https://doi.org/10.1145/3195970.3196018

Cross-Layer Dependency Analysis with Timing Dependence Graphs

Mischa Möstl and Rolf Ernst
Technische Universität Braunschweig
{moestl,ernst}@ida.ing.tu-bs.de

ABSTRACT

We present Non-Interference Analysis as a model-based method to automatically reveal, track and analyze end-to-end timing dependencies as part of a cross-layer dependency analysis in complex systems. Based on revealed timing dependencies of functional cause-effect chains, this method enables an automated FMEA inspection of timing behavior of individual functions. In consequence, this method can support safety-critical design processes w.r.t. the technical safety concept as mandated by safety standards such as ISO 26262. Our case-study from a state-of-the-art automated research vehicle and synthetic experiments confirm the applicability and scaleability of the proposed method.

ACM Reference Format:
Mischa Möstl and Rolf Ernst. . Cross-Layer Dependency Analysis with Timing Dependence Graphs. In *DAC '18: DAC '18:The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3196018

1 INTRODUCTION

Functional safety is defined as the absence of harm from unintended hazardous behavior w.r.t. the design intent (cp. [8, 1.51,Part1]). Hidden dependencies in highly-networked complex systems can trigger such malfunctioning behavior, and are one of the foremost design risks e.g. for automated vehicles. The ISO 26262 design process approaches safety in three phases: definition of *items*, a *functional safety concept* [8, Part3] to ensure the safety of the items, and a *technical safety concept* [8, Part4] that ensures the adherence of the implementation to the functional safety concept. This, however, creates a design problem for the elements of an item: For a truly conservative approach, a designer has to assume dependence if an element [8, 1.32, Part1] is shared within a system or among systems of a network [8, 1.129, Part1], unless required independence can be proved. This leads to a situation where at the beginning of the design all elements are dependent. Following the conservative design strategy, a designer has further to assume that any dependency leads to interference, i.e. failures causing malfunctioning behavior. The solution to this design problem is to systematically add design knowledge to be able to prove *freedom from interference*. To argue that an element A is free from interference from an element B, is either possible by a proof of strict non-interference or through a proof of safely-bounded interference, where the bounds must suffice the required safety level of element A.

As an example, consider a lateral guidance control that extends

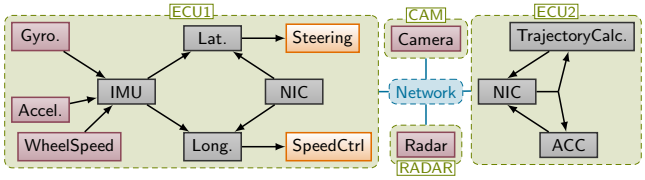


Figure 1: Functional Architecture of the research vehicle, with an updated lateral guidance functionality for automated driving.

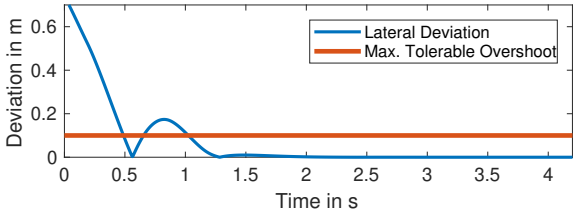


Figure 2: Lateral Deviation of the vehicle due to controller dead time. The red line indicates the maximal tolerable overshoot [19].

an Adaptive Cruise Control (ACC) system to an Inertial Navigation System (INS) for automated driving. Figure 1 shows the simplified functional architecture and platform that implements longitudinal (lower half) and lateral (top half) guidance control. Environment perception for trajectory planing is based on camera and radar sensors, as well as on motion estimation provided by an Inertial Measurement Unit (IMU) based on sensor-data fusion from gyroscopes, accelerometer, and wheel-speed sensors. [1] The function and communication path mapping to Electronic Control Units (ECUs) already exhibits dependencies between the longitudinal and the lateral guidance due to sharing common elements such as their ECU, and the data provided by the IMU. W.r.t. intended behavior of the lateral guidance, a Hazard and Risk Analysis (HARA) in an ISO 26262 process for automated vehicles in freeway traffic has identified a maximal tolerable overshoot of 0.1m [19]. To achieve this control performance, the function depends on strict timing constraints for its cause-effect chains to avoid controller instability or overshoots. Unpredictable and unanticipated timing such as unconsidered dead time that implies phase are potential causes for unstable and undesired behavior [13]. In the example, this means that the safety of lateral control depends on timely measurement values from the IMU and reference values from the trajectory calculation. Thresholds for e.g. the maximum tolerable data age (i.e. dead time) can be derived by functional analysis, and serve as timing

[1] We are aware, that this example is incomplete w.r.t. functionally-safe fully-automated driving. For illustration, we omitted parts of the safety concept, such as hardware redundancy.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06.
https://doi.org/10.1145/3195970.3196018

constraints for the implementation. Figure 2 shows the resulting control deviation for the lateral controller if measurement values from the IMU are delayed beyond these thresholds. Consequently, the implementation and integration must avoid interference caused by other elements on these constraints. In this example, either strict non-interference or bounded interference must be guaranteed for lateral and longitudinal control such that the violation of safety requirements (e.g. timing) can be avoided.

In order to claim *freedom from interference* and resolve an existing dependency, the system knowledge must be detailed to argue about the *relevance* of a dependence relation. E.g., [8, Part4] proposes Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) to identify paths of failure propagation. The goal of the technical safety concept is thus to ensure the independence of the function timings in the sense that they solely depend on artifacts and proofs that fulfill the same or higher levels of assurance than itself. The goal of dependency analysis is to support this action as, with a growing number of dependencies, (manual) methods such as FTA and FMEA become more challenging.

Problem Statement: While timing analysis of individual tasks is an established technique, systematic tracing and quantification of dependencies across layers to prove irrelevance and, hence, non-interference in the context of a safety concept is an open issue. To consider two items independent, we need dependency analysis covering **all** potential paths to prove that **no** functionally relevant dependency can possibly exist between these items. The search for these paths has to be conducted over the complete architecture of the design, which consists of an interconnected combination of several architecture layers describing functional, software, and hardware behavior of the system. Especially the question of how the data quality of timing-analysis input parameters influences bounds of functional requirements via timing dependence is of interest. For assessing a dependency's relevance, it is often not the actual result of a timing analysis that matters but the fact on what data basis a bound is derived. Regardless of the fault's cause, it is important how it affects this data and how it influences end-to-end behaviour in complex Cyber-Pysical Systems (CPSs).

Contribution: In this paper we show how to systematically link system parameters between different model layers that influence the timing behavior of the system (section 3). Based on the presented data structure of a Timing Dependence Graph (TDG) (section 4) we demonstrate how timing effects in complex CPSs can be explored. By the use of TDGs, we are able to reduce the problem complexity to reachability problems in the graph structure and ultimately support an ISO 26262 safety process (section 5).

2 RELATED WORK

A number of works have investigated the sensitivity of timing-analysis results, i.e. they argue about the robustness of computed bounds based on varying input parameters. Particularly, [4] computes exact results for strictly periodic systems w.r.t. varying execution times and periods, while other approaches rest on systematic searches in the solution space for entire systems [15, 22], or specifically target activation patterns, i.e. one input parameter type [11]. All of these methods have in common that sensitivity analysis is computationally intensive and neglects the relevance aspect of dependencies, i.e. the approach is not suitable to directly show the dependence of two parameters. It can merely show a correlation in the results and therefore cannot show a separation of different levels of safety criticality.

Strictly time-driven architectures enable isolation of the timing behavior of functions [9]. While this approach reduces dependency analysis to fewer layers and effects, it is not work conserving, is sensitive to later changes, and complicates reuse. While this is a potential alternative, vehicle networks and ECUs still mainly operate on priority driven scheduling, e.g. CAN, automotive versions of Ethernet (TSN), as well as the AUTOSAR OS itself [2], even for safety critical functions. Moreover, time-driven architectures still require a proof of non-interference, at least in case of shared elements (cameras, function blocks, ...). A further discussion is beyond this paper.

To avoid unintended hidden dependencies already in the design of complex CPSs, contracting methods seem a promising candidate. In these methods, software components are equipped with a model-based description of their behavior and a set of assumptions which needs to be fulfilled to provide a set of specified guarantees. A common application is the verification of pre and postconditions within a software architecture to guarantee functional properties, e.g. for safety. As this is carried out within a single architecture layer, it is often referred to as horizontal contracting. However, contracting to guarantee vertical (also referred to as non-functional) properties such as timing behavior is still an open issue [16].

Furthermore, a number of techniques exist to guarantee and enforce timing behavior as expected by performance analysis. E.g. [7, 10] study how to enforce specified behavior on runtime inputs. While these methods are costly and limit performance when used throughout a networked vehicle, they can selectively be deployed to remove interference detected with the methods proposed in this paper.

3 CROSS-LAYER MODEL

In the initial example we motivated that solely from the mapping of a (sub)function to a common execution resource, a timing dependency emerges. To formally capture these and other dependencies on individual layers, we introduce our multi-layer system model, which reveals cross-layer dependencies. In principle, we treat each layer as a graph, where elements from one layer can have a bidirectional mapping relation with nodes and edges on another layer.

In our example, the top layer models the system on a functional basis as in Figure 1. The functionalities are then decomposed into an executable functional model:

Definition 1. An *executable functional model* is a graph $\mathcal{RG} = (R, L, \xrightarrow{w}, \xrightarrow{r})$ with R as the set of runnables, L denoting the set of data labels and $\xrightarrow{w}, \xrightarrow{r}$ defining two precedence relations between runnables and labels and vice versa denoting writing and reading of labels.

Definition 2. A *runnable* $\rho_i \in R$ is executable workload and defined by a tuple (C_i^+, C_i^-, TR_i) whereas C_i^+ is a bound on its worst-case execution time (WCET), C_i^- a bound on its best-case execution time (BCET), and TR_i defines a triggering requirement for ρ_i.

In principle the executable function model corresponds to the

graph structure of e.g. SIMULINK models. Furthermore, cause-effect chains Ψ can be defined as alternating sequences of runnables and labels in order to annotate end-to-end timing requirements in the executable function model. We formalize two requirements: (a) the backward distance $d_{i,j}^{age}$ which is defined as the maximum time between a read event of ρ_j and the preceding write event of ρ_i on a common label (data age), and (b) the forward distance $d_{i,j}^{rea}$ which defines the maximum time between a write event to a label by ρ_i and its subsequent read by ρ_j (reaction-time). End-to-end requirements on a cause-effect chain can then be defined as sums of d^{age}/d^{rea} and the processing time between the read and write events of runnables. Note that, e.g. maximum data age can be directly mapped onto a delay block, e.g. in SIMULINK to account for dead time [1].

The runnable semantic allows a fine-grained description of the workload and behavior of embedded software. However, runnables are typically not executed individually but aggregated in schedulable entities with a common trigger condition to minimize context switches and enable low-overhead OS implementations, e.g. as in AUTOSAR [2]. These schedulable entities are referred to as tasks, and correspond to the well-established notion of real-time tasks and respective analysis techniques [6, 21].

Definition 3. A *task* $\tau_i \in \mathcal{T}$ is defined by a tuple $(C_i^+, C_i^-, \delta_{i,in}^+, \delta_{i,in}^-, \mathcal{S}_i)$, where C_i^+ denotes the WCET and C^- the BCET respectively, $\delta_{i,in}^+, \delta_{i,in}^-$ are functions that are event model abstractions of concrete execution traces that capture the maximum/minimum time interval between n consecutive activation events, and $\mathcal{S}_i$ is a set of three tuples (m, t, l) with $m \in \mathbb{N}$ denoting a unique sequence number of a read/write access of type $t \in \{read, write\}$ to a label $l \in L$.

Definition 4. A *task graph* is a tuple $\mathcal{TG} = (\mathcal{T}, \mathcal{R}, \rightarrow, \xi)$ with $\mathcal{T}$ representing the set of tasks, $\rightarrow$ defining a directed precedence relation between tasks such that output/completion events become activation events of the preceding one, $\mathcal{R}$ as the set of computation resources, and ξ as a right total function that maps each task to exactly one resource that provides execution time to its tasks according to its scheduling policy.

We also introduce mappings between $\mathcal{RG}$ and $\mathcal{TG}$ to complete the cross-layer model (together with the functional model that maps onto $\mathcal{RG}$). Each runnable maps to exactly one tasks, where the runnable'sWCET and BCET contributes to the one of its task and the order of runnables is fixed within each task. The read and write events of a label map to the read and write events of a task.

Figure 3a shows the executable function model next to the task graph in Figure 3b for our example. The cycle times of the individual runnables are indicated by their fill color as they correspond to tasks with names that indicate their period. Further, we assume Rate Monotonic (RM) Static-Priority Preemptive (SPP) scheduling for ECUs 1 and 2, while the Network is priority-based Ethernet.

Based on the description of the workload and its parametrization in the model, several bounds on the timing behavior can be computed, e.g. by Compositional Performance Analysis (CPA). To analyze an entire system of resources and tasks, CPA first analyzes all tasks τ_i on each resource, computing its worst-case response time (WCRT) R_i^+, best-case response time (BCRT) R_i^- and output event models $\delta_{i,out}^+, \delta_{i,out}^-$ based on the execution times bounds C_i^+, C_i^-, input event models δ_i^+, δ_i^-, and the scheduling policy of the resource $\xi(\tau_i)$. Intermediate results are the minimum and maximum q-event busy windows w_i^- and w_i^+ as well as the maximum number of backlogged activations q_{max}. Second, the computed output event models are propagated to their dependent tasks as new input event model. Subsequently, CPA iterates over these two analysis steps until the system converges, i.e. all output / input event models are stable and do not change anymore between two consecutive analysis steps [6]. How data age and reaction time for individual tasks can be computed based on CPA is e.g. described in [5].

4 TIMING DEPENDENCE GRAPHS

Dependencies, such as a data age/dead time requirement in the executable function model, can be hidden on other layers. In this section, we thus show how to enable traceability of dependencies across layers. As this is based on the scheduling behavior and its analysis, we define a data structure to capture the dependencies.

Definition 5. A *Timing Dependence Graph* is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of nodes $v_i, v_j \in \mathcal{V}$ and edges $e_k \in \mathcal{E}$ where each edge $e_k = (v_i, v_j)$ describes that v_j is dependent on v_i. Each node v_i either describes a task parameter $p \in P = \{C^+, C^-, \delta_{in}^+, \delta_{in}^-\}$ or an (intermediate) timing analysis result $r \in R = \{w^+, w^-, \delta_{out}^+, \delta_{out}^-, R^+, R^-, q_{max}\}$.

To convert parameters and results from the task model in nodes of the respective type we define two conversion functions:

Definition 6 (Conversion functions)**.** The *parameter conversion function* is a function

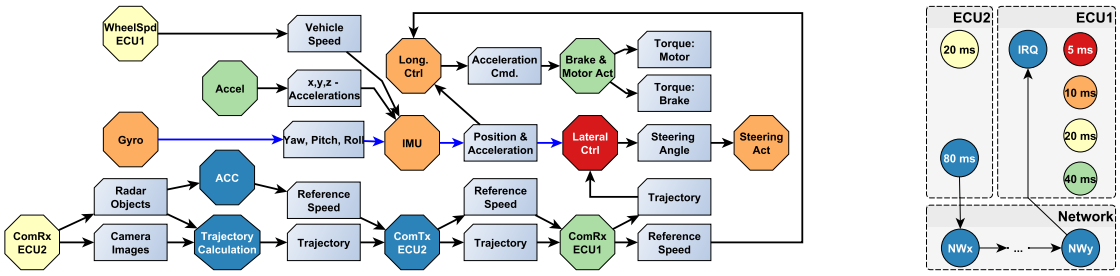
$$\vartheta_p : \mathcal{T} \times \{C^+, C^-, \delta_{in}^+, \delta_{in}^-\} \mapsto \mathcal{V} \tag{1}$$

that maps each input parameter type $p \in P$ for a task $\tau_i \in \mathcal{T}$ to a node $v = \vartheta_p(\tau_i, p)$ with $v \in \mathcal{V}$ in the TDG, and the *result conversion function*:

$$\vartheta_r : \mathcal{T} \times \{w^+, w^-, \delta_{out}^+, \delta_{out}^-, R^+, R^-, q_{max}\} \mapsto \mathcal{V} \tag{2}$$

that maps each result type $r \in R$ of a task $\tau_i \in \mathcal{T}$ to a node $v = \vartheta_r(\tau_i, r)$ with $v \in \mathcal{V}$ in the TDG.

This conversion function is analysis specific, i.e. how the busy-window and output event models are computed. In general, a TDG is constructed in four steps: First, for each task in the task graph, the timing dependency graph is populated with the nodes describing its parameters. In the second step, all explicit dependencies between tasks on different resources are added as edges in the graph. This happens for two tasks τ_a and τ_b by inserting two edges $e_k = (v_i, v_j)$ and $e_l = (v_m, v_n)$ into the dependency graph in order to capture the dependency between their output and input event model ($\delta_{a,out}^-/\delta_{a,out}^+$ and $\delta_{b,in}^-/\delta_{b,in}^+$). More precisely, $v_i = \vartheta_r(\tau_a, \delta_{out}^-)$ and $v_j = \vartheta_p(\tau_b, \delta_{in}^-)$ as well as $v_m = \vartheta_r(\tau_a, \delta_{out}^+)$ and $v_n = \vartheta_p(\tau_b, \delta_{in}^+)$. The third step then deals with the dependencies on each resource. It adds dependency edges according to the construction of the busy window (w^+/w^-), and the computation of response times (R^+/R^-). This implies that, for each scheduler, a specific transformation is necessary. Consequently, the third step must be carried out for each resource individually, respecting its scheduling analysis. It can further incorporate the treatment of dependent tasks on one resource, if not the task precedence constraints are treated according to step two. The fourth step then deals with capturing the dependencies that influence the computation of the output event model, based on the resource analysis results and the applied propagation strategy to bound them.



(a) The executable function model

(b) The task graph

Figure 3: Architecture of the example system. Colors of runnables (octagons) indicate the target task (circle) with the triggering period as task caption. Cycle times are based on [14].

We illustrate step two and three of constructing the TDG for the tasks on ECU2 from the example. The set of nodes $\mathcal{V}$ is populated based on $\mathcal{T}=\{\tau_{80ms}, \tau_{20ms}\}$ as well as two nodes v_i, v_j for $\delta^-_{NWx,in}$ and $\delta^+_{NWx,in}$ since τ_{NWx} is a dependent task of τ_{80ms}, and to illustrate step two. In the second step, the only two inserted edges are $e_k=(\vartheta_r(\tau_{80ms}, \delta^-_{out}), v_i)$ and $e_l=(\vartheta_r(\tau_{80ms}, \delta^+_{out}), v_j)$ since τ_{NWx} is the only dependent task in this scope. The third step, is dictated by the SPP scheduling and the rate-monotonic priority assignment, i.e. τ_{20ms} has a higher priority than τ_{80ms}. As in SPP w.o. blocking a higher-priority task's w^+ is independent from lower-priority tasks, w^+_{20ms} only depends on its own parameters C^+_{20ms} and $\delta^-_{20ms,in}$. However, w^+_{80ms} also depends on the maximum preemption time seen from τ_{20ms}, i.e. C^+_{20ms} and $\delta^-_{20ms,in}$, plus its own parameters $\delta^+_{80ms,in}$, C^+_{80ms} to maximize w^+_{80ms} to a conservative upper bound. The w^- for both tasks only depends on their own parameters C^- and δ^- as it lower bounds the time for a number of executions. WCRTs R^+_i in SPP are again directly dependent on w^+_i and $\delta^-_{i,in}$.

For proofs and in-depth descriptions on how the busy-window and BCRT/WCRT respectively can be bounded for a number of scheduling strategies we refer the reader to [17] for SPP including chained tasks, and [3] for Static-Priority Non-Preemptive (SPNP). Note that our approach is generally applicable to any analysis relying on the busy-window technique, e.g. also [20] who analyzes Switched-Ethernet traffic or [5] who shows analysis for data-age and reaction-time constraints.

The TDG captures the timing dependencies between all parameters involved in the system analysis. However, the goal is to identify the parameters (and results) that must hold in order for a bound to be valid.

Since the edges E of the TDG encode which parameters exert an influence on which parameter/result, the task at hand is to obtain all nodes from which the investigated node is reachable.

Definition 7. The *reachability graph* $\mathcal{G}^{reach}(v_t)=(\mathcal{V}^{reach}, \mathcal{E}^{reach})$ for a given node v_t is a graph with $\mathcal{V}^{reach}$ being the set of all nodes discovered by a breadth-first search from v_t in $\mathcal{G}$ with reversed edges and $\mathcal{E}^{reach}$ being the set of all traversed edges in original orientation.

Note that we refer to v_t as the root, although $\mathcal{G}^{reach}$ contains edges in the same orientation as the TDG, further the nodes which have an in-degree of 0 are referred to as leaves.

5 TRACING DEPENDENCIES

In this section we show how TDGs support a safety oriented design process, especially its implementation and integration phase and the technical safety concept [8, Part 4]. In the first place, TDGs allow to verify that timing behavior of elements with different safety levels is free from interference in the strict sense, i.e. that faults of one element can not lead to timing faults of other elements.

5.1 Strict Non-Interference

Definition 8. *Strict timing non-interference* is the inability of an element A to influence a timing requirement of element B.

We illustrate this on the initial example. Since lateral control exhibits higher dynamics than longitudinal control, the former has to be considered more critical as it has more stringent requirements w.r.t. controllability in the ISO 26262 sense. Consequently, it is attributed a higher Automotive Safety Integrity Level (ASIL) than the longitudinal control and freedom from interference must be established between the two. As already demonstrated in Figure 2 the data-age/dead-time of the cause-effect chain Ψ_1 (indicated in blue in Figure 3a) from Gyro via the IMU to the lateral controller (transporting the yaw angle) is particularly critical and inherits the lateral control's ASIL. For the mapping of $\mathcal{RG}$ to $\mathcal{TG}$ of the example we supply further knowledge about the implementation, not directly shown in Figure 3: (a) All runnables that are mapped to the same task are ordered such that writing and reading common labels happens in the same job of the task, (b) other read and write events of runnables occur at activation and termination events of the task they are mapped to and that label read and write times are accounted for in the tasks' execution time. However, any other mapping of events in $\mathcal{RG}$'s $\mathcal{S}$ to job instances of $\mathcal{TG}$ are possible, depending on the implementation. Ψ_1's end-to-end data-age requirement is computed as the sum of the data-ages in the chain, i.e. $d^{age}_{\Psi_1}=d^{age}_{Gyro,IMU} + d^{age}_{IMU,Lat}$. Via the mapping from $\mathcal{RG}$ to $\mathcal{TG}$ this translates into $R^+_{10ms} + d^{age}_{10ms,5ms}$, accounting for the time from activation of ρ_{Gyro} to the read event of ρ_{Lat}.

In the example, we assume that the Network is shared among a number of streams, which exhibit a lower ASIL (or none at all) than required for $d^{age}_{10ms,5ms}$. In consequence, these frames can influence the timing behavior of ECU1, as the reception of any Ethernet frame triggers an IRQ on the receiving ECU. This is indicated by the task precedence shown in Figure 3b between τ_{NWy} and τ_{IRQ}.

Figure 4 shows the resulting reachability graph $\mathcal{G}^{reach}(d^{age}_{10ms,5ms})$ of $d^{age}_{\Psi_1}$. What the reachability graph shows is that the timing behavior of event streams on the network can influence the IRQ-load of ECU1, and is thus able to taint the requirement of $\mathcal{G}^{reach}(d^{age}_{\Psi_1})$'s root. The interference from unqualified streams is highlighted in orange coloring, which shall indicate the lowest assurance level, while white coloring indicates a similar assurance level as $d^{age}_{10ms,5ms}$. Consequently, the example system would not fulfill the strict non-interference requirement.

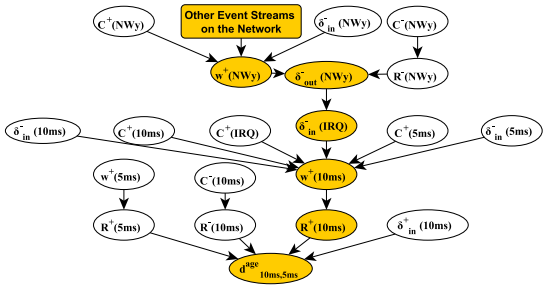


Figure 4: Reachability Graph $\mathcal{G}^{reach}(d^{age}_{10ms,5ms})$ for the data-age requirement of the yaw rate

To understand the general importance of automatic timing dependency analysis for system design we investigate system feasibility w.r.t. strict timing non-interference. A system is feasible under the strict non-interference constraint iff the reachability graph of the TDG of any constraint attributed with a ASIL does not contain any dependency on any element with a lower ASIL. To investigate feasibility under this constraint, we generate test systems that resemble typical automotive cause-effect chains. I.e. each cause-effect chain contains a number of runnables with varying triggering periods and implicit, unbuffered label communication. Without loss of generality we assume that each runnable is mapped into an individual task, and that each cause-effect chain contains at least one label between communicating tasks. Criticalities, i.e. ASILs between QM and D, are randomly (uniformly) assigned to each of the cause-effect chains, and the chains are generated with the SMFF/TGFF benchmark [12]. For each chain, the end-to-end data age is set as the safety-critical timing constraint. It is composed of the data age between the communicating tasks and the response times of tasks between labels.

Note that w.r.t. systems design and the technical safety concept all cause-effect chains are dependent at this stage, since information on the platform is missing to proof independence. For the experiment, the platforms are also generated with SMFF and the cause-effect chains are mapped in a sensor-to-actuator style, where no chain traverses a resource more than once, although consecutive tasks in the data flow of the cause-effect chain can reside on one resource (for over and undersampling). W.r.t. their timing behavior the chains on the generated platform are still dependent, as the necessary knowledge to compute the TDGs is still insufficient and thus no proof of independence is possible. In oder to construct the TDGs information on the scheduling strategy and parameters must be available. Automotive ECUs are typically SPP scheduled, consequently we assume this for the generated systems as well [2]. Subsequently, we conduct the experiment in two variants: In the first one, we assign priorities to individual tasks randomly, while in the second one priorities on each resource are assigned in RM order, i.e. according to the triggering period of the task. To determine feasibility, we compute the reachability graph of the WCRT and data-age requirements for all cause-effect chains and check whether a parameter node $v \in \mathcal{V}$ (e.g. the WCET) of a task from any lower-critical cause-effect chain is contained. I.e. a cause-effect chain is marked infeasible if one of its tasks experiences interference according to 8.

We generated 10000 test systems, each containing four resources with 40 cause-effect chains consisting of eight tasks each. The attraction factor for the SMFF mapper is set such that on average 2 tasks per chain reside on the same resource.

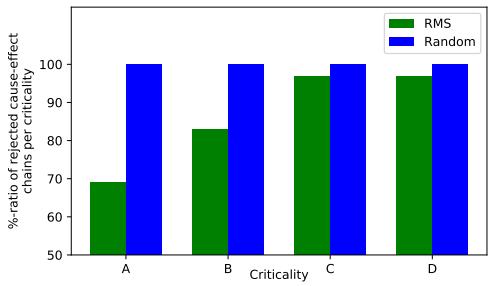


Figure 5: %-ratio of infeasible effect chains per criticality under the strict non-interference definition

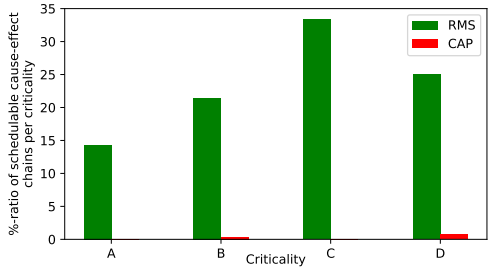


Figure 6: %-ratio of schedulable cause-effect chains per criticality

Figure 5 shows the %-ratios of infeasible chains (w.r.t. strict timing non-interference) over all chains per criticality. For an entirely random mapping almost all effect chains (above 99%) from a certain criticality experience interference from a lower criticality and is thus infeasible. This can be expected due to the random mapping and priority assignment. For the typically applied RM priority scheme (due to its optimal resource usage), still most of the chains experience interference from a lower criticality. Compared to the random priority assignment chains with oversampling turn feasible, since RM assignment causes higher task priorities. However, the result is unsatisfactory if chains with heterogeneous criticalities should be mapped to a common platform.

The unfeasible systems can only be made feasible if a design parameter is changed, e.g. the mapping or priority assignment. If we assume the mapping is fixed due to sensor and actuator dependencies, the priorities have to be assigned such that they isolate

timing of different criticalities. A Criticality as Priority (CAP) priority order on each resource turns all chains feasible, however, has implications on the timing performance and can affect schedulability compared to the optimal RM assignment. To investigate this, we assign WCET and BCET with the UUnifast algorithm (utilizations of 0.6 to 0.9 per resource, BCET probability of 0.5), and assume the period of a task to be its deadline. Figure 6 shows the deteriorated schedulability of the CAP compared to an RM assignment. Note that a chain is considered unschedulable if at least one task of the chain violates its deadline.

5.2 Bounded Interference

However, safety and performance are not necessarily antagonists. In cases where a schedulable, performance-efficient priority assignment (e.g. RM) can be found, but strict interference exists, timing-dependence analysis can be used as a design method to avoid infeasible systems. Therefore, we identify timing parameters that interfere with a requirement of a certain criticality via the TDG. The technical safety concept then only has to ensure that the interference from these parameters is bounded such that it can not violate a timing requirement. This only lifts the respective parameter's safety requirement, not the ASIL of its function or cause-effect chain. It relaxes the strict notion of non-interference from 8 to one of *bounded interference*. For bounded interference the TDG then systematically identifies timing parameters that must be assured to the criticality level of the strictly interfered requirement. I.e. although a timing parameter belongs to a lower-critical effect chain, the respective parameter must be qualified to the required level, e.g. through suitable documentation or analysis techniques. For instance, tight WCET parameters for low or uncritical functions are often not documented and specified to the same extent as functions with higher ASIL requirements. The TDG, however, can require this specifically for one parameter without propagating the whole safety requirement to the interfering function. After assuring identified requirements, the system is feasible under the bounded interference requirement.

5.3 Scalability

The runtime of our TDG construction is dependent on the number of runnables and tasks (assuming: #cores << #tasks + #runnables). Measurements[2] for systems with 4 up to 20 cores with a task structure similar to [18], and a random mapping of tasks to resources support this: The median time per run (out of 100 runs) is between 6s and 7s. This is identical to the runtime for TDG and reachability graph construction in the model of a publicly available real-world engine-management system with 4 cores and over 10000 runnables and labels each [18].

6 CONCLUSION

For a comprehensive view on a system's behavior to rule out interference ISO 26262 proposes methods like FTA and FMEA. Our proposed TDGs support a system-wide FMEA by proving timing non-interference proofs in an automated way. Furthermore, we have shown how to use TDGs as a design automation method for cases where strict timing non-interference can not be proved. Our Non-Interference Analysis increases design efficiency since it limits the separation of elements with different ASILs to relevant dependencies, and a proof that this suffices can be provided. This is well suited to be combined with run-time enforcement techniques such as [7, 10]. However, limiting their non-negligible overhead to a minimum since the technique is only applied where necessary.

[2] conducted single threaded on an Intel i5-3210M@2.5GHz

REFERENCES

[1] Karl-Erik Årzén, Anton Cervin, and Dan Henriksson. 2005. Implementation-Aware Embedded Control Systems. In *Handbook of Networked and Embedded Control Systems*. Birkhäuser.

[2] AUTOSAR 2016. *Specification of Operating System* (4.3 ed.). AUTOSAR.

[3] Iain John Bate. 1999. *Scheduling and timing analysis for safety critical real-time systems.* Ph.D. Dissertation. University of York.

[4] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. 2007. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems* 39, 1-3 (April 2007), 5–30.

[5] Kai-Björn Gemlau, Johannes Schlatow, Mischa Möstl, and Rolf Ernst. 2017. Compositional Analysis for the WATERS Industrial Challenge 2017. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. Dubrovnik, Croatia.

[6] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. 2005. System Level Performance Analysis - the SymTA/S Approach. In *IEE Proceedings Computers and Digital Techniques*.

[7] Kai Huang, Gang Chen, C. Buckl, and A. Knoll. 2012. Conforming the runtime inputs for hard real-time embedded systems. In *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 430 –436.

[8] Intern. Organization for Standardization - ISO 2011. *ISO 26262 - Road vehicles - Functional safety* (2 ed.). Intern. Organization for Standardization - ISO.

[9] Hermann Kopetz. 2011. *Real-Time Systems: Design Principles for Distributed Embedded Applications* (2 ed.). Springer.

[10] Moritz Neukirchner, Sophie Quinton, Rolf Ernst, and Kai Lampka. 2013. Multi-mode monitoring for mixed-criticality real-time systems. IEEE, 1–10.

[11] Moritz Neukirchner, Sophie Quinton, Tobias Michaels, Philip Axer, and Rolf Ernst. 2013. Sensitivity Analysis for Arbitrary Activation Patterns in Real-time Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '13)*. EDA Consortium, 135–140.

[12] Moritz Neukirchner, Steffen Stein, and Rolf Ernst. 2011. Smff: System models for free. In *2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2011)*. 6.

[13] Silviu-Iulian Niculescu. 2001. *Delay effects on stability: a robust control approach.* Vol. 269. Springer Science & Business Media.

[14] Marcus Nolte, Marcel Rose, Torben Stolte, and Markus Maurer. 2017. Model Predictive Control Based Trajectory Generation for Autonomous Vehicles – An Architectural Approach. Los Angeles, USA.

[15] R. Racu, R. Ernst, and A. Hamann. 2006. A formal approach to robustness maximization of complex heterogeneous embedded systems. In *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '06)*. 40–45. https://doi.org/10.1145/1176254.1176267

[16] A. Sangiovanni-Vincentelli. 2012. Taming Dr. Frankenstein: A primer on the challenges posed by cyber-physical systems. In *2012 IEEE Technology Time Machine Symposium (TTM)*. 1–1. https://doi.org/10.1109/TTM.2012.6509037

[17] Johannes Schlatow and Rolf Ernst. 2016. Response-Time Analysis for Task Chains in Communicating Threads. In *Real-Time Embedded Technology & Applications Symposium (RTAS)*. Vienna, Austria.

[18] Arne Hamann Simon Kramer, Dirk Ziegenbein. 2016. Real World Automotive Benchmark For Free. In *Proceedings of the 6th International Workshop on Analysis Tools and Methodologies for Embedded Real-Time Systems WATERS 2016.*

[19] T. Stolte, G. Bagschik, and M. Maurer. 2016. Safety goals and functional safety requirements for actuation systems of automated vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2191–2198.

[20] Daniel Thiele, Philip Axer, and Rolf Ernst. 2015. Improving Formal Timing Analysis of Switched Ethernet by Exploiting FIFO Scheduling. In *52nd Annual Design Automation Conference (DAC '15)*. ACM, New York, NY, USA, 41:1–41:6.

[21] L. Thiele, S. Chakraborty, and M. Naedele. 2000. Real-time calculus for scheduling hard real-time systems. In *The 2000 IEEE International Symposium on Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva*, Vol. 4. 101–104 vol.4.

[22] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. 2006. System architecture evaluation using modular performance analysis: a case study. *Intern. Journal on Software Tools for Technology Transfer* 8, 6 (July 2006), 649–667.