} \tag{4}$$

with parameters $K_I = 1$ and $T_1 = 0.1$. The plant is stabilized by a discrete PID-controller $C(z)$ with discrete transfer function in z-domain

$$C(z) = P \cdot I \cdot T_s \cdot \frac{1}{z-1} + D \cdot \frac{N}{1 + N \cdot T_s \cdot \frac{1}{z-1}} \tag{5}$$

with proportional factor $P = 34.63$, integral factor $I = 49.34$, differential factor $D = 2.97$ and a filter-coefficient of $N = 60$. The sampling rate is chosen to $T_s = 10\,ms$. The complete control system and it's signals is displayed in figure 5.

To derive a delay density form the system, we must stimulate the system to react. This is done by a step in the reference value from zero to one, leading to the step response of the closed loop system on the output of the plant. From the step in the reference value until the output has settled, it takes $150ms$. As explained in section 2, our roi has an interval of 15 samples. The performance criteria for this example is chosen as a maximum overshot of 1.3 on the output.

As described in section 3, we begin to search for the minimum delay of one sample, that causes the step response to reach an overshoot of $y = 1.3$. Meanwhile all other samples are not delayed. For our system, a single delay of $\delta(k) = 7.7\,ms$ brings the output to the bound of our specified performance criteria. This delay defines the first value for our maximum delay density $\zeta^+(1) = 7.7\,ms$. To obtain the next values, we search for a further delay on adjacent samples. In this example, no further delay is allowed for the next samples, as the output would fail the criteria. After three samples with no further delay ($\zeta^+(2) = \zeta^+(3) = \zeta^+(4) = 7.7\,ms$), a relative high additional delay would be allowed. Since we have the restriction, that a sample must

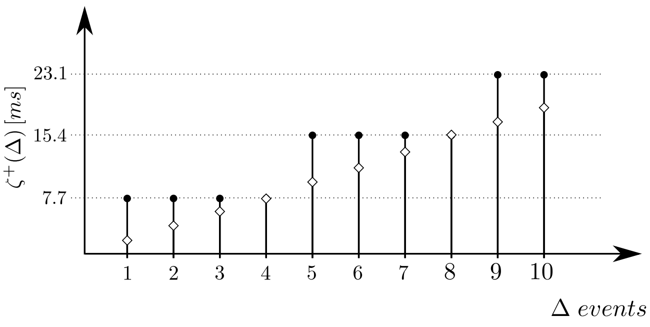

Figure 6: Maximum delay density for the presented control system and appropriate maximum delay for real-time analysis.

not be delayed longer than $\zeta^+(1) = 7.7\,ms$, the maximum delay density for five samples are $\zeta^+(5) = 15.4\,ms$. The following three values in the delay density keep again the same, based on the restrictions. This sequence repeats, so that the final maximum delay density can expressed in the following equation.

$$\zeta^+(\Delta) = 7.7 \cdot \left\lceil \frac{\Delta}{4} \right\rceil \, ms \tag{6}$$

When verifying the maximum delay density in equation (6) by a real-time analysis, we have to find the maximum allowed (δ^+). With equation (3), we can identify the maximum delay as

$$\delta^+ = \frac{7.7\,ms}{4} = 1.925\,ms \tag{7}$$

The derived delay density of equation (6) and the allowed maximum delay form equation (7) is displayed in figure 6.

When verified the delay density with real-time analysis methods, an end-to-end delay of $\delta^+ = 1.925ms$ fulfills our specification on the overshoot under all circumstances. There may be a higher delay, that also satisfies our performance criteria. For example a constant delay of $\delta(k) = 9\,ms$ results in an acceptable performance. However, a mechanism is needed, that guarantees that no shorter delays are possible. Without such a mechanism, the defined performance criteria would fail, whereas delays that fulfill equation (7) meet the criteria under all circumstances.

6. CONCLUSION

There exist a lot of systems, where timing negatively effects the functional behavior. The problem in many cases is, that in the deign process many developers are involved and timing issues appear in a late phase of the design phase. The problems could be avoided, if the system engineer would have a description with which he can express the system's timing requirements. With the delay density introduced in this paper it is possible to express the timing requirements in an early design phase and to serve a platform design as bounds.

We showed in this paper, how to derive a delay density from system dynamics. We also identified the relation to different timing analysis methods and how to verify a given delay density. An example system of a digital controller was taken, to demonstrate how a delay density can be obtained. Further the consequences on a real-time analysis were presented.

Future work includes a more efficient direct derivation of the delay density from the systems differential and difference equations.

7. REFERENCES

[1] AbsInt Angewandte Informatik GmbH. ait: worst-case execution time analyzers, January 2013.

[2] T. Bund, S. Moser, S. Kollmann, and F. Slomka. Guaranteed bounds for the control performance evaluation in distributed system architectures. In *Proceedings of the International Conference on Real-Time and Embedded Systems (RTES 2010), Singapore*, 2010.

[3] T. Bund, S. Moser, S. Kollmann, and F. Slomka. Jitter considerations for worst-case performance generation in digital controller design. *Cyber-Physical Systems &ndash; Enabling Multi-Nature Systems*, 4 2012.

[4] A. Cervin, B. Lincoln, J. Eker, K. Arzen, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 1–9. Gothenburg, Sweden, 2004.

[5] D. Goswami, R. Schneider, and S. Chakraborty. Co-design of cyber-physical systems via controllers with flexible delay constraints. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 225–230. IEEE Press, 2011.

[6] K. Gresser. An event model for deadline verification of hard real-time systems. In *Real-Time Systems, 1993. Proceedings., Fifth Euromicro Workshop on*, pages 118–123. IEEE, 1993.

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

[8] D. Henriksson, A. Cervin, and K. Årzén. Truetime: Real-time control system simulation with matlab/simulink. In *Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark*, 2003.

[9] S. Kollman, V. Pollex, K. Kempf, F. Slomka, M. Traub, T. Bone, J. Becker, et al. Comparative application of real-time verification methods to an automotive architecture. In *Proceedings of the 18th International Conference on Real-Time and Network Systems*, pages 89–98, 2010.

[10] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, jan 2003.

[11] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 2, pages 1319–1324. IEEE, 2002.

[12] MATLAB. *version 8.0.0 (R2012b)*. The MathWorks Inc., Natick, Massachusetts, 2012.

[13] J. Nilsson et al. *Real-time control systems with delays*. PhD thesis, Ph. D. dissertation, Department of Automatic Control, Lund Institute of Technology, 1998.

[14] S. Quinton, M. Hanke, and R. Ernst. Formal analysis of sporadic overload in real-time systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 515–520. IEEE, 2012.

[15] L. Santinelli, P. Yomsi, D. Maxim, and L. Cucu-Grosjean. A component-based framework for modeling and analyzing probabilistic real-time systems. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–8. IEEE, 2011.

[16] T. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. Wu, and J. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium, 1995. Proceedings*, pages 164–173. IEEE, 1995.

[17] E. Wandeler. *Modular performance analysis and interface-based design for embedded real-time systems*. PhD thesis, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, 2006.