Timing Overhead Analysis for Fault Tolerance Mechanisms

Kai Höfig

hoefig@cs.uni-kl.de

Abstract: The growing complexity of safety-critical embedded systems is leading to an increased complexity of safety analysis models. Often used fault tolerance mechanisms have complex failure behavior and produce overhead compared to systems without such mechanisms. The question arises whether the overhead for fault tolerance is acceptable for the increased safety of a system. In this paper, an approach is presented that uses safety analysis models of fault tolerance mechanisms and execution times of its subcomponents to generate *failure dependent* execution times. This provides a detailed view on the safety behavior in combination with the produced overhead and allows a trade-off analysis to find appropriate fault tolerance mechanisms.

1 Introduction

Safety-critical embedded systems are ubiquitous in our daily lives and can be found in many sectors, such as automotive, aerospace, medicine, automation, and energy industries. They are becoming more and more complex due to increasing functionality and automation in industry. The corresponding safety analysis models also grow in complexity and level of detail. To increase the safety of such systems, redundancies are often used within a certain mechanism to tolerate faults in the redundant units. These so-called *fault tolerance mechanisms* are widely used concepts and have a known behavior. They produce an overhead, e.g., in terms of execution time, energy consumption, additional hardware, additional software, or additionally required components compared to systems without such a mechanism. The problem arises whether the overhead produced by a fault tolerance mechanism is acceptable for the increased safety of a system.

Since safety requirements often contain a deadline and an upper bound for failure probability, e.g., *the system has to provide its function within x ms with a failure probability of less than y*, this paper addresses the property of *overhead in time* of fault tolerance mechanisms. The central approach presented in this paper aims at the fact that a fault tolerance mechanism is in a certain *mode* with a specific execution time for some faults being tolerated. The mode itself depends on failure modes given by safety analysis models. The combination of the time consumed by a mode and failure modes provides a detailed prospect of the timing behavior for a fault tolerance mechanism. Thereby a trade-off analysis in terms of execution time is supported.

In section 2, related approaches are described. Since failure modes of safety analysis models are here combined with execution times, that section is divided into approaches which belong to the research area of *Worst Case Execution Time* and into approaches that support the process of *safety engineering*. In section 3, an example system is described that is used to introduce the safety analysis model of *Component Fault Trees*, which provides failure modes as an input for the approach presented in this paper. Section 4 is the central section of this paper. The example system is picked up to describe the problem of modeling failure-dependent overhead in time manually. The central approach of generating execution times for fault tolerance mechanisms according to failure modes of the safety analysis model is formalized and applied to the example system. The generated execution times allow a sophisticated view of the overhead in time for such mechanisms. Section 5 concludes this paper and provides a perspective for future work.

2 Related Work

In this section, related approaches are discussed. Since the approach presented in this paper crosses two research areas, this section is divided into approaches that belong to the field of *Worst Case Execution Time* and into approaches that belong to the research area of *Safety Engineering*.

Current approaches to WCET analysis can be divided into measurement-based approaches and static timing analysis approaches [WEE$^+$08]. In static timing analysis, the execution times of individual static blocks are *computed* for a given program or a part of it. In general, these approaches provide safe upper bounds for the WCET by making pessimistic assumptions at the expense of overestimating the WCET in order to guarantee deadlines for the analyzed program. Advanced approaches, e.g. like those presented in [FH04], encompass precise hardware models to reduce overestimation of the WCET as much as possible. On the other hand, measurement-based approaches do not need to perform any complex analysis. They *measure* the execution time of a program on real hardware or processor simulators. These approaches can, in general, not provide a safe upper bound for the WCET, since neither an initial state nor a given input sequence can be proven to be the one that produces *the* WCET. Both static timing analysis and measurement-based approaches do not encompass additional timing failure modes or failure probabilities for calculating probabilistic WCETs for several modes of an analyzed system. However, approaches can be found that split the WCET in a probabilistic way. In [BCP03], an approach is presented that calculates probabilistically distributed WCETs. Here, the authors concentrate on *how* the nodes of a syntax tree have to be calculated if the WCETs for its leaves are *given* with probabilistic distribution. In a later work, the approach is extended to also encompass dependence structures. The authors solved the arising problem of multivariate distributions by using a stochastic tool for calculating a probabilistically distributed WCET. To *determine* the probability distribution, there have been some initial approaches in probabilistic WCET analysis. In [BE00], the authors use a measurement-based approach. The central idea is to measure the timings of a task and to stochastically drive the statement that a WCET will provide an upper bound for a certain subset of the input space. This approach is extended for scheduling in [BE01]. In [BBRN02] and [NHN03], probability distributions are calculated for response times of the *Controller Area Network* (CAN) Bus. These approaches may provide input to the approach presented in [BCP03], but do not aim at calculating WCETs. In contrast, the approach presented in this paper *generates* timing failure modes for fault tolerance mechanisms. This approach can therefore be taken into account as input for previously described approaches such as [BCP03].

Current approaches that automatically deduce safety analysis models from an annotated system development model mainly shift the complexity from the safety analysis model to the system development model. These approaches solve the problem of inconsistencies between the two models by combining the failure logic with the system development model entities by using annotations [PM01, Gru06, Rug05]. Other approaches, like [Boz03] and [JHSW06], rely on a formally executable design model to find inconsistencies between the model and its specification. This procedure allows a high degree of automation, but the type of system model is quite limited and the approaches do not solve the problem of finding appropriate failure modes. Only a few approaches deduce failure behavior by semantically enriching the system development model. In [GD02], an approach is presented that supports the analysis at a high level by providing recurring safety analysis model constructs mainly for redundancy mechanisms. These constructs decrease the complexity of a manual safety analysis, but do not provide a solution for generating timing failure modes. In [dMBSA08], a larger design space is covered, but the high degree of detail in the safety analysis model is achieved at the expense of a large number of annotations. Besides that, this approach does not aim at generating failure modes, but is more dedicated to a preliminary safety assessment aimed at designing an architecture that fulfills the safety requirements. The approach presented in this paper belongs to the group of semantic enrichment, since parts of safety analysis models are *used* to deduce timing failures.

In the next section, an example system is introduced along with its safety analysis model as the running example of this paper.

3 Example System

The example system of this paper is a simple fault tolerance mechanism. This section is used to introduce the methodology of *Component Fault Trees*. This safety analysis model relates parts of a fault tree and failure modes to components and ports (see [KLM03]), what makes it an interesting model for combining execution times of components with their failure modes. The approach presented in this paper uses CFTs, but also different safety analysis models, such as *Generalized Stochastic Petri Nets* or *Markov Chains* may provide input for it.

The example system is structured as follows: figure 1 shows on the left side the SysML *Internal Block Diagram* of a sorting mechanism FTSORT that performs a fault tolerant sorting of an array using two different sorting algorithms executed in the components PRIMARY and ALTERNATE. The system FTSORT starts its operation by storing a received array within a checkpoint. Then the checkpointing sends the array to the PRIMARY and triggers this component to start processing. The PRIMARY sends its result to the DECIDER, which checks whether the array is sorted. If the array is sorted, FTSORT has a result. Otherwise, the array is unsorted and has a VALUE failure. The DECIDER triggers the

CHECKPOINTING to restore the input. The checkpointing then sends the restored former input to the ALTERNATE and triggers this component to start processing on the restored input data. The result of the ALTERNATE is then taken as the result of FTSORT.

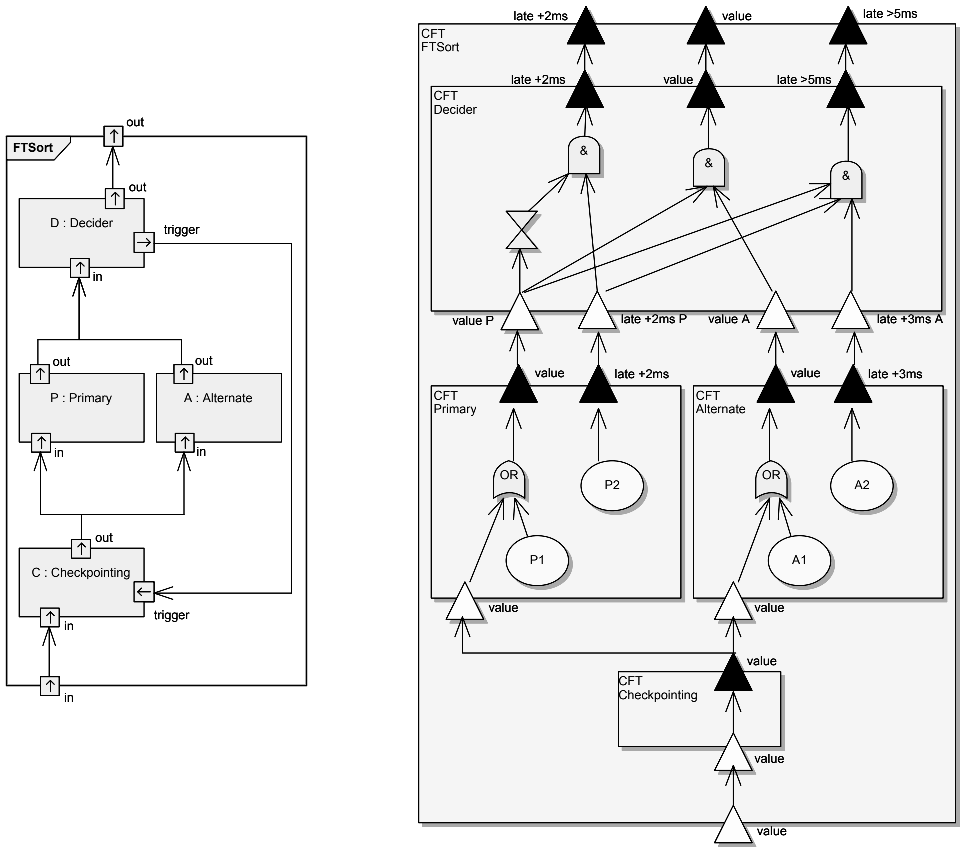

Figure 1: SysML Internal Block Diagram of FTSORT (left side) and related Component Fault Trees (right side).

The safety analysis model for this system is depicted on the right side of figure 1. Component Fault Trees (CFTs) are used to model the failure behavior of FTSORT[1]. Every subcomponent of FTSORT has an associated (sub)CFT. Every CFT has input and output failure modes (triangles). They are associated to the ports of the corresponding component (for example the output failure mode VALUE of the CFT PRIMARY is associated to the OUT port of the component PRIMARY). The port associations are straightforward and therefore not depicted to keep the figure readable. Within a CFT, conventional fault tree gates can be used to model the failure behavior of a component. For example, the com-

[1]For reasons of simplicity, no additional basic failure modes for DECIDER and CHECKPOINTING, a failure propagation via the trigger ports or other failure modes than VALUE and TIMING failures are included in this model.

ponent PRIMARY produces a VALUE failure if the basic failure mode P1 occurs or if it receives data with a VALUE failure (see CFT PRIMARY in figure 1). The system FTSORT produces a VALUE failure, if the basic failure modes P1 and A1 occur or if the provided input has a VALUE failure (see middle AND-gate in the CFT DECIDER in figure 1).

Additional to the VALUE failures, PRIMARY and ALTERNATE have two timing failure modes (see corresponding CFTs in figure 1). If the basic failure mode P2 occurs, the result of the PRIMARY is two milliseconds delayed. If the basic failure mode A2 occurs, the result of the ALTERNATE is three milliseconds delayed. The CFT DECIDER can be used to show that these two timing failure modes are insufficient to provide a sophisticated view on the timing failure modes of the entire mechanism. Both output failure modes LATE+2MS and LATE+3MS are used in this CFT. The DECIDER is delayed by 2 ms if the basic failure mode P2 is active and the PRIMARY component does not produce a VALUE failure (in this cause, the DECIDER would detect a failure and execute the ALTERNATE). This behavior is reflected by the leftmost AND-gate in the CFT DECIDER. Furthermore, the system is delayed by *at least* 5 ms, if the PRIMARY produces a VALUE failure, is additionally delayed by 2 ms and the ALTERNATE is delayed by 5ms (rightmost AND-gate of the CFT DECIDER). Nevertheless, these failure modes cannot be used to express the *overhead in time* for this mechanism, since the significance of the failure mode LATE>5MS, for example, is depending highly on the execution times of PRIMARY, DECIDER and CHECKPOINTING. If, e.g., the PRIMARY requires 500 ms to execute, the actual overhead in time is quite larger then 5 ms.

The safety analysis model indeed provides failure modes that involve a overhead in time, but generating absolute values can become an error prone and time-consuming task. Execution times have to be included in this calculation and additional timing failures complicate this process additionally. This is demonstrated in the next section and the central approach of this paper is presented that automates this process.

4 Analysis of Timing Overhead for Fault Tolerance Mechanisms

To tackle the problem of gaining a sophisticated view on the overhead in time for fault tolerance mechanisms, an approach is described in this section that automatically combines the execution times of components and failure modes of CFTs. As stated in the introduction, there are failure modes in safety analysis models that correspond with a certain *mode* of a fault tolerance mechanism. For example the failure mode VALUE of the component PRIMARY is corresponding to the mode in which the ALTERNATE is invoked to recover from this fault. If this failure mode is not active, the fault tolerance mechanism is in a different mode where only the PRIMARY redundant unit is executed. Such a set is here called a *run*. Since the here presented approach aims at execution times, the sets of executed elements for those two modes are needed as an input. Two run sets for FTSORT are depicted in table 1. The failure mode PRIMARY.OUT.VALUE relates to the value output failure mode of the CFT PRIMARY as depicted in figure 1. The consumed time for a run is based on the execution times of the subcomponents of FTSORT as follows: CHECKPOINTING: 1ms, PRIMARY: 2ms, ALTERNATE: 3ms, and DECIDER: 1ms. The execution

Run	Executed Elements	Consumed Time	Corresponding Failure Mode
1	C,P,D	4ms	**not** (Primary.out.value)
2	C,P,D,C,A,D	9ms	Primary.out.value

Table 1: Assumed execution times for the subcomponents of FTSORT

Run	Additional Time	Consumed Time	Corresponding Failure Mode
1	0ms	4ms	**not** (Primary.out.value)
1	2ms	6ms	**not** (Primary.out.value) **and** p2
2	0ms	9ms	Primary.out.value
2	2ms	11ms	Primary.out.value **and** p2
2	3ms	12ms	Primary.out.value **and** a2
2	2ms + 3ms	14ms	Primary.out.value **and** p2 **and** a2

Table 2: Possible Timings for FTSORT

times are also an input for this approach. As stated before, the output failure mode VALUE of the component PRIMARY corresponds to both runs.

Since the components PRIMARY and ALTERNATE provide additional timing failure modes LATE+2MS and LATE+3MS, six combinations of different timings can be possible. Those are depicted in table 2. The corresponding failure modes are a conjunction of the failure modes that correspond to a specific run as depicted in table 1 and the failure modes that correspond to an additional timing failure mode. E.g., the first run requires an additional execution time of 2 ms (what sums the consumed time up to 6 ms) if this run is executed *and* P2 is active (second row of table 2). This combination of additional timing failure modes and execution times provides a sophisticated view on the overhead in time produced by the fault tolerance mechanism. For quantified basic failure modes in the safety analysis model and a specific quantified requirement as described in section 1, the table can be used to directly perform a trade-off analysis of the fault tolerance mechanisms in terms of increased system safety and overhead in time.

Nevertheless, for every additional timing failure mode in one of the CFTs of PRIMARY or ALTERNATE, additional alternates, or additional timing failure modes elsewhere in FT-SORT, this table will expand rapidly. Calculating a larger number of such combinations manually is error prone and time consuming. Therefore, it is described in the following how this table can be generated if the set of runs is given, the corresponding failure modes are known, and additional timing failure modes are quantified in terms of absolute additional execution time.

Let $\mathcal{V}$ be the set of components that belong to a fault tolerance mechanism with

$$\mathcal{V} = \{V_1, .., V_{\bar{n}}\} \text{ with } |\mathcal{V}| = \bar{n} \in \mathbb{N}.$$

Each component V_i has one associated execution time, here labeled as t_i^0. With all optional additional timing failure modes of V_i, labeled as t_i^k with $k > 0$, the set $\mathcal{T}_i$ represents the different execution times for the component V_i with

$$\mathcal{T}_i = (t_i^0, .., t_i^{\bar{m}_i}) \text{ with } |\mathcal{T}_i| = \bar{m}_i + 1 \in \mathbb{N}.$$

Each t_i^k is a tuple that consists of a time and a failure mode fm with

$$t_i^k = (time, fm),$$

whereat $fm(t_i^k)$ represents the corresponding timing failure mode of the safety analysis model and $time(t_i^k)$ represents the consumed time if this failure mode is active. If necessary, $time(t_i^k)$ has to be combined with t_i^0 to reflect the execution time of the component in combination with additional required time for the failure mode. Additionally, $fm(t_i^0)$ is set to `true` to ease the later construction.

In the example, the component PRIMARY (P) has two execution times in $\mathcal{T}_P$: t_P^0 and t_P^1 with $time(t_P^0) = 2ms$, $time(t_P^1) = 4ms$, $fm(t_P^0) = true$ and $fm(t_P^1) = late + 2ms$.

These sets of execution times and their corresponding failure modes are used in the following to generate the different execution times of a run. The set $\mathcal{R} = (run_1, .., run_s)$ is the set of runs as described in the example. Each run is a tuple of executed components that belong to this run and corresponding failure modes that determine this run with

$$run_i = ((V_i, .., V_j), failuremodes), V_i, .., V_j \in \mathcal{V}$$

In the previous example, the corresponding sets for the runs as depicted in table 1 have the following form:

$$run_1 = ((C, P, D), not(Primary.out.value))$$

$$run_2 = ((C, P, D, C, A, D), Primary.out.value)$$

Using this sets, the all possible combinations of executions can be deducted. Let the set $\Omega(run_k)$ hold all combinations of execution times of components that are possible for a certain $run_K \in \mathcal{R}$ with

$$\Omega(run_k) = \{\omega = (t_m, .., t_n) \quad | \quad m \leq i \leq n, \; run_k = ((V_m, .., V_n), fm), V_i \in \mathcal{V}, \; t_i \in \mathcal{T}_i\}.$$

For the example system, the corresponding set for run_1 is $\Omega(run_1) = ((t_C^0, t_P^0, t_D^0), (t_C^0, t_P^1, t_D^0))$, since only the CFT for the PRIMARY has an additional timing failure mode. To generate tuples of execution times and corresponding failure modes for an ω_i, the execution times are summed up and the failure modes are combined.

In the following, execution times are combined by simply summing them up. It is here important to mention, that this is in general not applicable to approaches that generate *Worst*

Case Execution Times (WCETs), since for two components A and B, $WCET(A+B) \neq WCET(A) + WCET(B)$. This is true for many approaches that encompass processor states and stack values for calculating WCETs. To use such complexer WCET approaches in combination with our approach, the execution time for an entire run has to be calculated by a WCET tool. But to demonstrate our approach, we assume the independence of the given timings and for an $\omega \in \Omega(run_k)$ with $\omega = (t_m, .., t_n), t_i \in \mathcal{T}_i$, the consumed time is

$$time(\omega) = time(t_m) + .. + time(t_n).$$

The corresponding failure mode that models the execution of a specific combination of timing values for a run $\omega \in \Omega(run_k)$, is a conjunction of the specific failure mode of the run, $fm(run_k)$ and all failure modes that are active to result in the specific time behavior. The symbol $\wedge$ is here used to express the Boolean AND. The corresponding failuremode is then

$$fm(\omega) = fm(run_k) \wedge fm(t_m) \wedge .. \wedge fm(t_n).$$

In the example system, we previously constructed the set $\Omega(run_1)$ with $\omega_2 = (t_C^0, t_P^1, t_D^0)$. Since $fm(t_C^0) = fm(t_D^0) = true$, $fm(t_P^1) = p2$ and $fm(run_1) = not(Primary.out.value)$, the corresponding failuremode for ω_2 is:

$$\begin{aligned} fm(\omega_2) &= not(Primary.out.value) \wedge true \wedge p2 \wedge true \\ &= not(Primary.out.value) \wedge p2. \end{aligned}$$

Using this construction, the different execution times are combined with failure modes in a fashion as described for the example system at the beginning of this section. The construction of different alternate executions provides a sophisticated view on the overhead in time of fault tolerance mechanisms. The automated construction of possible alternates is less error prone than a manual approach and is capable to handle a larger amount of combinations and failure modes. In the next section, we conclude this paper and provide an outlook for future work.

5 Conclusion and Future Work

In this paper, an approach is presented that uses timing failure modes of safety analysis models and execution times of *modes* of fault tolerance mechanisms to automatically deduce different execution times along with their corresponding failure modes within the safety analysis model. This sophisticated view on execution times supports a trade-off analysis in terms of safety and overhead in time for fault tolerance mechanisms. Such a trade-off analysis can save costs when fault tolerance mechanisms cannot be identified as an appropriate solution by classic execution time analyses.

Section 3 provides an example for such a mechanism and motivates for combining execution times with safety analysis models to perform a trade-off analysis for the overhead in time of fault tolerance mechanisms. Since the number of possible combinations may be problematic for a manual approach, we present in section 4 an approach that automatically deduces execution times and corresponding failure modes. This requires clearly identified *modes* of fault tolerance mechanisms.

In our future work we concentrate on extending the approach to generic mechanisms, respectively to parallel redundancy. Furthermore, we want to be able to automatically deduce the execution behavior from a system model. This reduces the effort for applying the here presented methodology and also allows the use of WCET tools that are able to calculate tight upper bounds.

References

[BBRN02] I. Broster, A. Burns und G. Rodriguez-Navas. Probabilistic analysis of CAN with faults. *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, Seiten 269 – 278, 2002.

[BCP03] Guillem Bernat, Antoine Colin und Stefan Petters. pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. Bericht, 2003.

[BE00] A. Burns und S. Edgar. Predicting computation time for advanced processor architectures. *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, Seiten 89 –96, 2000.

[BE01] A. Burns und S. Edgar. Statistical analysis of WCET for scheduling. *Real-Time Systems Symposium, 2001. Proceedings. 22nd IEEE*, Seiten 215 – 224, dec. 2001.

[Boz03] M. Bozzano. ESACS: An integrated methodology for design and safety analysis of complex systems. In *Proc. of European Safety and Reliability Conf. ESREL*, Seiten 237–245, 2003.

[dMBSA08] M. A. de Miguel, J. F. Briones, J. P. Silva und A. Alonso. Integration of safety analysis in model-driven software development. *Software, IET*, 2(3):260–280, June 2008.

[FH04] C. Ferdinand und R. Heckmann. aiT: Worst-Case Execution Time Prediction by Static Program Analysis. In *Building the Information Society*, Jgg. 156/2004, Seiten 377–383, 2004.

[GD02] P. Ganesh und J.B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML SystemModels. *13th International Symposium on Software Reliability Engineering (ISSRE)*, 2002.

[Gru06] L. Grunske. Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. *Proc. Conf.Quality of Software Architectures QoSA*, 4214, 2006.

[IEC98] IEC61508. International Standard IEC 61508, 1998. International Electrotechnical Commission (IEC).

[JHSW06] A. Joshi, M.P.E. Heimdahl, M.P. Steven und M.W. Whalen. Model-Based Safety Analysis, 2006. NASA.

[KK07] Israel Koren und C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann, San Francisco, 2007.

[KLM03] Bernhard Kaiser, Peter Liggesmeyer und Oliver Mäckel. A new component concept for fault trees. In *SCS '03: Proceedings of the 8th Australian workshop on Safety critical systems and software*, Seiten 37–46, Darlinghurst, Australia, 2003. Australian Computer Society, Inc.

[LPW09] Philipp Lucas, Oleg Parshin und Reinhard Wilhelm. Operating Mode Specific WCET Analysis. In Charlotte Seidner, Hrsg., *Proceedings of the 3rd Junior Researcher Workshop on Real-Time Computing (JRWRTC)*, Seiten 15–18, October 2009.

[NHN03] T. Nolte, H. Hansson und C. Norstrom. Probabilistic worst-case response-time analysis for the controller area network. *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, Seiten 200 – 207, may. 2003.

[PM01] Y. Papadopoulos und M. Maruhn. Model-Based Automated Synthesis of Fault Trees from Matlab.Simulink Models. *International Conference on Dependable Systems and Networks*, 2001.

[Rug05] Ana Rugina. System Dependability Evaluation using AADL (Architecture Analysis and Design Language), 2005. LAAS-CNRS.

[WEE$^+$08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat und Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.