

A Quality of Control Architecture and Codesign Method

Martin Sanfridson, Martin Törngren and Jan Wikander
{mis, martin, jan}@md.kth.se

Mechatronics Laboratory, Royal Institute of Technology, KTH, Stockholm

Abstract: An architecture and a method supporting codesign of flexible computer control systems are proposed in this paper. We call this approach Quality of Control to reflect the emphasis on control performance. Key components of the architecture include negotiation to optimize overall quality, admission control, explicit specifications of control characteristics and timing constraints, and on-line estimation of the control performance as a function of the actual system timing. The purpose is to give flexibility in terms of scalability, maintainability, configurability and graceful degradation.

1 Introduction*

The traditional design of an embedded system is rather static: a fixed design specification is mapped on a fully known platform, where a good a priori schedulability analysis requires the task set to be well-known [7]. A change in the specification or the platform will force a redesign. On the other hand, no one would endure an office LAN incapable of scalability, and most people expect a personal computer to have some plug-and-play capabilities. Loosely speaking, our intention is to require these properties for dedicated computer control systems, where the real-time, cost and dependability requirements are tougher.

The purpose of the proposed architecture is to cope with the complexity of software development for flexible control systems, primarily with respect to real-time requirements and control performance. To this end the architecture includes an adaptation mechanism with the aim of increasing the overall satisfaction, by rescheduling available resources. *Quality of Control*, QoC, is a runtime strategy where the quality (or performance) of a set of applications with real-time requirements is managed.

The proposed architecture is a continuation of [3][4] which focused on a specific hardware and scheduling policy. An experimental scalability layer was implemented on top of the commercial RTOS called OSE Epsilon including a link handler for the communication bus, [9]. The contribution of the currently proposed architecture is the top-down approach trying to tackle the problem of providing scalability, which in the best-case scenario could help decrease time to market, encompass applications having different kinds of real-time requirements, facilitate both a product family based on the assembly of modules and product upgrades.

A corner stone is the *temporal separation* of the optimization and scheduling, i.e. the computationally costly

optimization is either event triggered or invoked quite rarely compared to the applications. A *separation of concerns* does not only make the architecture easier to understand but also enables the construction of a middleware on top of many existing RTOS for distributed systems. A return to familiar grounds is the need for worst-case execution time estimations and hard deadlines, for the sake of predicting real-time behaviour in the short term, avoiding uncontrollable bursty overload that threatens to violate the *allowable* state space.

2 Application characteristics

In a computer control system, several kinds of applications can be found. The partitioning and allocation are primarily static in an embedded system, and the scalability features are relatively seldom used. Nevertheless, flexibility is the key factor of the architecture. A worst-case design is both desirable and workable in such a playground, open only to known and well specified extensions and replacements. The worst-case design should be accompanied by a consideration of the average behaviour which defines the quality or steady state control performance. Not all types of applications are susceptible to jitter and for some it does not make sense to optimize for e.g. period or latency, since it will not improve the perceived or measured quality. Feedback controllers are sensitive to the actual period, period jitter, delay and delay jitter [4]. A computer control system is typically safety-critical, which means critical tasks, including tasks closing the loop, cannot be allowed to exhibit excessive latencies.

A task can be characterized in three dimensions: mission-criticality, deadline requirement and timeliness requirement, see Figure 1. These are otherwise often lumped together. Mission-criticality is intimately connected to the usefulness of the entire embedded system: a mission-critical task must not be suspended or terminated. The deadline is an artifact, a design choice, that belongs to the implementation view of the computer system rather than the application. The idea to distinguish between urgency and importance has been pointed out before [1]. We informally define timeliness as the tightness of jitter measured relative to a suitable time constant. Strong timeliness implies difficulties in scheduling. The cube drawn with bold lines contains the most important and strongly timeliness sensitive applications, requiring hard real-time guarantees. The idea behind a differentiation is to improve chances for finding a feasible schedule, to increase the resource utilization and to facilitate design, e.g. to delimit the scope of validation and testing.

With a large number of soft, weak and non-mission-critical applications, it becomes easier to maximize the utilization and to find a feasible schedule. Control tasks

* This work has been supported in part by the Flexcon project funded by SSF, the Swedish Strategic research Foundation.

can have strong timeliness and soft deadline requirements (which covers so called firm tasks).

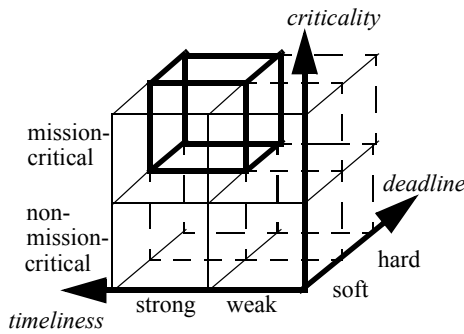


Figure 1. Task models: mission-critical vs. deadline vs. timeliness.

3 Requirements

A list of requirements for the architecture can be set up:

- *A scalable computer system.* The ability to add, remove and replace hardware and functionality is a property strongly associated with a modular design concept.
- *Adaptive scheduling strategy.* It must be possible to affect periods and delays, since the performance of a control application can be described as a function of these. Tasks and messages should be possible to reconfigure, to start and stop. It should be possible to optimize the use of e.g. processing units and communication channels.
- *Timing guarantees.* The global real-time scheduling must allow for a predictable processing in the short term, in most cases combined with application specific overload handling. This effectively means that a worst-case design is needed, i.e. a schedulability test. The timing behaviour must be measurable.
- *Separation of concerns.* A generic supervisory QoC manager should be separated from the applications as much as possible.
- *Separation of time scales.* The cascaded QoC manager should be invoked less often than the applications. This will preclude propagation of timing disturbances from the adaption of the scheduling to the closed loop dynamics, and it allows computationally intensive optimization algorithms.

4 Architecture overview

The basic building blocks of the architecture are depicted in Figure 2. At the bottom we find the hardware and RTOS. The scalability layer, if not part of the RTOS, manages the discovery, join and disconnection of applications and hardware. In the basic layer, the *elementary functions* (EF) are found. They perform the application specific work at a low level, that would be executed even on a bare bone system, e.g. a reading of a sensor or a control law computation. The *application function* (AF) constitutes in principle a specification of an application at an abstraction

level above the EF. The AF contains a task graph describing the logical and temporal coupling of its EFs. The AF also contains a specification of how the quality for the specific application is computed, and how the latency and jitter should be measured. Every EF is part of at least one AF and some EFs require mutual exclusion.

The *global function* GF, is a specification of the whole system, at an abstraction level above AF. It contains the function logic for cooperating AFs, functional mutual exclusion of AFs, exception handling, etc.

The global optimization strategy and admission control are generic but depend of course to some extent on the applications. In order to negotiate, basic *QoC control mechanisms* are needed to gather information regarding the timing behaviour, for book-keeping to minimize the management overhead, for monitoring deadline misses, for effectuating changes, etc. The QoC control mechanisms are non-application specific, but depend on the choice of RTOS, the protocols, the scheduling strategy, etc.

The union of the shaded boxes in Figure 2, as well as parts of the configuration and functional logic both in AF and GF, can be said to comprise the traditional setup, which excludes QoC related features. The left side of the figure, depicts the *QoC management specific* parts and the right side shows the *application specific* parts. The middleware is also split into one basic layer and an optimization layer. The AF and EF do not know the scheduling or optimization strategy being used. The EF does not know how to compute the quality, but can help measure the latency and jitter. To handle e.g. product variants or graceful degradation, alternative versions of an AF having the same purpose but using an alternative setup of EFs, can be deployed.

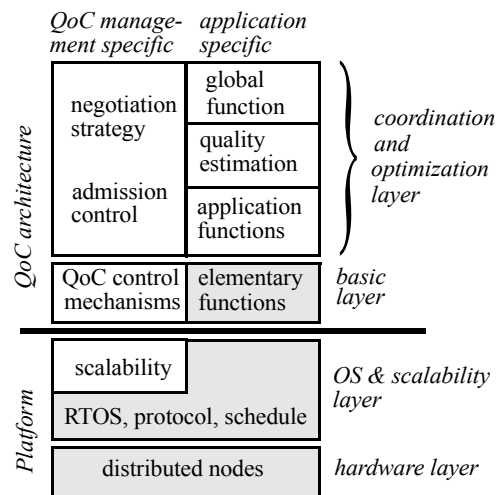


Figure 2. Basic building blocks of the functional architecture.

The diagram in Figure 3 depicts the data flow of the supervisory feedback loop. A control loop (AF₁) using three

EFs, is found at the top. The thick arrows represent the execution strategy control (triggering, scheduling and communication), and the supervision to measure the timing behaviour of an AF's task chain. The measured actual timing is used for the model based quality estimation. The negotiator compares the demands of different AFs with the estimated quality and tries to find new values for scheduling parameters such as period, priority, and offset.

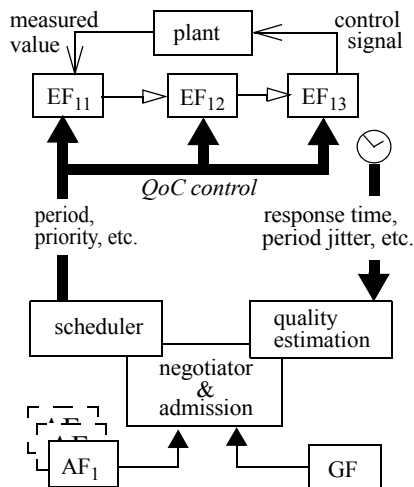


Figure 3. A flow diagram of the cascaded feedback negotiation (bold lines).

5 Quality estimation

Now, we need a strategy to optimize the total benefit or quality, B_{tot} . Applications have different objectives but contribute to the global goal. Since there are several types of cooperating applications (a controller is just one of these) using the same hardware, a so called *tutti-frutti problem* must be tackled. The way to express this mix of apples and bananas usually boils down to a sum $B_{tot} = \sum_k w_k B_k$ (or a similar expression), with a normalized benefit function B_k scaled by the weight w_k , for an application function AF_k . The drawback is naturally that the choice of w_k is left to the discretion of the system designer or user. There are many ways to devise an optimization routine which uses the benefits as objective functions, constrained by the allowable ranges (or values) of the period and delay given by the design specification. The optimization can for example be based on convex optimization, simulated annealing, fuzzy logic, and micro economic models. There are numerous of references on usable ideas like these in the vast literature of QoS [10]. However, the choice of optimization routine is hardly decisive for the success of the scalability concept.

A benefit function is needed for every AF participating in the negotiation. As a benefit function for a control application, a *quality function* integrating the continuous time squared and weighted control and measurement signals can be the origin, see e.g. [2] and [5]. A global clock, or a similar concept, is needed to accurately calculate the

latencies and jitter on-line. It is the quality associated with an AF that is of interest, and a single EF within the task chain cannot estimate the control performance, since it generally depends on an end-to-end delay.

The control performance function is based on steady state conditions, i.e. the average performance — the performance during a *long term* perspective. The control performance function, based on a special form of the H_2 -norm, is well applicable to time-varying periods and delays including transient timing errors. However, it is not sufficient to measure the short time perspective of transient timing errors. The system gain, or the H_∞ -norm, is actually better suited for this robustness issue, see e.g. [6], but there is a lack of supporting theory here. In order to guarantee the steady state performance, timing behaviour in the *short term* perspective has to be well-behaved and predictable. This can be enforced by introducing hard relative deadlines, which of course can be longer than the period. The deadline is determined at design time by the sensitivity to transient errors, delay jitter and constant delay. Violation of a hard (end-to-end) deadline implies violation of the domain over which the performance function is specified.

6 Admission control

Admission control belongs to the long term time scale and is a part of the QoS management. In order to uphold hard real-time requirements, it will occasionally be necessary to gracefully terminate a non-mission-critical AF, or to deny start-up. An AF with tough real-time requirements can on the other hand be of relatively minor importance, and be shut down to the benefit of mission-critical applications. It is convenient to arrange AFs in order of *mission-critical priority* or importance. Two classes can be distinguished within this range: a mission-critical and a non-mission-critical, Figure 1. The non-mission-critical AF does not cause a system failure if forced to terminate or denied admittance.

The task graph of an AF needs to be taken into account when its EFs are forced to terminate, since more than one AF can use the same EF, e.g. a (read only) sensor. Using multiple mutually exclusive AF having the same purpose but different configurations of EFs is a way to implement *graceful degradation* (limp home mode).

Overload handling belongs, on the contrary, to the short time scale. A traditional and simple solution to this unusual situation is skipping instances, but the situation violates the domain of the performance measure and jeopardizes the stability of the control loop.

7 Negotiation and activation

The separation of activation rates between optimization and task chain has several benefits: the optimization routine can run in the background being a computationally expensive soft and weak task, the QoC management stays more logically separated from the scheduling algorithm, and additional modes in the dynamics of the application are avoided. First, the changes done by the QoC manager

must be schedulable and second, a switch of control laws must be *bumpless*, to deny transient upsets in the control signal.

The activation can be *periodic* or *event-triggered*. The choice depends on the purpose and the information available, see Table 1. However, since the mapping from a change of parameters to control performance is difficult to obtain, the optimization is better based on a periodical feedback approach: the scheduling parameters are altered, the result is observed and fed back. The idea of a *one shot* negotiation, is that parts of the temporal design are postponed until a very late design time or even run time, when vital information readily becomes available.

Activation	Purpose	Information needs
One shot	Configuration of an assembly of modules	The real-time behaviour is <i>predictable</i> and the quality estimated. Default values can be used.
	Reconfiguration after upgrade/repair	
Event triggered	Scalability	
	Mode changes	
Periodic	Continuous supervision	The RT behaviour is <i>measurable</i> and quality estimated.

Table 1. Activation, purpose and information.

The activation interval of the periodic negotiation should be long enough to record enough data to analyse the control performance, and considerably long compared to the dynamics of the application, not to risk inducing additional dynamics into the control loop.

8 Codesign method

A main principle is the division of work in the codesign, cf. Figure 2. The role of the control engineer is to make a control synthesis and specify the allowable domain of the timing properties. For each AF, a benefit function must be supplied together with a domain for which the benefit function is valid. The deadline is determined at design time by the sensitivity to transient errors, delay jitter and constant delay.

The task of the computer engineer includes among other things the scheduling. Every EF has to be partitioned and allocated, presumably to dedicated hardware. Specific combinations of EFs must be approved, implemented, verified and tested to encompass any planned use of the scalability feature. Another idea of the separations outlined in Figure 2, is that the programmer implementing EFs should be liberated from reinventing generic functionality that could belong to a configurable middleware.

9 Implementation

The proposed architecture is not bound to a specific RTOS, middleware or hardware, and any suitable schedul-

ing policy can be considered. The architecture however requires that the implementation supports the requirements listed in Section 3.

Some of the required techniques are already available in commercial RTOSes, for example for supervising and measuring the execution times of threads, for starting and stopping threads, and supervising overruns, see e.g. [10]. The availability of a global clock in a distributed system largely facilitates measuring the actual timing as well as tailoring the desired timing (through scheduling and synchronization). However, measurements and estimations of end-to-end delays are also possible in asynchronous systems [8] based on run-time measurements of varying sub-delays part of the end to end delay.

10 Future work

The architecture presented here can be elaborated in many ways. One aspect to look more into is dependability, since it is generally thought of as being adversary to flexibility, and control applications are frequently safety-critical, cf. [10]. From a control point of view, a problem to study is that the switch of control laws implemented for different operating conditions (e.g. sampling periods) should be bumpless. The scalability feature also demands predictable timing behaviour, and not only for control applications. A key research topic in the real-time scheduling area is on-line schedulability tests.

11 References

- [1] Jensen D., "Asynchronous Decentralized Realtime Computer Systems", NATO Advanced study Institute on RT Computing, Sint Maarten, October 1992.
- [2] Nilsson J., "Real-time control systems with delays", PhD thesis, ISRN LUTFD2/TFRT--1049--SE, 1998.
- [3] Sanfridson M., "Problem formulation for QoS Management in Automatic Control", Technical Report TRITA-MMK 2000:3, Mechatronics Lab, KTH, March 2000.
- [4] Sanfridson M., "Timing Problems in Distributed Control", Licentiate Thesis, TRITA-MMK 2000:14, Mechatronics Lab, KTH, May 2000.
- [5] Sanfridson M., "Discretization of loss function for a control loop having time-varying period and delay", ISRN KTH/MMK/R--03/2--SE, Damek, KTH, 2004.
- [6] Skogestad and Postlethwaite, "Multivariable feedback control - Analysis and design", ISBN 0-471-94277-4, Wiley, 1996.
- [7] Stankovic et al., "Strategic directions in real-time and embedded systems", ACM Computing Surveys, Vol. 28, No. 4, December 1996.
- [8] Törngren M., Garbergs B. and Berggren H. "A Distributed Computer Testbed for Real-Time Control of Machinery", 5th ECRTS, Finland, June 1993.
- [9] Österman, J., "Scalability and QoS for Embedded Distributed Control Systems", MMK 2001:81 MDA181, KTH, Stockholm, 2001.
- [10] Artist road map on "Adaptive Real-Time Systems for Quality of Service Management", www.artist-embedded.org/Roadmaps/A3-roadmap.pdf