Early Safety Evaluation of Design Decisions in E/E Architecture according to ISO 26262

Vladimir Rupanov, Alois Knoll
Technische Universität München
Boltzmannstr. 3
Garching, Germany
{rupanov,knoll}@in.tum.de

Ludger Fiege, Michael Armbruster, Gernot Spiegelberg
Corporate Research & Technologies
Siemens AG
Otto-Hahn-Ring 6
Munich, Germany
{firstname.name}@siemens.com

Christian Buckl
ForTISS GmbH
Guerickestr. 25
Munich, Germany
buckl@fortiss.org

ABSTRACT

ISO 26262 addresses development of safe in-vehicle functions by specifying methods potentially used in the design and development lifecycle. It does not indicate what is sufficient and leaves room for interpretation. However, the architects of electric/electronic systems need design boundaries to make decisions during architecture evolution without adding a risk of late architectural changes. Designing and changing a system benefits from correct selection of safety mechanisms at early design stages. This paper presents an iterative architecture design and refinement process that is centered around ISO 26262 requirements. We propose a domain-specific modeling scheme and component repositories to build up a bottom-up analysis framework that allows early quantitative safety evaluation. To guarantee that the target ASIL level can be reached, we complement our design-time component-level analysis with conservative top-down analysis. Given that analysis starts at early design stages, evolution of the architecture is supported by different levels of detail used in the analysis framework.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms

Reliability

Keywords

Automotive Systems, Architecture Modeling, Functional Safety, Integration of Analysis Techniques

ISARCS'12, June 26–28, 2012, Bertinoro, Italy.

1. INTRODUCTION

In today's cars, most of the functionality is implemented using hardware and software solutions. As more and more safety-critical functions heavily rely on software, safety becomes a hot topic. The recently published international standard ISO 26262 [14, part 1] adresses this topic by defining a design process and proposing safety mechanisms. It does not indicate what is sufficient and leaves room for interpretation [8]. The architects of in-vehicle systems need, however, design boundaries to make decisions during architecture evolution without adding a risk of late architecture changes.

To reduce the complexity of the electric/electronic (E/E) architectures, the car manufacturers intend to use generic hardware and software platforms executing mixed-criticality functions, such as AUTOSAR[1]. This enables software reuse, but limits the system-level analysis for early assessment of functional and non-functional properties [10]. An E/E system architect has to account on numerous design aspects at the same time: real-time properties, safety, security, cost, etc. As late architectural changes are very expensive, it is extremely important to support early architectural decisions. An important part of these decisions covered by ISO 26262 is selecting and configuring safety measures.

In this paper, we propose an approach for safety analysis and design guidance early in the design process. Although safety analysis has been applied in the automotive industry for decades, no common safety lifecycle was applied. ISO 26262 triggers a change in the development lifecycle that requires adaptation and alignment of numerous processes in E/E system development. The standardized lifecycle also acts as an enabling factor for extensive use of model-based tools for engineering support including automation of routine analysis steps, collection of data in a unified format and reuse of that data in new developments.

This paper suggests to systematically evaluate safety mechanisms in context of ISO 26262 requirements and assess design alternatives from different viewpoints (performance, cost). The approach starts with a specification of the fault behavior of hardware component models. We relate safety

[1]AUTOSAR: AUTomotive Open System ARchitecture, more details at: http://www.autosar.org

mechanisms to fault models and provide methods to evaluate achievable coverage. The presented design methodology supports evolution of E/E architecture (EEA) under a guarantee that the target ASIL (Automotive Safety Integrity Level) can be reached by stepwise refinement of both component and safety mechanism models controlled by evaluation of ISO 26262 architectural metrics. The necessary steps for implementing our approach are discussed including the definition of metamodels and quantitative metrics and an automotive case study presented.

The rest of the paper is organized as follows. Section 2 provides the reader with an overview of the context and the range of applications that are motivating our work. In Section 3 we present an iterative process of safety-centric architecture development. We discuss modeling abstractions, which support encapsulation of important attributes within reusable model hierarchies, in Section 4. Associated standard-defined metrics are linked to models in Section 5. We discuss application possibilities for the presented approach in Section 6. The paper is concluded by a review of related work in Section 7 and a summary in Section 8.

2. SAFE ICT PLATFORM FOR MASS PRODUCTION VEHICLES

During the last 30 years, electronic systems have become widely used in cars, resulting in numerous advances in driving safety, comfort and controllability of vehicles. The use of computers in vehicle applications raises the question of adequate behavior of automotive systems, especially of those controlling or related to vital functions, such as braking, steering and longitudinal speed control. E/E systems already play a key role in implementation of assistance functions like electronic stability program (ESP) or electronic brakeforce distribution (EBD), and the likely introduction of Drive-by-Wire systems will lead to total reliance of driver safety on E/E systems [13].

Safety is a dependability attribute of a system or an architecture, which reflects "absence of catastrophic consequences on the user(s) and the environment" [2]. Absolute safety is hardly reachable, so safety-related standards introduce a notion of "unacceptable risk" [14, part 1] into the safety definition. The ASIL assignment to a function specifies which level of risk is acceptable. The safety of a system is directly influenced by the EEA, which describes the subsystems, their boundaries and interfaces, and includes the allocation of functions to hardware and software elements. In context of systems with integrated architecture, EEA represents the functional structure of the system (functional concept) and physical structure of the system (hardware components) and mapping between them.

The state of the art design process in industry focuses on optimizing safety at the function level. This approach was feasible, as most functions could be analyzed in an isolated fashion. With the trend towards more interconnected functions, such as a global energy management in electric cars, the complete architecture must be analyzed according to its suitability to achieve safety. The trend towards system safety is also increased by shortened development cycles. As functions have to be integrated into the car in shorter time intervals, E/E platforms providing generic mechanisms to reach safety goals are becoming more important [5]. The name "platform" comprises the following item of a full vehicle control-system: *a*) a central (fault-tolerant) platform-computer with access to all sensors and actuators via network; *b*) an operating system extended by a middleware running mainly on the central platform-computer which provides means to execute and transparently protect vehicle system functions and provides mode management functionality (including platform-reconfiguration due to faults, master-slave switchovers to ensure fail-operational behavior); *c*) a communication and power-distribution network. When talking about generic mechanisms, one has to distinguish between two categories of functions. For some functions it is possible to define a safe state. These functions can be designed in a way that any failure in a system leads to a safe state (which means passivation of the considered function), so they have "fail-safe", or "fail-passive" requirements. In the second category, functions do not have a defined safe state: these functions need to operate in order to maintain driver and occupant safety. "Fail-operational" behavior is necessary to meet quantitative safety requirements for these functions [13].

An important task for an architecture designer is to provide a reasonable architecture sketch and its evolution into a safe and stable system, without limiting the software capabilities and undergoing deep changes. Depending on the selected safety mechanisms (SMs), the safety goals for a specific instantiation of the E/E platforms may be violated. Safety mechanisms, as defined by [14, part 1], are measures implemented by an E/E system function or element to detect or control failures. The faults in hardware components result from aging / wear-out, aggressive environment and manufacturing process variations.

Although, ISO 26262 defines concrete safety mechanisms, literature reports different problems [8], mainly regarding the correct selection of the right level of detail and traceability between FMEA and safety assessment. This leads to the problem, that sufficiency and adequacy of safety mechanisms are hard to predict for a concrete EEA design. We present a solution for the last problem by suggesting an approach for early safety evaluation to simplify the correct selection of safety mechanisms and avoiding changes late in the design process. It is important to note that this paper focuses on random faults in hardware and does not target the toleration of systematic faults in hardware and software. The latter problem is tackled by the suggested development process of the ISO 26262.

There is a huge range of implementations of SMs available on the market in the form of either hardware component features or middleware, but there are no tools to quantify the effect of selected mechanisms in advance and to compare these against objectives (e.g., target values for ISO 26262 architectural metrics). Our approach aims at methods and tools that support system architects in evaluating design choices. The methods that engineers apply are usually dealing with consequences of faults; it is usually not possible to identify the source. Failure mode analyses in inductive (e.g., FMEA) or deductive (e.g., Fault Tree Analysis, FTA) methods rely on partially or fully automated [18] use of detailed failure modes of components and analysis of safety measures that are implemented in the system to reduce or eliminate the risk. We propose a focused approach to model design trade-offs and limit effort spent for safety estimation.

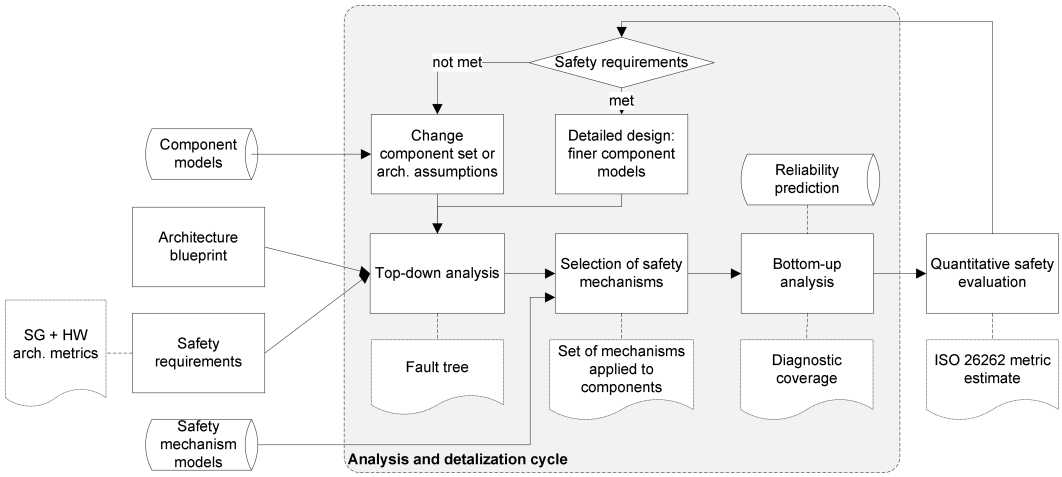


Figure 1: Architecture design cycle: phases and artifacts

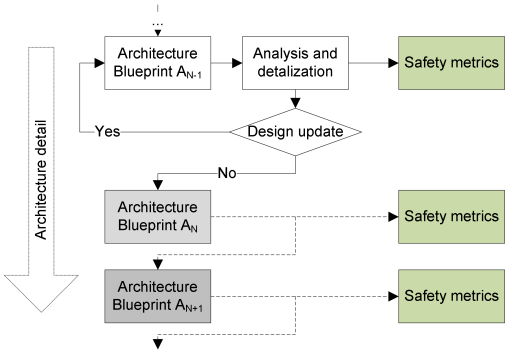


Figure 2: Evolutionary staged development process

3. ITERATIVE DESIGN PROCESS

This section in combination with the following two describes our approach for design-time quantitative safety analysis. It is based on quantitative evaluation of safety-related metrics in each design cycle explained in this section. Structured models, which are described in the next section, enable the automation of safety assessment (section 5). The approach can be applied iteratively, which means that design boundaries are evaluated at each EEA evolution cycle, and design decisions are driven by results of this evaluation.

We want to provide guidelines for system architect in selecting hardware components and hardware and/or software implemented safety mechanisms to be realized in the EEA that targets a certain ASIL level. This is reached by identifying boundaries of EEA design guaranteeing that each solution within these boundaries satisfies the safety requirements. Safety is only one design criterion (although, a primary architectural driver for critical systems); cost, performance, and other quality attributes are also important, and comparison of design alternatives with regard to other attributes needs to be supported. To achieve the defined goals in the design environment of an evolving EEA, we need: *a*) support for EEA evolution: methods and models enable different levels of detail; *b*) methods for early conservative quantitative evaluation of item safety; *c*) estimation of quality attributes (resources, cost); *d*) selection of safety mechanisms from possible alternatives; *e*) proof of design for safety according to the ISO 26262 standard; *f*) means of safety mechanism instantiation. The EEA design is a process of cyclic refinement in a domain-specific safety prediction framework (figure 1). Cycles represent levels of detail of EEA design that are passed throughout the design process. Every cycle of the process begins with model update: **an architecture blueprint** is created or updated. An initial blueprint defines a set of computer nodes (Electronic Control Units - ECU's), functions (items) and preliminary assumptions on components of each node. Typical artifacts of the architecture draft are: structural block diagrams, functional network diagrams, function deployment. At the next step, **identification of safety requirements** takes place. Requirements include safety goal (SG) definitions, ISO 26262 hardware architectural metric and probability of safety goal violation target values. Metric target values are directly derived from hazard and risk analysis of the items to be deployed on the EEA (i.e. from the *usage profile*). These requirements are typically stable throughout the whole EEA design process. **Deductive analysis** is initiated based on the preliminary hazard analysis and the identified SGs. A typical method applied at this stage is fault tree analysis[2]. The result of its application is a set of fault trees where leaves represent node-level failure effects of components (such effects are "value delivered late" and "wrong value"). The fault trees built at this stage are a *system-level safety evaluation model*. **Selection of safety mechanisms** is performed based on coverage requirements identified before (using *component- and mechanism- specific evaluation models*). In addition, *design quality attributes* such as memory footprint and performance are taken into account. We return to this in detail in next paragraphs. The methodology includes these attributes into analysis to avoid late architecture updates because of not implementable requirements to the execution platform. **Inductive analysis** is performed as a form of typical failure modes, effects and diagnostics analysis (FMEDA). We extract diagnostic cov-

[2]http://www.fault-tree.net

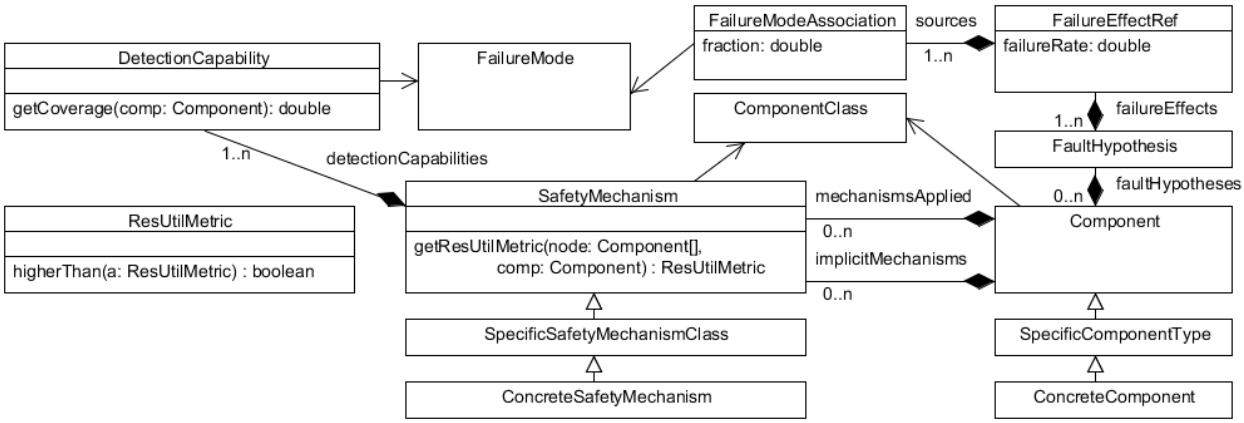


Figure 3: Essential part of the proposed metamodel

erage of specific mechanisms in application to components of the ECU, and use this information to calculate total coverage and intermediate failure rates for failure effects (i.e. apply *composition algorithms* at node level). **Combination of artifacts and quantification** consists of using FMEDA results as inputs in the fault tree and enables the evaluation of quantitative metrics according to ISO26262. This step results in a preliminary assessment of the item's expected safety. An **update of the architecture** may be vital if the evaluated metrics or design quality attributes did not match the target. If these have matched, the cycle is repeated with introduction of next level of detail (figure 2).

At the initial stage, only basic architectural assumptions are typically present: classes of components, structure of the system. Based on this information it is possible to define the basic structure of the system, identify functional safety requirements and to validate plausibility of the structure. At further stages, technical safety requirements arise as component descriptions are specified. This can be seen as a "stacking model" of evolution: at each next level the volume of information increases, more specific component and mechanism models are applied in the architecture blueprint, so the design space is reduced. This approach enables the development of multiple product lines. Updating a function set that is supported by the EEA results in new requirements. If an architecture has been developed through a series of blueprints, the approach allows to perform the evaluation of blueprints in reversed sequence starting with the most detailed one. This approach results in a fast selection of a valid baseline which has sufficient design space to satisfy the updated requirements.

To limit the scope, it is useful to split the design into smaller parts, providing certain budgets to each part. This approach is productive from the safety point of view, as conservative estimations hold even after the full design is evaluated. From the cost side it is more risky, as cost reduction is not efficient without re-estimation of the overall budget. A reasonable balance needs to be found, for example, by reassigning the budget based on slack of the parts.

The set of hazardous events that lead to failure effects in hardware is not stable with time. For example, integrated circuits become smaller and more sensitive to single event upsets (SEU), electromagnetic interference (EMI) and other environmental disturbances. Changes in the architecture often lead to re-estimation of the failure modes and effects. Detection mechanisms are to be adjusted to these changes as a result. Both the set of mechanisms and their parametrization over components might change over time. Therefore, we propose a **repository-based approach** to store information on both components and detection mechanisms. This allows us to systematically collect a component hierarchy (from basic components down to concrete devices) and corresponding detection mechanisms in a consistent manner and enables reuse of data that has once been entered into the system. We propose to use two repositories: one for component models and one for models of safety mechanisms. Compatibility of a safety mechanism and a component (application pattern of the safety mechanism) is unambiguously defined through a composition of component class CC and covered failure modes $\{FM_i\}$.

Methods for evaluation of safety-relevant and other design quality attributes are encapsulated into safety mechanism models. This enables building modeling tools for evaluation of safety metrics on system level and resource utilization on node level. Definition of a consistent set of metrics is an important part of automation of the process implementation. The use of model-based approach also makes automated instantiation of the mechanisms possible through generation of code from the final model of EEA.

4. STRUCTURED DESCRIPTION OF SAFETY MECHANISMS

As mentioned above, an adequate approach to modeling safety mechanisms and components allows partial automation of analysis activities while integrated in a straightforward process. In this section we present a suitable metamodel for representing the essential part of design artifacts. At the top level, EEA is represented by a set of computing nodes, where functional networks responsible for execution of each item, and network or bus connections between those. Network-related components can be treated as a separate node [20], and are not in focus of our attention. It is possible to describe the top-down propagation structure

of such a system using existing generic analysis frameworks. In this paper we pick component fault trees, but other top-down safety-oriented frameworks are plausible. In their basic version, component fault trees (CFT) are layered fault propagation graphs just like normal fault trees without the requirement for the graph to be a tree. We use these to describe top-level undesired events, which is the violation of safety goals, break these hierarchically down to node level and further to failure effects of certain components in certain node. Mathematically, each CFT represents a logical function from its input ports and internal events to its output ports [16]. We use this modeling approach without any significant changes.
To analyze the node-level fault behaviors and achieve flexibility in modeling safety mechanisms, the following requirements are to be considered:

1. Models of safety mechanisms have to be kept separate from component models, as a single safety mechanism can cover numerous failure modes of different components causing the same failure effect on the node level.
2. Flexible description of safety mechanisms should allow specialization with regard to particular attributes (such as specific algorithm, signature or array size) at later design cycles.
3. Semantic correctness (applicability of specific mechanism to certain component) has to be resolved through component class hierarchy.
4. Models need to provide sufficient information for bottom-up quantitative analysis up to node-level failure effects set, which acts as an interface layer to top-down analysis.
5. Architecture evolution must be enabled by hierarchic approach to component modeling.

To satisfy these requirements, the metamodel shown on figure 3 is proposed. The essential part of the metamodel is compact and can potentially be integrated with any system modeling framework (more details follow in Section 7). We explain the relations between entities in detail below.
A node N in the EEA is modeled by a list of components and a list of safety mechanism instances:$\{\{C_j\}, \{SM_k\}\}$. This allows analysis of all the component-failure effects on node level. A `Component` C_i is identified as a source of a set of `FailureEffects` $\{FE_{C_j,k}\}$ that can lead to violation of safety goal on the top level. These failure effects are included into `Component`'s `faultHypothesis`. A `FailureEffect` FE can be caused by a number of different `FailureModes`: $\{FM_i\}$: `FaultHypothesis` associates them with `fraction` $K_{FMi,Cj}$ as percentage of failure rate for an effect caused by specific failure mode.[3][4]

[3]For electronic and mechanical components, distribution of failure modes can be found in datasheets or special literature (e.g. [4], [21])
[4]Failure modes decomposition to the lowest level (traceable to physical failure mechanisms) is not required. Extra detail could lead to increasing design space, and conservative estimations do not change significantly. The traditional classification is often enough: value("wrong"), timing("early"/"late"), "omission"/"commission". It might be even reasonable to assign a single "arbitrary" failure effect to a single "arbitrary" failure mode if detailed failure behavior is not defined.

Hierarchy of components is supported by inheritance from `Component` type. A basic component (for example, variable memory) has only limited notion of failure modes and effects that it can cause on the node level. After we continuously refine the design, the specialized component has a refined (concretized) `faultHypothesis` and some implicit safety mechanisms that are embedded in the specific chip.
A `SafetyMechanism` is modeled as possessing one or more detection capabilities. An object of class `DetectionCapability` characterizes mechanism's coverage DC of specific failure modes (through the use of `getCoverage()` method) of specific component classes CC (association is managed through hierarchy of `ComponentClass` enumeration values). In simple cases capabilities can be represented in a tabular form: $\{\{CC_i,\ FM_i,\ DC_i\}\}$. When a safety mechanism SM is instantiated in an EEA model, it is applied to one or more specific components by adding `mechanismsApplied` reference list. Some components have also predefined mechanisms that are implicit (e.g., embedded error detection and correction logic), these are modeled by `implicitMechanisms` reference list. Additionally, a set of functions to evaluate resource utilization is encapsulated into a safety mechanism `getResUtilMetric(N,C)`, which can be used to compare design decisions. `ComponentClass` value hierarchy and `ResUtilMetric` are arbitrary: the only requirement is the possibility to compare two typically values of a metric (it is not typically a scalar value).
Inheritance of safety mechanisms also supports different levels of detail description at different EEA evolution stages: the same mechanism can pass stages from basic description (e.g., a "march" memory test), which can only derive certain (theoretic) limits on coverage of component failure modes, to a highly specific implementation (e.g. MATS++, a "march" test with high coverage), which is fully parametrized and can evaluate exact statistical coverage of certain failure modes. Another typical situation is that some mechanisms are initially not introduced in the system, and then after selection of specific component it appear implicitly. An example is an SRAM chip with built-in EDC logic. An alternative to inheritance-based modeling might be the use of feature models [3] to model component and safety mechanism variety, e.g. representing safety mechanisms with a multiple-level "or"-tree, every next level of which increases the detail.
Our method also allows accounting on design quality attributes, such as performance and memory footprint. To achieve this, each safety mechanism has an associated function that provides representative metric. This metric can be a single value or a vector (if multi-criteria optimization is performed). Implementation of EEA evolution process with these modeling capabilities requires adequate implementation with tools and integration with other quality attribute-specific development processes.

5. QUANTITATIVE ANALYSIS

We describe in this section the approach to quantitative safety analysis that can be automated and makes use of the models defined in the previous section. The state-of-the-art requirements to EEA safety are defined by ISO 26262-5 [14, part 5]. To simplify the diagnostic coverage assessment process, all the random hardware faults in ISO 26262 are classified into *single point*, *multiple point*, *safe* faults, and a set of architectural metrics is defined based on these classes. Most important categories are single-point faults, residual

ASIL	Single point faults metric, $SPFM$	Latent faults metric, LFM	Probability of violation of safety goal P_{VSG}
B	$\geq 90\%$	$\geq 60\%$	$< 10^{-7}h^{-1}$
C	$\geq 97\%$	$\geq 80\%$	$< 10^{-7}h^{-1}$
D	$\geq 99\%$	$\geq 90\%$	$< 10^{-8}h^{-1}$

Table 1: ASIL-specific target values for architectural and probabilistic metrics [14, part 5]

faults (not covered by any mechanism) and latent multiple point faults.[5] This classification is used throughout the standard and is used to compose a set of metrics, which characterize the achieved safety level. Both single point faults metric (SPFM) and latent faults metric (LFM) characterize the EEA coverage of dangerous (related to safety goals) events:

$$SPFM = 1 - (\frac{\sum(\lambda_{\text{SPF}} + \lambda_{\text{RF}})}{\sum \lambda} \quad (1)$$

$$LFM = 1 - \frac{\sum \lambda_{\text{MPF\_latent}}}{\sum(\lambda - \lambda_{\text{SPF}} - \lambda_{\text{RF}})} \quad (2)$$

λ_{SPF}, λ_{RF} and $\lambda_{MPFLatent}$ represent the failure rates of corresponding non-intersecting fault classes:

$$\lambda = \lambda_{\text{SPF}} + (\lambda_{\text{detected}} + \lambda_{\text{RF}}) + (\lambda_{\text{MPF\_detected}} + \lambda_{\text{MPF\_perceived}} + \lambda_{\text{MPF\_latent}}) \quad (3)$$

The higher the value of each of the architectural metrics, the more robust is the design. Target values for each metric are summarized in table 1.
Another set of requirements are probabilistic metrics of hardware. Each ASIL level is assigned a quantitative target value enforced by ISO 26262. It can be either based on existing similar systems (derived either by analysis or from field data), or derived from standard recommendations (table 1). All faults in the system are assigned failure rate classes depending on how associated failure rate compares with the target for specified ASIL level:

$$\lambda_{class\ i} < \frac{P_{VSGmax}}{10^{3-i}}$$

Based on failure rate classes, some ASIL-specific constraints are applied (for example, residual and single point faults are acceptable in an ASIL D system only if ranked as class 1).
We use the modeling schema proposed in the previous section, to predict quantitative safety metrics required by ISO 26262 and to analyze a set of safety mechanisms for sufficiency when implementing specific item. We suggest the following workflow:

- Based on the safety goal and its classification, the target values for the following quantitative requirements can be set[6]:

$$\begin{aligned} P_{VSG} &< P_{VSGmax} \\ SPFM &> SPFM_{min} \\ LFM &> LFM_{min} \end{aligned} \quad (4)$$

- **Top-down analysis** Based on preliminary hazard analysis, we can build a fault tree that lets us trace back the failure propagation to node-level failure effects. We can build such a tree without specific knowledge of the hardware used on exact node, just by specifying generic components (and, of course, generic failure effects). Evolution of the architecture will lead to a finer definition of the fault tree. Such a fault tree allows evaluation of probability of violation of safety goal based on node-level failure rates associated with failure effects which are leaf nodes of the fault tree. So, to perform full P_{VSG} calculation, we need the results of bottom-up analysis.
- **Bottom-up analysis** On each node we analyze the set of components for dangerous failure rates. Based on fault hypothesis of the component C and the knowledge of dangerous failure effects FE, coverage of the safety mechanism SM is evaluated ($DC_{SM,FM_k,C}$), and dangerous failure rates are calculated from the "fraction" K_{FM_k,C_j} (as the sum of all dedicated failure mode specific ones):

$$DC_{SM,FE,C} = K_{FM_1,C} \times DC_{SM,FM_1,C} + ... + K_{FM_n} \times DC_{SM,FM_n,C}$$

  Given that safety mechanisms are instantiated independently, we calculate residual failure rate:

$$\lambda_{RF,FE,C} = \lambda_{FE,C} \times (1 - \sum DC_{SM,FE_k,C})$$

  Iterating over all components, we calculate SPFM value (eq.1).

  Calculating with the same method safety mechanism coverage over non-dangerous failure modes, we evaluate $\lambda_{\text{MPF\_detected}}$ – "detected" multiple point failure rate. Assuming no multiple point failures are perceived, the conservative evaluation of eq. 2 looks like this:

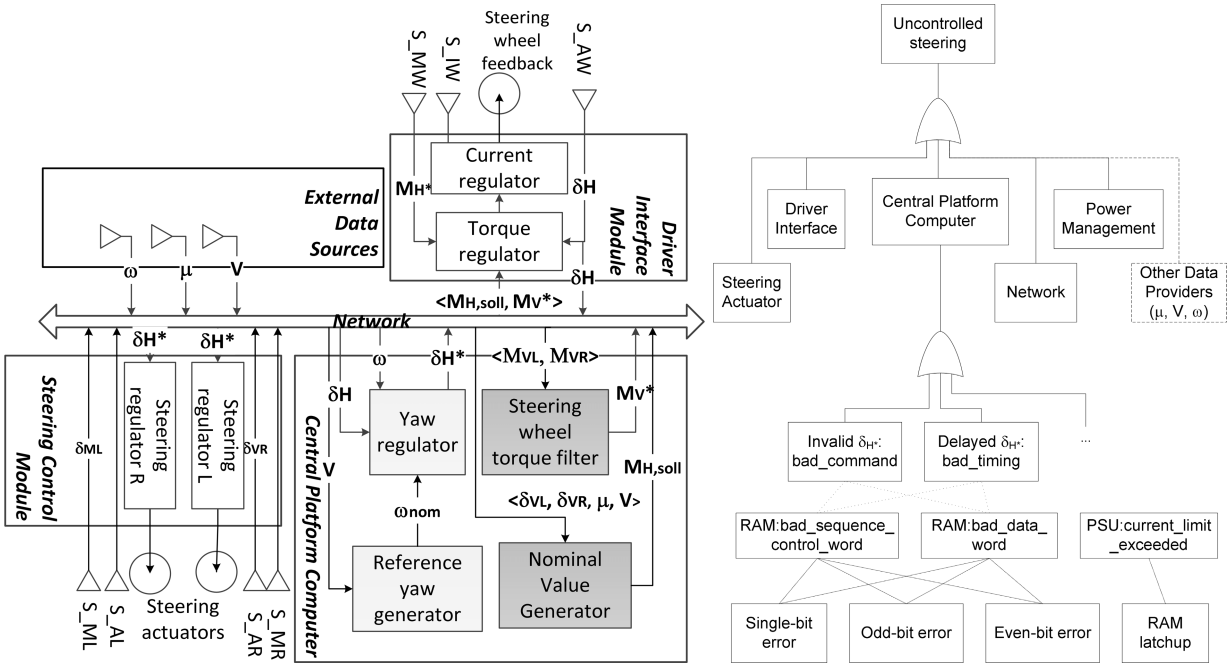
$$LFM = 1 - \frac{\sum(\lambda - \lambda_{\text{SPF}} - \lambda_{\text{RF}} - \lambda_{\text{MPF\_latent}})}{\sum(\lambda - \lambda_{\text{SPF}} - \lambda_{\text{RF}})}$$

- **Completing the evaluation** Adding summary residual failure rates ($\lambda_{\text{SPF}} + \lambda_{\text{RF}} + \lambda_{\text{MPF\_LATENT}}$) into the top-level fault tree, we quantify conservatively the probability of the top-level event, which is in our case violation of the safety goal.

Evaluation of resource utilization metrics is dependent on the metric selected. In fact, selection of good metrics and (possibly) establishment of a link between those and external dedicated analysis tools (e.g. timing analysis) enables significant increase of architect's outlook.

[5]Scope of multiple point fault analysis is limited to order of two unless higher-order faults are shown relevant by the safety concept.

[6]Here we use for simplicity reasons a quantification model without accounting on failure rate classes. Specific constraints are defined precisely in [14, part 5-9.9.4.]



(a) System structure (b) Fault tree, failure effects and failure modes

Figure 4: Application: steer-by-wire system

6. APPLICATION

We have chosen a steer-by-wire system (figure 4a) as a motivating example. There are multiple reasons for that: *a*) the steer-by-wire system features a number of multi-level control loops, which makes it hard to analyze fully using traditional techniques like FTA; *b*) it includes the driver input controls and steering axis-side sensors and actuator, which means that it is surely deployed on at least three different ECUs. The steer-by-wire system functionality and initial architecture blueprint are inspired by [20] and [7]: the steering-related functions are allocated at the nodes of an ICT with centralized concept (figure 4a). To fit in this paper, we limit our analysis by one item and one class of components, namely volatile memory (RAM). Typical mechanisms applied to RAM are common for other component types. These include different levels of replication, monitoring with redundant signatures, functional tests and application-specific plausibility checks.

Architecture draft. To further simplify the analysis, we analyze the parts of steer-by-wire system on the "Central Platform Computer" (CPC) node as groups of functions (differently coloured in fig. 4a). "Steering Control" function group controls steering regulator by providing steering command δ_{H*} to it, "Driver Feedback" receives data and provides tactile feedback to the steering wheel (DI node). Both classes of functions use data from external systems, reading sensor and steering regulator state values (SCU node). Developing the ICT platform, we want to extract requirements to CPC components and establish an EEA design process, which keeps system safety in specific limits required by the standard.

Identification of safety requirements. This function has no safe state: safety is maintained through maintaining "operational" mode of this function. The safety goal is "no uncontrolled steering should occur", fault tolerant timespan requirement: $\Delta\ T_{max} = 50ms$. For all usage scenarios except parking, we classify probability of exposure factor as E4, controllability factor as C3 and severity factor as S3 (all maximal values possible). The resulting ASIL level is D. ASIL D requirements according to ISO 26262 (metric values and probability of safety goal violation) are:

$$SPFM > 99\%;\ LFM > 90\%; P_{VSG} < 10^{-8}h^{-1}$$

Failure rate assumptions. We use conservative estimations for a typical DRAM chip: $\lambda_{hard} = 500\ FIT$, $\lambda_{soft} = 100000\ FIT$.

Top-down analysis. We identify the nodes and failure effects that can cause violation of a safety goal. For CPC: node-level failure effects: invalid command (*bad_command*), late (or missing) command (*bad_timing*). A fragment of the corresponding fault tree is given in figure 4b. Next, two RAM failure effects are traced up to top-level failure effects (data faults may cause incorrect branches and thus also lead to time-domain failure effects):

$$bad\_sequence\_control\_word \rightarrow \{bad\_timing, bad\_command\}$$

$$bad\_data\_word \rightarrow \{bad\_timing, bad\_command\}$$

These failure effects map to the following RAM failure modes (table 2): single-bit (SB) error, odd-bit (OB) error, even-bit (EB) error; each of these failure modes are further decomposed into permanent (P) and transient (T).

At the initial level of detail, selection of safety mechanisms is driven by the requirement to achieve maximal possible coverage of all the identified failure modes. We form our **repository of safety mechanisms** from our knowledge in the problem domain and information from [14, part 5]. The

Mechanism	Correction possible	Maximum total coverage, %	SB P, %	SB T, %	OB P, %	OB T, %	EB P, %	EB T, %
Monitoring with parity	No	60	100	100	100	100	0	0
Monitoring with EDC	Partial	99	100	100	0-50	0-50	20-60	20-60
Monitoring with signature	No	99	100	100	50-100	50-100	40-90	40-90
Block replication	Yes	99	99	100	99	100	99	100
RAM test	No	99	100	0	20-98	0	20-98	0

Table 2: Maximal coverage of safety mechanism classes [14, Part 5, Annex D]

set of safety mechanisms is represented in table 2, annotated by coverage limits for certain failure modes. We complete the data from the standard with our knowledge (e.g. any RAM test is capable of detecting only permanent faults, so its coverage limits are aligned proportionally between failure modes). At the first cycle we use the upper and lower range limits for coverage for plausibility check: if it is generally possible to reach required level of safety.

Architectural decisions driven by ISO requirements. Our strategy is to select a set of mechanisms that provides high coverage over all failure modes. As our goal is fail-operational behavior, another important quality of safety mechanisms is the ability to correct detected errors. We can see that RAM tests can reach high coverage at all "permanent" failure modes, which are uncorrectable by their nature. Transient faults have to be detected and either corrected or result in reactions, which allow delivery of steering command within fault-tolerant timespan (e.g. repeated calculation if the input data has not been corrupted). Simple analysis shows that block replication is the only option providing both correction capability and highest coverage [7].

We select a combination of two mechanisms for the CPC node RAM: a functional test, a detection mechanism with high coverage (signature monitoring) and replication. In this case SPFM value is very close to 100% as more than 99% of the single point faults are covered by replication mechanism, and residual faults may only occur as a result of implementation of replication mechanism (e.g. software code in EEPROM, CPU errors, etc.). Latent faults are covered to a great extent by the signature monitoring, so an LMF value of $\geq$ 97% can be claimed. P_{VSG} metric can only be evaluated in combination with other components. If we assign certain budget of failure rate to memory (e.g. $10^{-9}h^{-1}$ out of $10^{-8}h^{-1}$), and assume that $P_{VSG} \approx \lambda_{perm\_residual}$ then $P_{VSG} \approx= \lambda_{hard} \times (1 - DC_{test}) \times (1 - DC_{replication_perm}) = 500 \times 0.0099 \times 0.01 \times 10^{-9} = 0.495 \times 10^{-9}$. So, if a correct instantiation of safety mechanisms is possible, ISO 26262 quantitative requirements will be satisfied.

Evolution of component models occurs by increasing detail level of the component. In our case, at first stage SRAM or DRAM is selected based on cost and size requirements, fault hypothesis and comparison of achievable metric values. Further selection includes different component features that lead to update of fault hypothesis or inclusion of additional safety mechanisms. Two examples are provided below:

- **Update of the fault hypothesis.** Some RAM chips implement scrambling techniques to achieve chip architecture and reliability optimization. For example, a distributed folding scheme used in a logically 2K*32 memory chip allows using a 256*256 memory cell array, so the bitlines in the chip are kept short. A side-effect is hardware bit interleaving, so in the memory chip each two logically adjacent bits of a single 32-bit word are physically separated by 7 bits of other words. This causes changes in the fault hypothesis of the RAM chip, stating that odd-bit and even-bit faults are less likely in favor of single-bit faults.
- **Implicit safety mechanisms.** Some RAM chips implement an EDC circuit which implements (typically) parity-modified Hamming code with Hamming distance $d = 4$. This allows to detect any two-bit errors and correct single-bit errors. Using such a RAM component requires that we update the set of implicit mechanisms associated with it. This also introduces new failure modes: an error in the ECC circuit can deliver $false\_positive\_correction$ (data corruption happens) and $false\_negative\_correction$ (correction of corrupted word does not happen). We include these failure modes into the fault hypothesis, and will be in the next iteration looking for safety mechanisms which can cover these.

Now let us consider **evolution of safety mechanism models** with "Monitoring with signature" mechanism class. At the next design stage we need to select from a number of mechanism classes with higher implementation detail: "Monitoring with CRC", "Monitoring with modified checksum". CRC-based mechanism requires higher computational effort and (optionally) an array of constants for computation. Checksum-based mechanism suffers from lower detection capabilities, and plausibility check is required at this level to choose the right evolution direction. Encapsulated models for evaluation of both coverage, cycle count and ROM footprint provide the architect with data which drives CRC selection. The next level decomposition of "Monitoring with CRC" leads to several options including CRC polynomial size (8, 16, 32) and decomposes further to specific polynomials. Evaluation models, based on available evidence (e.g. [19]) are applied to select the right polynomial size. As a final decision, specific implementation algorithm[8],block size and monitoring schedule are selected.

7. RELATED WORK

The rapid growth of the number and importance of E/E systems resulted in tailoring of IEC 61508 [12] standard for automotive domain and development of ISO 26262 safety

[7]Similar observations for other node components (especially, for CPU) might lead to an update of architecture blueprint (duplicate the CPC node) and resulting reformulation of requirements to the CPC node (fail-silent behavior).

[8]There are at least three generic implementations that provide different balance between performance and ROM footprint [19].

standard [14]. It completes the existing homologation regulations - (e.g. FMVSS[9] or ECE norms[10]) by introducing additional constraints on the way how the components of E/E systems are developed or reused, integrated and verified.

In many cases ISO 26262 is seen only as a collection of process practices, which are important for developing a dependable product in limited time [17]. This leads to full or partial avoidance of quantitative measures. In [15] the authors present a methodology with the goal of integration of architecture and failure net modeling, allocation of safety mechanisms to architectural elements, and traceability to requirements and test coverage. The discussion, however, concentrates around systematic requirements tracking (DOORS), reflection of requirements to system architecture (SysML) and tracing them down to analysis tools. The authors of another engineering support method [11] take an artifact-centric point of view and concentrate on the modeling approach of all aspects of E/E architectures, which is implemented in an emerging "PREEvision" toolset. Selection of right safety mechanisms is, however, not supported sufficiently in the mentioned tools.

There are a number of developing model-based dependability analysis techniques that are aiming at specification and automated analysis of EEA dependability. FTOS [6] is a tool for synthesis of fault-tolerant real-time systems. It provides a system engineering approach which allows (under an assumption of model correctness) generation of source code for real-time systems. FTOS allows modeling of failure modes of components and their probabilistic behavior, but does not provide quantitative evaluation of achieved system safety.

An approach to selection of safety mechanisms has been proposed in [22]. It is similar to our approach in that a "library of diagnostic techniques" is used to deliver safety mechanisms for IEC 61508 compliance. The evaluation of alternatives during selection stage in performed inside the library and is not specified in detail that allows comparison with our approach.

Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS, [18]) is a top-down methodology to perform automated safety analysis of component-level hierarchical models using information from interface-focused FMEA. Relation to top-level safety goals is maintained through manually performed functional failure analysis. Optimization of dependability is addressed by HiP-HOPS [1], but the evolutionary concept is missing: a separate model is required for early design stages. Often detailed fault propagation logic is not available for components of automotive systems, which renders the approach less useful than in military and aerospace domains. Other related approaches to failure logic analysis build on component fault trees (CFT, [16]), which wrap the failure behavior into graph-based modularized fault trees, and on Fault Propagation and Transformation Calculus (FPTC, [23]), where a clear notation is used to describe the local failure logic of the components. These methods are not intended to provide full engineering support and can be used in combination with our design process.

In the modeling area most relevant standards are MARTE[11] – a UML profile for modeling real-time systems, and EAST-ADL[12] – a domain-specific language for development of automotive electronic systems. MARTE as a real-time profile concentrates on precise timing behavior modeling. Due to its generic nature MARTE supports modeling of other quality attributes including failure behavior. EAST-ADL provides multiple levels of vehicle description, where "Design Level" models are in many points similar or intersecting with our concepts (e.g., hardware components are put in a hierarchy using prototypes). We see both MARTE and EAST-ADL as perspective metamodels compatible with our approach.

Our method is based on an application-specific safety prediction framework. Similar to generic framework presented in [9], it includes all the elements required (encapsulated evaluation models, operational / usage profiles, composition algorithm and evaluation algorithm) but is adapted to automotive system domain and its specifics. The key difference is the optimization-centric selection and configuration of safety mechanisms.

In summary, system synthesis oriented tools currently lack the safety constraints and decision support for early modeling steps. Our design methodology can provide significant benefits for the evolution-time analysis and optimization of E/E architectures.

8. CONCLUSION AND OUTLOOK

In this paper we presented a method to evaluate design choices early in development process and add architectural detail until a specific safe and cost-effective E/E architecture is derived. We encapsulate an important part of domain knowledge (coverage of safety mechanisms and associated resource tradeoffs) in hierarchical models, which results in higher determinism of design decisions.

As a side effect, evidence and arguments on safety-essential attributes, such as coverage and fault models, are continuously accumulated within design infrastructure, which makes reuse of this data, and even of the whole architecture blueprint, possible. To provide maximum benefit, the presented approach has to be applied in a model-based development process. In such a setting requirement-driven iterative process provides high traceability from input requirements to implementation and quantitative argumentation necessary for ISO 26262 certification. Our preliminary evaluation with a practical application shows that iterative evolution allows concentration at sufficiently high modeling level (thus, reducing the required level of expertise) to make design decisions. Exact fault models are used (if necessary and available at all) only at late design steps, when related information is available. Set of metrics for evaluation of design quality attributes (particularly, resource-related constraints) is not covered in this paper and is subject for further definition. Development of repositories and applicability of the approach at the larger scale (as well as issues of integration with real processes in the industry) are very important and will be considered in the nearest future.

Further refinement and formalization of our methodology heads in multiple directions. We are working on tooling

[9]Federal Motor Vehicle Safety Standards is a series of regulations issued by National Highway Traffic Safety Administration

[10]Regulations issued by United Nations Economic Commission for Europe

[11]Modeling and Analysis of Real-Time and Embedded Systems, http://omgmarte.org

[12]Electronics Architecture and Software Technology – Architecture Description Language, http://www.east-adl.info

support that can handle SysML models as input. SysML models are already used to represent automotive architectures: they have been proven suitable for continuous data streams [15], and are often used in a tight combination with MARTE profile, which allows coordination of safety mechanism selection with real-time behavioral models of the E/E architecture. Another perspective extension is related to AADL[13]. AADL Error Annex uses a state-based formalism to define fault models. Taking into account existence of analysis methods for AADL fault models, and the fact that AADL is a widely used cross-industrial standard, integration of our approach with AADL models has a good chance to result in a solid standard-supported toolchain.
Development and integration of engineering support tools remains an important prerequisite of a qualitative change in the E/E architectures. Our methodology has the potential to speed up preliminary safety analyses for automotive systems, assisting safety architect in generating design decisions that allow rapid rapid adaptation to ISO 26262 development lifecycle.

9. REFERENCES

[1] M. Adachi, Y. Papadopoulos, S. Sharvia, D. Parker, and T. Tohdo. An approach to optimization of fault tolerant architectures using HiP-HOPS. *Software – Practice and Experience*, 41(11):1303–1327, 2011.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[3] D. Batory. Feature models, grammars, and propositional formulas. In H. Obbink and K. Pohl, editors, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[4] A. Birolini. *Reliability Engineering : Theory and Practice*. Springer, 2010.

[5] C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Staehle, and A. Knoll. The software car: Building ICT architectures for future electric vehicles. In *Proceedings of the first IEEE International Electric Vehicle Conference (IEVC)*, 2012.

[6] C. Buckl, D. Sojer, and A. Knoll. FTOS: Model-driven development of fault-tolerant automation systems. In *ETFA*, pages 1–8, 2010.

[7] US 6 219 604. Steer-by-wire steering system for motorized vehicles E. Dilger, P. Ahner et al., 04 2001.

[8] T. Dittel and H.-J. Aryus. How to "survive" a safety case according to ISO 26262. In E. Schoitsch, editor, *SAFECOMP 2010, LNCS 6351*, pages 97–111, 2010.

[9] L. Grunske. Early quality prediction of component-based systems – a generic framework. *Journal of Systems and Software*, 80(5):678 – 686, 2007.

[10] H. Heinecke, W. Damm, B. Josko, A. Metzner, H. Kopetz, A. L. Sangiovanni-Vincentelli, and M. D. Natale. Software components for reliable automotive systems. In *DATE*, pages 549–554, 2008.

[11] M. Hillenbrand, M. Heinz, N. Adler, K. D. Müller-Glaser, J. Matheis, and C. Reichmann. ISO/DIS 26262 in the context of electric and electronic architecture modeling. In H. Giese, editor, *ISARCS 2010, LNCS 6150*, pages 179–192, 2010.

[12] IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission (IEC), TC 65/SC 65A, 2010.

[13] R. Isermann, R. Schwarz, and S. Stolzl. Fault-tolerant drive-by-wire systems. *IEEE Control Systems*, 22(5):64–81, 2002.

[14] ISO 26262:2011. Road vehicles - Functional safety. International Organization for Standardization (ISO), TC 22/SC 3, 2010.

[15] B. Kaiser, V. Klaas, S. Schulz, C. Herbst, and P. Lascych. Integrating system modelling with safety activities. In E. Schoitsch, editor, *International Conference on Computer Safety, Reliability and Security*, pages 452–465, 2010.

[16] B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian workshop on Safety critical systems and software - Volume 33*, SCS '03, pages 37–46, Darlinghurst, Australia, 2003.

[17] P. Löw, R. Pabst, and E. Petry. Normiert auf die strasse. *iX kompakt*, Jan(1):136–138, 2011.

[18] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety*, 71(3):229–247, 2001.

[19] J. Ray and P. Koopman. Efficient high hamming distance crcs for embedded networks. In *DSN*, pages 3–12, 2006.

[20] R. Reichel and M. Armbruster. X-by-Wire platform - concept and design. *Automatisierungstechnik*, 59(9):583–596, 2011.

[21] RIAC FMD97. Failure mode / mechanism distribution. The Reliability Information Analysis Center (RIAC), 1997.

[22] D. Sojer, D. Knoll, and C. Buckl. Synthesis of diagnostic techniques based on an IEC 61508-aware metamodel. In *SIES*, pages 59–62, 2011.

[23] M. Wallace. Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes on Theoretical Computer Science*, 141(3):53–71, 2005.

[13] Architecture Analysis & Design Language (SAE AS5506A), www.aadl.info