Ingredients for the Specification of Mixed-Criticality Real-Time Systems *

Raimund Kirner
University of Hertfordshire
United Kingdom
r.kirner@herts.ac.uk

Abstract

Models for real-time computing are available with different timing requirements. With the ongoing trend towards integration of services of different degrees of timing strictness on one single platform, there is a need to specify computing models for such scenarios.

In this paper we study the requirements to specify mixed criticality real-time systems (MCRTS). Mixed criticality systems have been studied intensively over the last years. Existing formulations of the scheduling problem for mixed criticality systems do not consider the different timing strictness requirements of the tasks. In this paper we argue that mixed criticality properties as well as real-time properties have to be considered together in order to provide the maximal utility of a system. Based on that argument we present a list of ingredients required for the specification of MCRTS. We outline conceptually, how a system can take advantage of having MCRTS specifications available. We present some examples to show the usefulness of specifying MCRTS properties for real-life systems.

1 Introduction

Computer systems with their timing behaviour being part of their correctness criterion are called real-time computer systems. In classical real-time models a deadline is called *firm*, if the result is of no value after the deadline, otherwise the deadline would be called *soft*. If damaging failure can happen after a firm deadline has been missed, the deadline is called *hard* [9].

To cope with the increasing complexity of cyberphysical systems, application vendors are increasingly demanded towards solutions with mixed timing requirements and criticality levels of services. The research community has reacted by providing platforms with sufficient ways to realise temporal isolation of tasks in order to execute the jointly, regardless of having different timing requirements. An example is the *ACROSS* MPSoC (multi-processor system-on-chip) platform which promotes time-triggered communication on an MPSoC network as the communication backbone [14]. The strong separation based on the time-triggered network supports isolation of cores, allowing also mixed-criticality integration. Another example is the *CompSOC* platform of Goossens et al., offering a virtual execution platform for each application running on it [3]. Design flows have been developed to map hard-real-time, soft-real-time, and non-real-time dataflow applications as well as Kahn process networks [7] onto the CompSOC platform. Composability for each resources is achieved by using preemptive time-division multiplexing (TDM) between applications for access to processor and NoC, which avoids interference between applications.

Chakraborty et al. have worked on mixed-criticality with timing and stability constraints based on FlexRay time-triggered networks [4]. They classify applications into two categories: (i) safety-critical control applications with stability and performance constraints, and (ii) time-critical applications with only deadline constraints. A schedule is constructed while at the same time optimising the sampling rate of the control applications.

Tamas-Selicean and Pop have worked on mixed criticality by considering hard-real-time tasks of different SIL [2] levels [16]. They need to solve (1) task-to-processor mapping, (2) task-to-partition assignment, (3) slot allocation at each processor, and (4) static schedule tables. Baruah et al. have also worked on different aspects of scheduling sporadic task sets with mixed criticality [1, 10].

Summarising the state of the art, there has been done considerable amount of work on scheduling applications with different (timing) criticalities. There are also some MPSoC platforms proposed that support the engineering of such systems. What we found to be missing yet is a system-

*The research leading to these results has received funding from the FP7 ARTEMIS-JU research project "ConstRaint and Application driven Framework for Tailoring Embedded Real-time Systems" (CRAFTERS). under contract no 295371 and the IST FP7 research project "Asynchronous and Dynamic Virtualization through performance ANalysis to support Concurrency Engineering (ADVANCE)".

atic concept of how to characterise system services of different criticality and real-time properties in a uniform way.

One of the limitations of the classical deadline-based real-time computing model is its narrow definition of utility, while a utility function provides more flexibility [6, 12]. For example, scheduling based on utility functions allows for more flexibility to maximise system utility [11, 8].

In Section 2 of this paper we present a uniform way of characterising systems consisting of services with different timing requirements and criticality properties. Using these properties, we present a generic definition of mixed time-criticality for mixed-criticality real-time systems (MCRTS). In Section 3 we describe the attributes a programming model should provide in order to specify mixed time-criticality. Section 4 we outline the generic principle of how to take advantage of MCRTS specifications. In Section 5 we show some use cases for mixed time-criticality systems. Section 6 concludes this work.

2 Mixed Time-Criticality

In the following we clarify the concept of *mixed time-criticality*. A computer system typically has to provide different functionalities, which are usually summarised by its specification. We call these different functionalities *services*. A system may provide multiple services with different time criticality.

2.1 Temporal Service Utility

Central for the concept of mixed time-criticality are the timing requirements of real-time services. We characterise the timing requirements by the temporal utility functions. The temporal utility functions for latency, throughput, and jitter are called *service latency utility function* (SLUF), *service throughput utility function* (STUF) and *service jitter utility function* (SJUF). An important property of these temporal utility functions is that they acknowledge the difference of the timing metrics where a result utility could become zero and the timing metrics chosen as technical limit [8]. In the following we demonstrate the system specification by means of the SLUF, with the use of STUF and SJUF being being analogous.

A simple example a SLUF is shown in Figure 1. In this example the service utility is not only positive, but it also becomes negative after a certain time. A positive utility means that the system is able to provide a useful service. A negative utility means that the corresponding latency is outside the valid time range and as a consequence can result in a damage of the system. In the given example a service provided later than a certain time interval, denoted as *critical latency*, can provide system damage. This range of service latency is marked as *damaging failure* in Figure 1. The

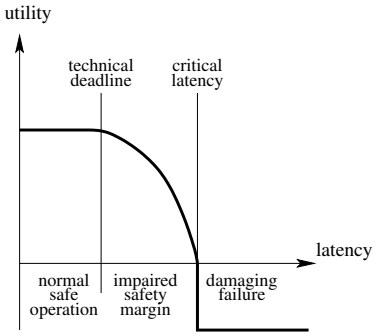

Figure 1. SLUF of a Real-Time Service

range of service latency marked as *normal safe operation* denotes the service latency considered as optimal operation, providing the maximum and constant service utility. There is also a latency range denoted as *impaired safety margin*, which does not yet cause any system damage, but exhibits a declining utility. The declining utility is a reflection of the reduced safety margin of the corresponding latency interval. When designing a system we usually set the deadline of a service to the end of the uncompromised service utility range, which we marked with *technical deadline* in the figure.

In the example the utility drops abruptly to full potential damage after the *critical latency*. However, in general a latency transition from *impaired safety margin* to *damaging failure* could also be more smooth, depending on the application. For example, with fuel injection in a motor there is a certain range of injection time where the motor becomes less efficient and is going to be increasingly worn out with later/earlier injection timing. In general, the transition from *normal safe operation* to *damaging failure* can be quite manifold, depending on the concrete real-time service. Research on real-time computing is most often oblivious of this diversity of transition characteristics of different real-time services.

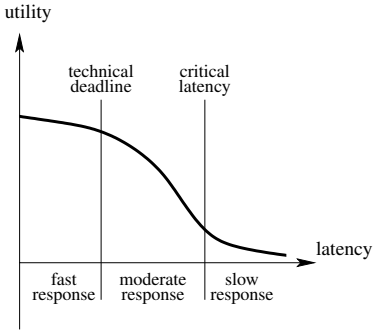

Figure 2. SLUF of a Real-Time Service without Potential Damage

Characteristic for real-time services is that their SLUF shows a significant value reduction for some range of the latency values. Figure 1 shows a typical example of how

the SLUF of a real-time service can look like. A negative SLUF value, as shown in the figure, is considered to represent damaging failure.

The existence of negative SLUF values is not a precondition for a service to be classified as a real-time service. For example, Figure 2 shows an example SLUF of a real-time service without any potential damage interval. However, the figure shows a significant variation among the SLUF values, which is a sufficient criterion to classify the service as real-time service.

On the contrary, to be classified as a non-real-time service, its SLUF may only exhibit a rather small variation. For example, the SLUF in Figure 3 can be considered as a non-real-time service, due to its small variation of the SLUF. There always will be a SLUF variation for any service, as no service is of use with an indefinitely long response time. For exactly that reasons one may not be able to cast a strict separation between what is a real-time service and what is a non-real-time service. However, the comparison of the SLUF graph of Figure 2 and Figure 3 should make it sufficiently clear of when it is adequate to speak of a real-time service or of a non-real-time service.

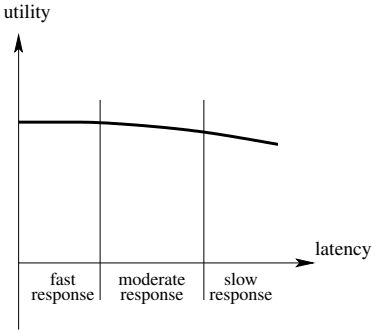

Figure 3. SLUF of a Non-Real-Time Service

2.2 Performance Characteristics

System services are normally connected with requirements on performance metrics, like *throughput*, *latency*, and *jitter*. The throughput describes how many input elements a system is able to process per time. The latency describes the delay between arrival of data at the system input and the release of the processed results at the system output. The jitter describes the variation of the latency.

These three performance metrics are the basic performance characteristics. However there is a difference on how the performance characteristics are being evaluated. For example, for real-time services it is important to consider the boundaries of the performance metrics. Foremost the minimum throughput, the maximum latency, and the maximum jitter are important to ensure the timeliness of a real-time service. It is worth noting that by binding the maximum latency and jitter, we implicitly also bind the minimum latency.

For non-real-time services it is typically the mean value of the performance characteristics that matters.

2.2.1 Primary Limits and Tolerance Ranges

Services can have besides their normal safe operation range another operation range of *impaired safety margin*, as shown by the SLUF in Figure 1. A subrange of this impaired safety margin might be used as a so-called tolerance range to extend the operability of the service for operation situations of unexpected resource shortage, etc. Figure 4 shows an example of a tolerance range of a real-time service's latency. The concept of tolerance ranges for real-time requirements is a rather recent notion, which has been introduced in [8].

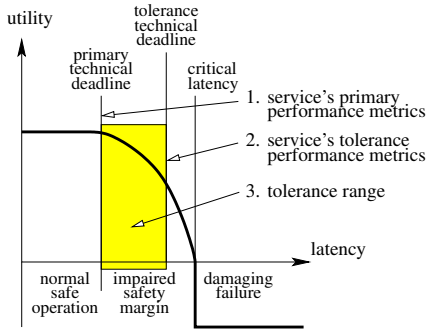

Figure 4. Example: Specification of a Real-Time Service with Tolerance Range

2.3 Deadline/Timing Strictness

Besides the specification of a service's performance metrics it is also useful to specify how strict the given performance metrics are meant to be. This information can help the system scheduler to decide for the best overall performance in case that not all services can be run at optimal performance.

2.3.1 Timing of Non-Real-Time Services

In case of non-real-time services there is no strict expectation about the timing of a service. Although in general the expectation is that a faster timing is considered as being better, for example, by providing a more responsive human user interface. However, in case of multiple non-real-time services there might be a performance trade-off necessary between the different services, as it might be not possible to schedule all services for maximal performance. This is typically the case when these services share resources, like processing cycles, bus access, etc. For example, if one service of higher priority takes in one round significantly longer

than expected, this different level of timing strictness can be used to find a best compromise for scheduling the lower-priority services.

Thus, in case of multiple non-real-time services it **also makes sense to specify their expected performance**, such that the system is able to schedule them according to their relative performance expectations. The performance metrics might be specified by mean values, or by intervals of their mean values.

2.3.2 Timing of Real-Time Services

For real-time services there are concrete expectations about the boundaries of performance metrics. Most important is the latency of real-time services, which might be specified as maximum allowed latency or as a *latency interval* in case it is also important to specify the shortest allowed latency. Often the throughput of real-time services is important as well. The throughput might be specified by the minimum required throughput. Optionally the throughput of real-time services might be specified as an interval, in case it is important to limit the maximum processing rate as well. The allowed jitter is given by the allowed variability of the latency.

An important question about any specified performance boundaries of real-time services is how strict they are. In case that any timing violation of a service would put the system at high risk of damaging failure, the service is called a *hard real-time* service.

A soft real-time service will continue to have a positive SLUF value at least for some of the timing violations. Depending on the concrete application, there is a wide range of what probability of timing violations is still acceptable.

2.4 Service Criticality

Another aspect of the service utility is the categorisation of its possible negative value, defining the criticality of the service. A service with a possibly high negative utility is of high criticality. Similarly, a service with none or very low possible negative criticality is of low criticality. In between, the service might be classified of medium criticality. Our classification into low, medium, and high criticality is meant to demonstrate the concept. In real application domains there are industry standards that concretely define the set of different criticality levels and how their classification is done. For example, the DO-178B standard of the civil avionics domain defines five *Design Assurance Levels* (DAL) [13], the ISO 26262 standard of the automotive domain defines four *Automotive Safety Integrity Levels* (ASIL) [5], and the IEC 61508 standard of the automation domain defines four *Safety Integrity Levels* (SIL) [2].

2.5 Mixed Time-Criticality Systems

Based on the concepts described in this section, we can now define mixed time-criticality systems as follows:

Definition 2.1 (Mixed Time-Criticality) *is a property of systems comprising multiple services, of which at least one service is a real-time service, and at least one of the the following properties holds:*

1. *the services include different values of performance metrics or tolerance ranges,*
2. *the services include different levels of timing strictness,*
3. *the services include different criticality levels.*

Definition 2.1 states that a system with mixed-time critical services has services of different importance and/or services with different timing strictness or performance requirements. These three properties are linked together via the SLUF.

All these properties will be taken into account by the scheduler to maximise provision of required services. The mixed time-criticality information can be used to prioritise services in case the available resources are not sufficient to provide all services.

3 Specification of Mixed Time-Criticality

In Section 2 we have introduced the concept of mixed time-criticality and discussed it constituents. In the following we outline what specifications the source code of an MTC system could include in order to specify the mixed time-criticality of the system's services. We discuss this aspect here on a rather conceptual level, without focusing on a specific programming language.

Since the specification of mixed time-criticality properties has to be done for each individual service, it is natural to add the information to that software regions that represent that service.

3.1 Specification of Performance Metrics

As described in Section 2.2, the performance of a service might be described by its latency, throughput, and jitter. We propose the specification of these performance metrics by a primary specification and an optional tolerance range, as explained in the following. As described in Section 2.2.1, the so-called *tolerance-range* is meant to describe an additional operational range of reduced but still acceptable service utility. This tolerance range can give the system a means to decide at runtime in case of unexpected resource shortage.

Specification of throughput [Hz]:
`mean/minimum/range`, optional `tolerance-range`
For non-real-time services we specify the expected mean value of throughput. For real-time services we specify the required minimal value of throughput, but in case the maximum value of throughput also needs to be bounded, we specify the throughput as a range.

Specification of latency [ms]:
`mean/maximum/range`, optional `tolerance range`
For non-real-time services we specify the expected mean value of latency. For real-time services we specify the required maximal value of latency, but in case the minimal value of latency also need to be bounded, we specify the latency as a range.

Specification of jitter [ms]:
`maximum`, optional `tolerance-range`
To bound the jitter we specify the maximum jitter. Optionally, we might want to specify an additional tolerance range for the jitter beyond the primary maximum value.

The specification of all three performance metrics is not mandatory. One might only specify those for which requirements exist.

3.1.1 Specification of Timing-strictness

As described in Section 2.3, it is useful to specify how strict the given performance metrics are meant to be. In addition, in Section 3.1 we introduced two classes of performance metrics, the primary one and an optional one for reduced but still acceptable service utility. To support that we propose the following classifier of timing strictness:

Specification of Firm Deadline: `FIRMRT`
In case that no violation of the technical deadline is considered viable, the timing strictness is firm real-time. With this timing strictness a specification of a tolerance range for performance metrics is not meaningful, as the service is expected to be always within the primary performance limits.

Specification of Soft Deadline: `SOFTRT(p1)`, optional `SOFTRT-tolerance(p2)`
If a violation of the primary performance limits within a certain probability is considered viable, then we speak of soft real-time systems. The parameter $p1$ is the probability of violating the primary performance limits, and $p2$ is the probability of violating the performance limits of the optional tolerance range. This violation probability is given as violations per hour, with ultra-dependable systems like civil avionics requiring a value smaller than 10^{-9} as there the overall probability of any type of failure is limited to 10^{-9}. The optional property `SOFTRT-tolerance`($l2$) makes only sense if also any tolerance range of a performance metrics has been specified. To make use of the tolerance range it is required that $p1 > p2$.

Specification of Non-Real-Time: `NONRT`
In case a service is considered to be practically non-real-time, then it does not have any timing strictness, expressed by the attribute `NONRT`.

3.1.2 Specification of Service Criticality

As explained in Section 2.4, also the service criticality is an important decision-criterion for scheduling the processor resources in case of only insufficient resources being available to run all services at optimal performance. We propose the specification of criticality levels in a generic way using integer values:

`CRIT(k)`

with $k = 0$ representing the lowest criticality. A meaningful specification of service criticality levels has to be compliant with the domain-specific safety standards, as mentioned in Section 2.4. For example, if the safety standard uses four levels, one might use four criticality levels as a direct relation, or use even more, if the engineering approach would need a more fine-grained subdivision, e.g., to derive scheduling priorities.

4 Optimising Service Quality of Mixed Time-Criticality

The primary purpose of this paper is to study mixed-criticality real-time systems (MCRTS) and list the different requirements needed to specify them. In this section we want to give an outline of how this information can be used in order to provide an optimisation framework to maximise the utility of an MCRTS. This optimisation framework will provide more flexibility than those listed in the state-of-the-art in Section 1, as it allows to trade timing properties to improve the set of feasible high-criticality tasks.

The conventional scheduling problem for mixed criticality systems can be seen as maximising the number of schedulable services weighted by their criticality as shown in Equation 1:

$$SUF_{opt,mc} = \max \sum_{i \in \text{SERVICES}} CRIT(i) \wedge \text{SCHEDULABLE}(i) \quad (1)$$

We propose to extend this goal function for MCRTS to also consider the relevant timing properties. For example, if we consider the latency as important, we have the additional freedom to trade latency along its tolerance range (see Section 2.2.1) to get a wider set of critical services scheduled:

$$SUF_{opt,mcrts} = \max \sum_{i \in \text{SERVICES}} CRIT(i) \cdot SLUF(i, l_i) \wedge \text{SCHEDULABLE}(i) \quad (2)$$

In above goal function for MCRTS systems the optimisation parameter l_i represents the latency for service i and and $SLUF(i, l_i)$ represents the SLUF of service i.

This concept can also be further extended to take into account additional timing properties like throughput. Given our goal function for MCRTS, such an extension is straight forward. For example, to add throughput to goal function Equation 2 we just have to multiply the term with $STUF(i, t_i)$ where t_i would be an additional optimisation parameter representing the throughput of Service i and $STUF(i, t_i)$ represents the STUF of service i:

$$SUF_{opt,mcrts} = \max \sum_{i \in \text{SERVICES}} CRIT(i) \cdot SLUF(i, l_i) \cdot STUF(i, t_i) \wedge \text{SCHEDULABLE}(i) \quad (3)$$

Section 5 gives some ideas of how such a framework can be beneficial for real applications.

5 Examples of Mixed Time-Criticality

In the following we briefly describe different applications that fit to the concept of mixed time-criticality. As a general pattern, it can be also the case that the time-criticality of a service changes dynamically, for example, by changing the mode the system is in.

5.1 Services of Different Timing-strictness but Same Criticality

There are cases where a service is of high criticality, even though the timing strictness allows for some flexibility. For example, in an aircraft there are numerous services that are essential for the safety of the passengers, even though their utilisation is not bounded by a firm deadline.

For example, the cabin pressurisation in a plane is controlled by an outflow valve to create a comfortable environment for aircraft passengers. Controlling the cabin pressurisation is of high criticality, as a failure to do so can put the passengers' life at risk. But the service is of relatively moderate timing strictness, as its corresponding SLUF has a rather smooth transition from normal safe operation to damaging failure. Once the cabin pressure leaves the comfort zone, the passengers are still being able to compensate for a certain amount of pressure loss by hyperventilation [17]. This would give additional time to control the cabin pressure in order to avoid serious damage.

5.2 Services of Same Timing Strictness but Different Criticality

As a generic pattern, using a time-triggered (TT) communication interface enforces technical deadline with high timing strictness, imposed by the length of the so-called TT communication round. Any two services running on a node would both be required to adhere to the same timing strictness to function flawlessly. However, these two services might have different criticality levels. Based on these different criticality levels the local scheduler of a node can prefer the execution of the higher criticality service, allowing for a higher probability to maintain its service in case of unexpected increases of processing latency.

5.3 Services of Different Timing Strictness and also Different Criticality

Schrijver and Creemers from Philips Healthcare published a use case of X-ray treatment with image processing [15]. This use case has basically three services to provide: real-time image filtering (IF), real-time feature detection (FD), non-real-time post-processing (PP). The IF service is meant to provide a frame rate of 24 frames/second. The FD service is significantly more complex to calculate than IF, but at the same time has a lower update rate requirement than IF. The update rate of FD depends on the available free computing resources and should ideally cope with the moving speed of the relevant object to be highlighted in the video stream. The PP service is meant to run in the background when none of the IF and FD services is active. Without running the PP in background and instead executing them just at the end of the day when switching off the machine, would result in a considerable delay of the shutdown process. The three services also have different criticality levels, with IF having the highest criticality, and PP the lowest criticality.

6 Summary and Conclusion

In this paper we argued that the specification of mixed-criticality real-time systems (MCRTS) should jointly consider criticality properties and timing properties in order to maximise the utility of the system. Central for the development of the generic concept of mixed time-criticality has

been the uniform characterisation of hard-real-time, soft-real-time, and non-real-time services by their temporal service utility functions for latency, throughput, and jitter. These temporal service utility functions together with the criticality values to the different services provides a novel way to optimise the utility of MCRTS. Important to this additional flexibility is the concept of *tolerance ranges* of timing properties.

Currently we work on a coordination language for MCRTS that will include the ability to specify the properties identified in this paper. Further work will be on scheduling techniques that take advantage of the additional flexibility of MCRTS specifications to improve the robustness of MCRTS systems.

References

[1] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems*, pages 1–36, 2013.

[2] I. E. Commission. Functional safety of electrical / electronic / programmable electronic safety-related systems. IEC standard 61508, 1998.

[3] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. B. Nejad, A. Nelson, and S. Sinha. Virtual execution platforms for mixed-time-criticality applications: the CompSoC architecture and design flow. In editor, editor, *Proc. 5th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, pages 23–30, San Juan, Puerto Rico, Dec. 2012.

[4] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *Proc. Conference on Design, Automation and Test in Europe*, pages 1227–1232, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[5] ISO/DIS. Road vehicles – functional safety. ISO/DIS standard 26262, Nov 2011. International Standard.

[6] E. D. Jensen. Asynchronous decentralized real-time computer systems. In W. A. Halang and A. D. Stoyenko, editors, Real-Time Computing*, Proc. of NATA Advanced Study Insititute*, St, Martin, Oct. 1992. Springer Verlag.

[7] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Proc. IFIP Congress on Information Processing*, Stockholm, Sweden, Aug. 1974. ISBN: 0-7204-2803-3.

[8] R. Kirner. A uniform model for tolerance-based real-time computing. In *Proc. 17th IEEE Int'l Symposium on Object/Component/Service-oriented Real-Time Distributed Computing*, Reno, Nevada, USA, June 2014.

[9] H. Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2nd edition, 2011. ISBN: 978-1-4419-8236-0.

[10] H. Li and S. Baruah. Outstanding paper award: Global mixed-criticality scheduling on multiprocessors. In *Proc. 24th Euromicro Conference on Real-Time Systems*, pages 166–175, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[11] B. Ravindran and T. Hegazy. A predictive algorithm for adaptive resource management of periodic tasks in asynchronous real-time distributed systems. In *Proc. 15th Int'l Parallel and Distributed Processing Symposium*, Apr. 2001.

[12] B. Ravindran, D. E. Jensen, and P. Li. On recent advances in time/utility function real-time scheduling and resource management. In *Proc. 8th IEEE Int'l Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 55–60. IEEE, 2005.

[13] RTCA. Software considerations in airborne systems and equipment certification. RTCA/DO-178B, 1992.

[14] C. E. Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek. The ACROSS MPSoC - a new generation of multi-core processors designed for safety-critical embedded systems. In *Proc. 5th Euromicro Conference on Digital System Design (DSD)*, pages 105–113, Cesme, Izmir, Turkey, Sep. 2012. IEEE.

[15] M. Schrijver and M. Creemers. Running real-time and best-effort applications concurrently on common off-the-shelf hardware. In C. Grelck, K. Hammond, and S. Scholz, editors, *2nd HiPEAC Workshop on Feedback-Directed Compiler Optimization for Multicore Architectures (FDCOMA'13)*, Berlin, Germany, Jan. 2013. HiPEAC.

[16] D. Tamas-Selicean and P. Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Proc. 32nd Real-Time Systems Symposium (RTSS)*, pages 24–33. IEEE, 2011.

[17] Wikipedia. Cabin pressurization. web page: `http://en.wikipedia.org/wiki/Cabin_pressurization`, 2013. accessed online on 13th August 2013 at 21:00.