2016
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

Efficient Exploration of Safety-Relevant Systems Through a Link Between Analysis and Simulation

Moomen Chaari[*], Wolfgang Ecker[*], Thomas Kruse, Cristiano Novello, and Bogdan-Andrei Tabacaru[*]

Infineon Technologies AG - 85579 Neubiberg, Germany
[*]Technische Universität München

{Moomen.Chaari,Wolfgang.Ecker,Thomas.Kruse,Cristiano.Novello,
Bogdan-Andrei.Tabacaru}@infineon.com

***Abstract*—In the automotive domain, where failures may lead to serious human injuries, the ISO 26262 standard provides state of the art guidelines for safety evaluation. It prescribes a comprehensive flow ranging from hazard analysis and risk assessment procedures to fault injection and simulation techniques.**

In this paper, we present a model-based approach to bridge the gap between safety analysis and fault simulation. By formalizing analysis procedures on the one hand and establishing a model-driven data mapping to fault injection and simulation contexts on the other hand, we achieve an efficient exploration methodology for safety-related systems. The quality and consistency of safety evaluation outcomes are subsequently improved and a considerable speed-up of the safety assessment cycle is realized by reducing up to 60% of the cumbersome manual tasks.

Keywords*—*safety evaluation; failure analysis; FMEDA; fault injection and simulation; model-driven development

I. INTRODUCTION

Integrated circuits are used nowadays in a wide range of safety-critical systems whose operation may affect human life. Therefore, safety assessment represents a key step in the design and manufacturing cycle. Through this assessment, the risk of an erroneous behavior with critical impact on system safety is minimized. For this, designers follow safety evaluation guidelines given by domain-relevant standards (e.g., the ISO 26262 standard for functional safety of road vehicles [1] which has been built upon the IEC 61508 standard [2]). To certify an electrical/electronic (E/E) system as functionally safe in compliance with ISO 26262, an extensive procedure is required. It starts by predicting potential risks, whose causes and effects are subsequently identified. To mitigate failure effects, appropriate countermeasures are deployed. Then, evaluation metrics are computed to provide evidence about the system's safety level. This *probabilistic* safety analysis process relies on manual data gathering and inspection, expert judgment, and spreadsheet based calculations. It is performed using different methods such as Fault Tree Analysis (FTA), Failure Modes, Effects, and Diagnostic Analysis (FMEDA), and Dependent Failure Analysis (DFA). When the targeted safety level is particularly high, the standard prescribes the additional application of fault injection techniques. For this, faults are deliberately inserted into an executable system model whose response is afterwards monitored. Running modified system models on appropriate simulation platforms enables the perception of failure effects, the detection of faulty scenarios leading to critical system failures, and the qualification of already integrated safety/diagnostic mechanisms. Thus, we consider fault injection as a *quantitative* safety assessment process as it delivers measured values for failure rates and coverage metrics.

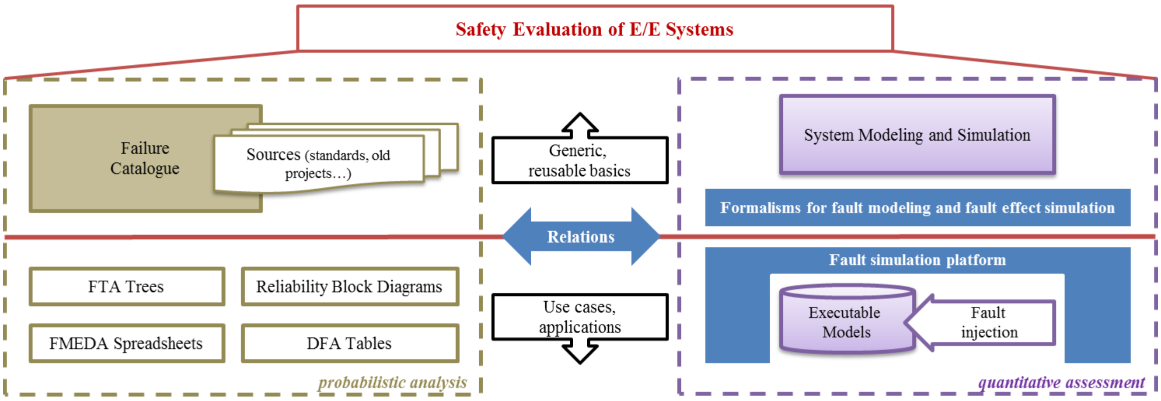

Safety Evaluation of E/E Systems
Failure Catalogue
Sources (standards, old projects…)
Generic, reusable basics
System Modeling and Simulation
Formalisms for fault modeling and fault effect simulation
Relations
FTA Trees
Reliability Block Diagrams
FMEDA Spreadsheets
DFA Tables
Use cases, applications
Fault simulation platform
Executable Models
Fault injection
probabilistic analysis
quantitative assessment

Figure 1 - Different Perspectives of Safety Evaluation

Figure 1 gives an overview of the two considered safety evaluation perspectives. For each perspective, we differentiate between reusable generic basics and application-dependent use cases. On the one hand, a failure catalogue is commonly used in the industry for probabilistic analysis. It represents a database for already known failure modes, recognized failure rates, and similar knowledge derived from relevant reliability handbooks or gathered from older projects. Among tangible reflections of such a failure catalogue, we mention fully constructed fault trees and filled FMEDA spreadsheets. On the other hand, quantitative assessment relies on system modeling and simulation guidelines, extended through formalisms for fault modeling and fault effect simulation. These are applied through fault-injection campaigns performed on executable models within an appropriate simulation platform.

In this paper, we address the connection between safety analysis and fault injection in Section II. After presenting the most relevant safety analysis methods (Subsection II.A) and fault injection techniques (Subsection II.B), differences and potential correlations between both contexts are outlined in Subsection II.C. Afterwards, our model-driven solution to link analysis and simulation is detailed in Section III. After a general presentation of the approach (Subsection III.A), the metamodel-based formalization of safety analysis methods is explained in Subsection III.B and their consequent mapping to fault injection in Subsection III.C.

II. THE CHALLENGE: CONNECTING SAFETY ANALYSIS WITH FAULT INJECTION

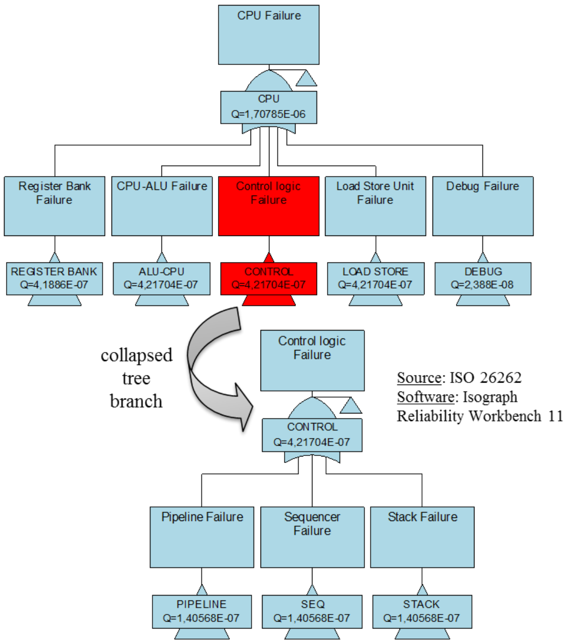
A. *Safety Analysis Methods*

According to the ISO 26262 standard, safety analyses are performed at different stages of the safety evaluation cycle. They are mainly applied to (i) validate safety goals and concepts, (ii) verify safety requirements, (iii) identify conditions and events (faults, failures, etc.) that may cause a safety goal violation, and (iv) determine appropriate counter-measures [1]. The standard mentions several safety analysis methods and differentiates between them with respect to their cause-to-effect exploration direction (deductive versus inductive) and to their level of detail (qualitative versus quantitative).

Fault Tree Analysis (FTA) is a well-established *deductive* (*top-down*) safety analysis approach. It considers undesirable events causing system failures and subsequently analyzes all possible causes. The *fault tree* is an acyclic graphical representation which uses logical connectors and gates and depicts successive levels of events. It is built through a systematic *backward-stepping* process starting at the top-event and leading eventually to the *elementary* (also called *primary*) events [3]. The predominant combinatorial aspect of FTA makes it inappropriate for advanced system descriptions including timing and sequencing. However, it has been extended by different approaches such as Temporal Fault Trees (TFT) and Dynamic Fault Trees (DFT). The TFT approach captures timing dependencies between faults and events. Furthermore, in the DFT methodology, the FTA syntax is extended with additional fault tree gates (e.g., functional dependency and priority AND) to model dynamic behavior of fault-tolerant systems such as sequence-dependent failures. The FTA methodology can be summarized through the following three fundamental observations:

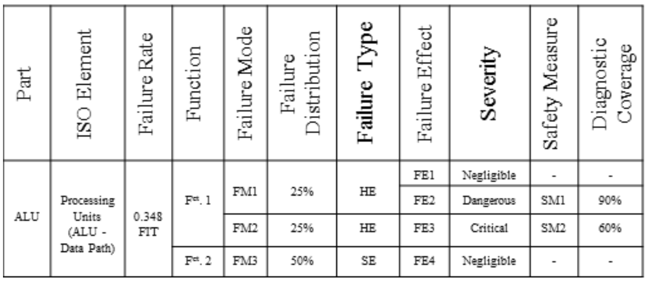
1. *Overall structure*: The analysis is performed top-down starting at the system failure for which all potential causes must be identified. The analysis information is structured using a tree format containing a number of events which are interconnected using *logical combinations* (see example in Figure 2). The tree contains necessarily one and exactly one top event. To enable modularization and allow separate analysis of different system parts, the tree can be optionally decomposed in further sub-trees.
2. *Events*: We differentiate between *top events*, *primary events* (leaf nodes of the tree), and remaining *intermediate events*. Primary events can be *basic* (requiring no further development), *undeveloped* (not further on developed because of unavailable information for example), *conditioning* (specific conditions or restrictions that may apply to the logic gates), or *external*.
3. *Failure logic*: The tree construction is based on backward-stepping from top events to primary events. A non-primary event can be initiated by one single event or by a combination of events. Beyond classical combinations such as *AND*, *OR*, and *XOR* gates, we also consider FTA specific gates such as *VotingOR*.

Another popular safety evaluation technique is Failure Modes and Effects Analysis (FMEA). This *inductive* (*bottom-up*) methodology consists in (i) identifying potential failure modes, (ii) evaluating their impact on system behavior, and (iii) proposing appropriate countermeasures to mitigate them [4]. Basically, FMEA is a qualitative technique. However, a quantitative aspect is introduced through (i) a criticality evaluation (Failure Modes, Effects, and Criticality Analysis - FMECA) and (ii) a diagnostic coverage analysis (Failure Modes, Effects, and Diagnostic Analysis - FMEDA). By analyzing the occurrence probabilities of severe failure effects and characterizing the risk detection through coverage factors and failure rates, design actions are prioritized and failure mitigation mechanisms are refined [5], [6]. FMEDA documentation is done in tabular form, as seen in the simplified extract of an FMEDA spreadsheet given in Table 1. For each safety-relevant system part, an element type according to the ISO standard and a failure rate are allocated. The multiple functions of the part can be affected by different failure modes which are then characterized by their failure distributions (basically the percentage of their contribution to the failure rate of the whole part), their failure types (hard errors vs. soft errors) and their failure effects. These are classified according to their severity and finally, appropriate safety measures to mitigate failure effects are documented along with their diagnostic coverage values.



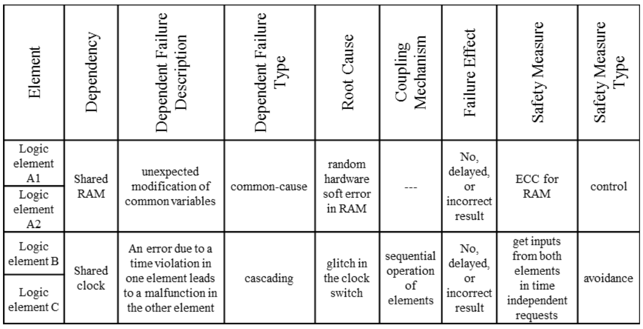
Source: ISO 26262
Software: Isograph
Reliability Workbench 11

Figure 2 - Example of a Fault Tree



Part	ISO Element	Failure Rate	Function	Failure Mode	Failure Distribution	**Failure Type**	Failure Effect	**Severity**	Safety Measure	Diagnostic Coverage
ALU	Processing Units (ALU - Data Path)	0.348 FIT	F^{ct}. 1	FM1	25%	HE	FE1	Negligible	-	-
							FE2	Dangerous	SM1	90%
				FM2	25%	HE	FE3	Critical	SM2	60%
			F^{ct}. 2	FM3	50%	SE	FE4	Negligible	-	-

Table 1 - Simple Example of an FMEDA Table



Element	Dependency	Dependent Failure Description	Dependent Failure Type	Root Cause	Coupling Mechanism	Failure Effect	Safety Measure	Safety Measure Type
Logic element A1	Shared RAM	unexpected modification of common variables	common-cause	random hardware soft error in RAM	---	No, delayed, or incorrect result	ECC for RAM	control
Logic element A2								
Logic element B	Shared clock	An error due to a time violation in one element leads to a malfunction in the other element	cascading	glitch in the clock switch	sequential operation of elements	No, delayed, or incorrect result	get inputs from both elements in time independent requests	avoidance
Logic element C								

Table 2 - Simple Example of a DFA Table

The third major safety analysis technique is Dependent Failure Analysis (DFA) where the probability of simultaneous or successive occurrence of two given *dependent failures* cannot be simply expressed as the product of their respective occurrence probabilities. Dependent failures are classified as *common-cause failures* or *cascading failures*. Common-cause failures result from a single specific event or root cause, while cascading failures are failures of a device element causing at least one other element to fail because of *coupling mechanisms* in the architecture. DFA is documented in tabular form (see example in Table 2).

B. *Fault Injection Techniques*

Fault injection is an approach which has been used over the last decades in order to assess the dependability of a system under test. It consists in the deliberate insertion of faults into a system which is afterwards monitored to determine its behavior in response to the introduced faults. The main purposes of fault injection are the observation of system reactions in faulty environments and the validation of fault detection and/or correction mechanisms. Thereby, appropriate injection spots, observation/diagnostic points, and workloads are determined. Inactive fault locations may be discarded by using so-called operational-profiles (quantitative characterization of the way a system is used [7]). Furthermore, fault collapsing is commonly used to reduce the amount of faults to be injected. Several fault injection techniques are used by the design community. Some of them deal with physical prototypes, while others address virtual models. There are different fault injection categories [8], [9]:

- *Hardware-based fault injection*: It is accomplished at the physical level to affect the actual system's hardware. This may be done by changing environment parameters (e.g., heavy ion radiation, electromagnetic interferences...), modifying circuit pin values, or disturbing power supplies (e.g., power rails manipulation with voltage sags).
- *Software-based fault injection*: It is applied to reproduce faults and errors occurring in the hardware. It consists in introducing faults into applications or operating systems and addresses implementation details, program state, and communications. Based on the application phase, we differentiate between *compile-time* and *runtime injection*.
- *Simulation-based fault injection*: It is conducted at early evaluation stages by introducing faults in register-transfer and gate-level models. It can be performed (i) through *code modification* by adding *saboteurs* or by applying *mutants* or (ii) through *simulator built-in commands* which extend simulation tools to support fault-insertion and response-monitoring.

C. Differences and Potential Correlations

The gap between safety analysis (SubsectionII.A) and fault injection (Subsection II.B) results from their differences in terms of requirements, work styles, and purposes. On the one hand, analysis aims at providing safety evidence at higher abstraction levels and at early design stages. Such safety evidence is achieved by (i) inspecting conceptual system descriptions, (ii) gathering application-related data, (iii) evaluating safety-requirements satisfaction, and (iv) performing statistical calculations. Safety analysis data is structured in multiple ways, such as tables, trees, etc. For large systems considered in industrial contexts, data creation and maintenance are particularly cumbersome and error-prone because of manual tasks. In many cases, safety analysts are confronted with FMEDA spreadsheets containing thousands of lines and with fault trees including hundreds of items. On the other hand, fault injection aims at validating concrete system implementations with respect to safety by (i) extending nominal design models with expected behavior under faulty conditions and (ii) observing the resulting impact through simulation and monitoring. However, there are multiple limitations: the fully deterministic and detailed system model required by fault-injection is only available at a late implementation phase, where redesign is very costly. Furthermore, the tremendous number of possible deviations from correct system behavior makes exhaustive fault-injection campaigns rather unfeasible. It is thus clear that tackling safety evaluation challenges for today's complex systems requires a balanced mixture of both scopes, taking advantage of the benefits and minimizing the drawbacks of each one of them. Therefore, the link between safety analysis (FME(C/D)A, FTA, DFA,...) and fault simulation represents a significant research topic. In our work, we address this challenging issue by considering the structural similarities between both procedures. Indeed, independently from the abstraction layer and the evaluation methodology, three basic element groups are always considered:

1. *Targets*: the system elements or functions that must be protected to ensure the dependable, or more specifically in our case the *safe* operation of the system.
2. *Threats*: malfunctions, discrepancies, failures, and other events against which the targets must be protected.
3. *Countermeasures*: different mechanisms and measures used to protect the targets and mitigate the threats.

Based on this perception, and through a formalization of safety-analysis artefacts making them compatible with model-driven techniques, we establish a correlation between analysis and simulation domains. Thereby, automated data exchange in the system-safety evaluation cycle is achieved, the overall efforts are reduced, and the quality of the resulting outcome is improved.

III. Model-Based Safety Analysis and Link to Fault Injection and Simulation

Our work introduces model-based alternatives to traditional safety analysis procedures and maps them to quantitative assessment (see Figure 1). The application of model-driven development techniques, particularly metamodeling and code-generation, allows a formalization of safety-analysis methods, enables a systematic extraction of relevant analysis data, and offers speed-up and quality improvement by reducing manual tasks.

A. Approach Overview

Figure 3 shows on the one hand the traditional safety evaluation flow and illustrates on the other hand the alternatively proposed flow for model-driven linking between safety analysis and fault simulation.

Traditionally, the safety evaluation is performed by different teams: one for the safety analysis and another one for the fault injection and simulation. The conventional inputs for safety analysis include the design specification, the area information, the safety requirements, the failure catalogue, and the reliability data consisting in the base failure rates for the different component types. Safety engineers take this information as source for the analysis which is documented for example in the FMEDA Excel spreadsheet. Through a tedious manual input, which is very costly and time-consuming, the analysis artefacts are identified and put into the spreadsheet. On the other side, fault injection is performed when it comes to ASIL C and D which are the most stringent safety and integrity levels in ISO 26262. For that purpose, the responsible team inserts faults in the nominal system model through Monte-Carlo oriented campaigns for example. In some cases, the outcomes of the FMEDA are used to define the injection points. Running the corresponding simulations allows the measurement of diagnostic coverage values and potentially the comparison of the simulation results with the analysis outcomes. This correlation between the two contexts requires a highly complicated manual inspection of the analysis data. In real-life cases, where FMEDA spreadsheets contain thousands of lines, going through the analysis data to derive fault injection points is cumbersome and error-prone. That is why we developed an alternative flow to introduce a systematic and automated mapping between the two safety evaluation contexts. The conventional inputs remain basically unchanged. However, the contents of the failure catalogue are covered by a formalized failure modes database along with the reliability data. This database makes it easier to systematically get the relevant information for the analysis.

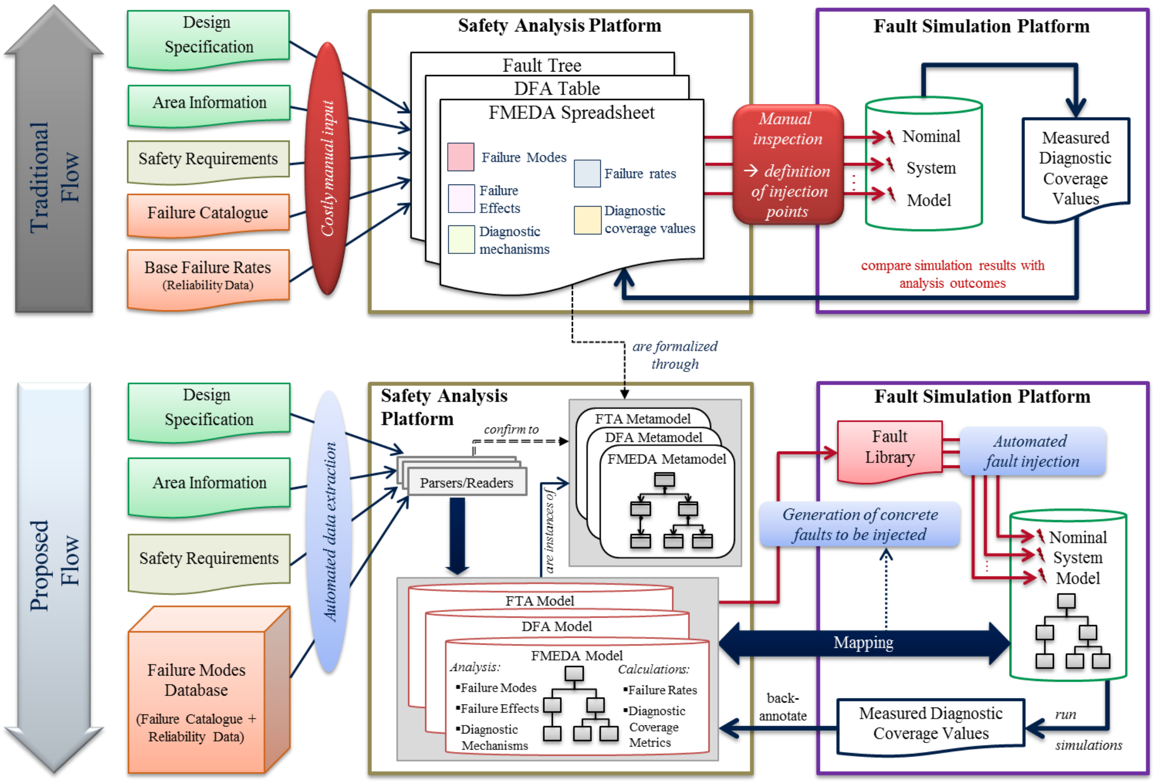


Figure 3 - Approach Overview: Model-Based Safety Analysis and Link to Fault Injection and Simulation

The central point of the solution consists in substituting the traditionally used formats for safety analysis and documentation by corresponding data models built as instances of respective *metamodels*. These metamodels cover the structure of FMEDA spreadsheets, fault trees, and DFA tables, so that the analysis artefacts, their respective properties, and the relationships between them are described in a formal way. Subsequently, the manual data input is replaced by an automated data extraction using a set of parsers and readers leading to the construction of appropriate data models which syntactically and semantically confirms to the metamodels. These data models represent the link to the simulation context. In fact, through model-to-model mapping, specific elements of the safety analysis data models are associated to the appropriate elements of the nominal system model to be simulated. The outcome of the mapping is used to generate the basis of a fault library listing the concrete faults to be injected in the system model: this basis is the list of fault types or classes to be injected and the related injection points in the system model. The fault library is accordingly processed in the fault simulation platform to automatically trigger fault injection campaigns. Eventually, the obtained results through running the corresponding simulations are compared to those of the analysis, allowing consistency and quality checks of the expert judgement on which the analytical methods are based. For example, measured diagnostic coverage values during simulation are back-annotated into the FMEDA data model. Potential refinements are then undertaken if the target values of the safety evaluation metrics have not been reached.

B. Formalized Safety Analysis through Metamodeling

Model-driven development (MDD) is an established approach in system engineering. It reduces complexity and enhances productivity by applying models to raise the level at which applications are developed. A further abstraction step is based on so-called *metamodels* which highlight model properties, specify the relationships between their elements, and determine the constraints that they must validate in a formal way [10]. This formalization aspect offered by metamodeling is the main motive to apply it in the context of our work on safety evaluation. We respectively construct metamodels for FTA [11], FMEDA [12], and DFA [13]. In the remainder of this section, we focus on the FMEDA metamodel represented in Figure 5.

In order to construct the metamodel, we take two basic sources into account. First, we consider the FMEDA steps previously mentioned in Subsection II.A. Second, the structure of the spreadsheets used by safety engineers to document FMEDA outcomes is used as a reference (an example of an FMEDA table has been represented in an abstracted and condensed in Table 1). In the FMEDA metamodel (see Figure 5), the team members and the allocation of their responsibilities are captured by the root class.

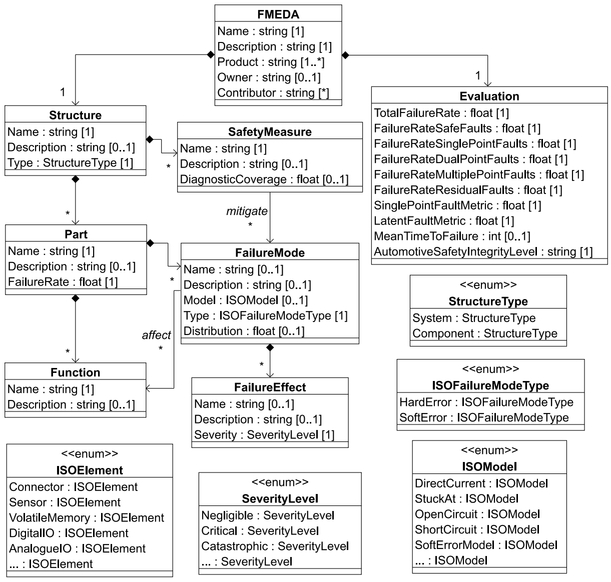

FMEDA
Name : string [1]
Description : string [1]
Product : string [1..*]
Owner : string [0..1]
Contributor : string [*]
1
1
Structure
Name : string [1]
Description : string [0..1]
Type : StructureType [1]
SafetyMeasure
Name : string [1]
Description : string [0..1]
DiagnosticCoverage : float [0..1]
Evaluation
TotalFailureRate : float [1]
FailureRateSafeFaults : float [1]
FailureRateSinglePointFaults : float [1]
FailureRateDualPointFaults : float [1]
FailureRateMultiplePointFaults : float [1]
FailureRateResidualFaults : float [1]
SinglePointFaultMetric : float [1]
LatentFaultMetric : float [1]
MeanTimeToFailure : int [0..1]
AutomotiveSafetyIntegrityLevel : string [1]
*
mitigate
*
*
Part
Name : string [1]
Description : string [0..1]
FailureRate : float [1]
FailureMode
Name : string [0..1]
Description : string [0..1]
Model : ISOModel [0..1]
Type : ISOFailureModeType [1]
Distribution : float [0..1]
*
affect
*
*
<<enum>>
StructureType
System : StructureType
Component : StructureType
*
Function
Name : string [1]
Description : string [0..1]
FailureEffect
Name : string [0..1]
Description : string [0..1]
Severity : SeverityLevel [1]
<<enum>>
ISOFailureModeType
HardError : ISOFailureModeType
SoftError : ISOFailureModeType
<<enum>>
ISOElement
Connector : ISOElement
Sensor : ISOElement
VolatileMemory : ISOElement
DigitalIO : ISOElement
AnalogueIO : ISOElement
... : ISOElement
<<enum>>
SeverityLevel
Negligible : SeverityLevel
Critical : SeverityLevel
Catastrophic : SeverityLevel
... : SeverityLevel
<<enum>>
ISOModel
DirectCurrent : ISOModel
StuckAt : ISOModel
OpenCircuit : ISOModel
ShortCircuit : ISOModel
SoftErrorModel : ISOModel
... : ISOModel

Figure 5 - Simplified FMEDA Metamodel

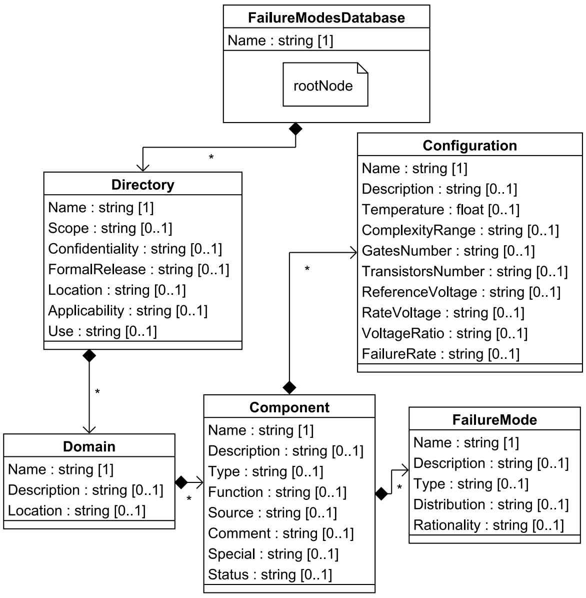

FailureModesDatabase
Name : string [1]
rootNode
*
Directory
Name : string [1]
Scope : string [0..1]
Confidentiality : string [0..1]
FormalRelease : string [0..1]
Location : string [0..1]
Applicability : string [0..1]
Use : string [0..1]
Configuration
Name : string [1]
Description : string [0..1]
Temperature : float [0..1]
ComplexityRange : string [0..1]
GatesNumber : string [0..1]
TransistorsNumber : string [0..1]
ReferenceVoltage : string [0..1]
RateVoltage : string [0..1]
VoltageRatio : string [0..1]
FailureRate : string [0..1]
*
*
Domain
Name : string [1]
Description : string [0..1]
Location : string [0..1]
*
Component
Name : string [1]
Description : string [0..1]
Type : string [0..1]
Function : string [0..1]
Source : string [0..1]
Comment : string [0..1]
Special : string [0..1]
Status : string [0..1]
*
FailureMode
Name : string [1]
Description : string [0..1]
Type : string [0..1]
Distribution : string [0..1]
Rationality : string [0..1]

Figure 4 - Simplified Metamodel for Failure Modes Database

The parts of the structure being analyzed, may it be a system, a component, or a subcomponent, are characterized through their types as given by the ISO 26262 standard, the failure rate they have, and the functions they fulfill. Furthermore, the failure modes affecting these functions are covered along with the corresponding attributes such as the failure type and the failure distribution. The failure effects and their severity levels are also taken into account and the safety measures used to mitigate one or more failure modes are addressed along with their diagnostic coverage. Based on the FMEDA metamodel, a framework is created. It consists of an Application Programming Interface and a set of tools including data readers and parsers, writers, and other plugins used for the construction, visualization, and modification of data models. A key artefact of the framework is a plugin used for importing relevant failure modes from the corresponding database. This database contains information derived from internal failure catalogues, field return data of older projects, as well as fault model lists provided by the ISO 26262 standard in Part 5, Annex D. It also includes typical base failure rate values for component types as given by recognized reliability handbooks or industrial norms such as the Siemens Norm SN 29500. The structure of the failure modes database is also covered by a metamodel (see Figure 4), from which we generate an SQL API to manage the database. Different directories, corresponding to the different potential data sources, are included in the database. In each directory, multiple domains are taken into account (e.g., the processing units and communication domains in the ISO 26262 standard). A set of components, or more precisely component types are related to every domain. Each component is then subject to a group of failure modes, characterized for example by the distribution and the rationale behind it. The component failure rates depend on the technology and operation parameters. This is covered by the database in so-called *configurations*.

C. Model-Driven Data Mapping from Safety Analysis to Fault Simulation

Supporting model-based safety analysis through FMEDA, FTA, and DFA metamodels (see Subsection III.B) is the first step in the transition from safety analysis to fault injection. Indeed, the correspondingly built data models as a substitution for huge amounts of tabular data do not only simplify exploration, maintenance, and reuse. They also represent the starting point for the second step of the transition, namely the mapping towards fault injection processes. By identifying relations between those safety data models (sets of associated objects which can be handled using a common object-oriented programming language such as Python or C++) on the one hand and elements of executable system models (generally developed in SystemC, SystemVerilog, VHDL...) on the other hand, the link between both contexts is established. Here, we achieve a considerable effort reduction by applying model-driven techniques and systematic matching algorithms to identify relevant mapping points.

The first step of the mapping procedure is to capture the nominal system model on which we perform fault injection and simulation. This nominal model can have a more or less complex architecture depending on the considered development stage which may range from early conception to final implementation. Moreover, it can be described at different abstraction levels such as TLM (Transaction Level Modeling), RTL (Register Transfer Level), or GL (Gate Level). Multiple modeling languages are also taken into account. In our use cases, we mainly consider virtual prototypes developed in SystemC which has been extensively addressed in the literature for fault-injection, error simulation, and safety evaluation purposes (e.g., [14], [15]). In order to obtain a generic mechanism for capturing the system model within the model-driven safety evaluation environment, we constructed an abstract metamodel to cover the basic system structure independently from the abstraction level and the modeling language. In fact, despite all syntactical and semantic differences, the system model architecture can be simply represented as a set of interconnected elements constituting a structural hierarchy. Based on this perception, we create the metamodel shown in Figure 6.

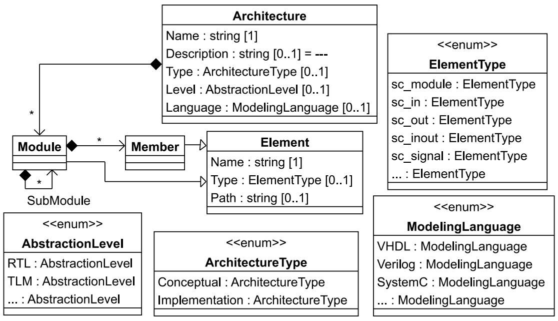


Figure 6 - Simplified Metamodel of System Architecture

	Safety Analysis	**Fault Injection**
Targets	Parts, Functions (FMEDA), Redundant elements (DFA)	Modules, Submodules, Entities, Components…
Threats	Failure modes (FMEDA), Dependent failures (DFA), Events (FTA)	Injection points (signals, ports, variables, sockets, processes…)
	Failure Effects (FMEDA, DFA), Events (FTA)	Observation points (signals, ports…)
Counter-measures	Safety measures (FMEDA, DFA)	Diagnostic and correction points (modules, submodules, signals…)

Table 3 - Mapping Between Safety Analysis Artefacts and Relevant System Elements for Fault Injection and Simulation

Once the nominal system model is captured, its elements are then considered as mapping targets for the different safety analysis artefacts such as safety-relevant system parts, failure modes and effects, and safety measures. Associations between FMEDA/FTA/DFA data models and system model objects (instances of the metamodel classes in Figure 6) are created through appropriate *references* according to the correspondences outlined in Table 3. The mapping is enabled by so-called *matching algorithms* which identify potential candidates (i.e., system model elements) that may be assigned to a specific analysis artefact. Matching algorithms are either based on the unique IDs of the data model objects in both contexts or on their names (more details in [13]).

Afterwards, a configuration step is required. In addition to the identified injection, observation, and diagnostic points, extra details required for the injection are defined (e.g., signal/variable values to be inserted and exact time frames). This configuration is either manually performed or semi-automatically derived from operational profiles.

The final task of the transition between the safety analysis and the fault injection and simulation contexts is the template-based generation of the fault library out of the safety analysis data model which has been extended by mapping and configuration information. In this library, all concrete faults to be inserted into the system model are listed in a manageable format by the associated tools in the fault injection and simulation platform.

IV. MAIN CONTRIBUTIONS

The model-driven safety evaluation environment proposed in this paper offers considerable improvements and benefits in comparison to the traditionally applied procedures for analysis and assessment. Model-based automation support is achieved through the safety-analysis metamodels (for FTA, FMEDA, and DFA) and the associated frameworks facilitating data extraction, visualization, and manipulation. The substitution of classical safety analysis documents by structured and concise data models, built as instances of the respective metamodels, offers opportunities for reuse, consistency checks, and template-based generation of derived model views. Thereby, the process cycle is accelerated and the outcome quality of the quantitative evaluation is increased. In addition to effort savings due to automation support, the environment offers a dynamically extendable failure modes database which provides a well-organized compilation of available prior knowledge about safety and reliability. A special plugin allows the storage of newly added failure modes or failure rate values by the environment users during analysis iterations.

Another essential contribution of this work is the established link between safety analysis and fault simulation in industrial contexts for automotive applications. Semantic and syntactic correspondences are defined and model-to-model transformation techniques are applied to enable structures and systematic data exchange between different teams working in different contexts on decoupled platforms related to safety evaluation. The basic idea behind that is the classification of data model objects on both sides onto three basic groups of safety evaluation elements: targets, threats, and counter-measures. Matching algorithms are then used to identify relevant mapping candidates and generating extensive parts of the fault library to automatically trigger fault injection campaigns. Among the case studies used to evaluate the feasibility and practicability of the proposed methodology, the safety evaluation of a CPU model (MIPS architecture) performed within the new environment shows a reduction of up to 60% of the manual tasks in comparison to the traditional process.

V. CONCLUSION AND FUTURE WORK

In this paper, we addressed the divergences between the two main safety evaluation aspects: probabilistic analysis and fault simulation. As both of them are needed to certify safety-critical systems, a systematic data exchange is convenient. Through model-driven development techniques, we formalize established safety analysis methods in the industry (FMEDA, FTA, and DFA) and create a link to fault injection and simulation. Thereby, manual data entering is replaced by an automated extraction and an extensive framework is used to enhance data handling and visualization. Matching algorithms are applied to map specific artefacts of safety analysis data models to appropriate system model elements. A fault library is accordingly generated and a back-annotation mechanism from simulation to analysis is implemented in order to detect potential deficiencies in the analysis.

In Summary, our approach offers a considerable speed-up of the safety assessment cycle, reduces cumbersome and error-prone tasks, and improves the quality of evaluation outcomes. In the future, the matching algorithms used for the detection of mapping candidates will be improved and the created metamodels will be enhanced to enable automated requirement traceability and implement more appropriate consistency checks.

ACKNOWLEDGMENT

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) in the funded project EffektiV under grant 01IS13022.

REFERENCES

[1] ISO, *CD, "26262, Road vehicles–Functional safety," International Standard ISO/FDIS,* 2011.

[2] IEC, *(International Electrotechnical Commission) and others. "Functional safety of electrical/ electronic/ programmable electronic safety related systems",* 2000.

[3] C. A. Ericson and C. Ll, „Fault tree analysis," in *System Safety Conference*, Orlando, Florida, 1999.

[4] H. Pentti and H. Atte, „Failure mode and effects analysis of software-based automation systems," *VTT Industrial Systems, STUK-YTO-TR*, Bd. 190, p. 190, 2002.

[5] S. Bernardi, J. Merseguer and D. C. Petriu, „Dependability Analysis Techniques," in *Model-Driven Dependability Assessment of Software Systems*, Springer, 20013, pp. 73-90.

[6] W. M. Goble and A. Brombacher, „Using a failure modes, effects and diagnostic analysis (FMEDA) to measure diagnostic coverage in programmable electronic systems," *Reliability engineering & system safety,* Bd. 66, Nr. 2, pp. 145-148, 1999.

[7] J. D. Musa, „The operational profile in software reliability engineering: an overview," in *Software Reliability Engineering, 1992. Proceedings., Third International Symposium on*, 1992.

[8] A. Benso and P. Prinetto, Fault injection techniques and tools for embedded systems reliability evaluation, Springer, 2003.

[9] H. Ziade, R. A. Ayoubi, R. Velazco and others, „A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, Bd. 1, Nr. 2, pp. 171-186, 2004.

[10] S. Shukla, „Metamodeling: What is it good for?," *Design & Test of Computers, IEEE,* Bd. 26, Nr. 3, 2009.

[11] M. Chaari, W. Ecker, T. Kruse, C. Novello and B.-A. Tabacaru, „Transformation of Failure Propagation Models into Fault Trees for Safety Evaluation Purposes," in *Dependable Systems and Networks (DSN) Industrial Track*, Toulouse, France, 2016.

[12] M. Chaari, W. Ecker, C. Novello, B.-A. Tabacaru and T. Kruse, „A Model-Based and Simulation-Assisted FMEDA Approach for Safety-Relevant E/E Systems," in *Proceedings of the 52nd Annual Design Automation Conference*, San Francisco, CA, USA, 2015.

[13] M. Chaari, W. Ecker, B.-A. Tabacaru, C. Novello and T. Kruse, „Linking Model-Based Safety Analysis to Fault Injection and Simulation in Virtual Prototypes," in *Electronic Design Automation Workshop (edaWorkshop 16)*, Hannover, Germany, 2016.

[14] B.-A. Tabacaru, M. Chaari, W. Ecker and T. Kruse, „Runtime Fault-Injection Tool for Executable SystemC Models," in *DVCon India (Design and Verification Conference)*, Bangalore, India, 2014.

[15] B.-A. Tabacaru, M. Chaari, W. Ecker, T. Kruse, K. Liu, N. Hatami, C. Novello, H. Post and A. von Schwerin, „Fault-Injection Techniques for TLM-Based Virtual Prototypes," in *Forum on Specification and Design Languages (FDL) -- Work in Progress (WiP)*, Barcelona, Spain, 2015.