Available online at www.sciencedirect.com

ScienceDirect

IFAC-PapersOnLine 48-7 (2015) 023–028

I-SafE: An integrated Safety Engineering Tool

Pablo Oliveira Antonino, David Santiago Velasco Moncada, Daniel Schneider, Mario Trapp, Jan Reich

Embedded Systems Division, Fraunhofer IESE, Kaiserslautern, Germany
{Pablo.Antonino, Santiago.Velasco, Daniel.Schneider, Mario.Trapp, Jan.Reich}@iese.fraunhofer.de)

Abstract: Traditionally, safety engineering has been a matter of tables and textual documents and even of pen and paper. Even in the age of computerization, this did has not really changed significantly, as the state of the practice in safety engineering is nowadays dominated by Excel sheets and Word files. Nevertheless, a range of computer-aided safety analysis and modeling techniques have emerged and are being put to good use. The problem here is, however, that there is a lack of profound integration between different safety artifacts on the one hand and the general engineering artifacts on the other hand. In addition, between the different safety analysis techniques and the regular engineering techniques, there is usually a range of different tools in use that are not really compatible with each other. To overcome this problem, we conceptualized and implemented an integrated multi-analyses and multi-viewpoint safety engineering tool that enables tight integration between different models within and across different engineering disciplines. This paper gives an overview of the main features of this tool.

Keywords: Safety analysis, safety requirements, architecture, failure model, traceability.

1. INTRODUCTION

Safety engineering artifacts have been defined by means of natural text in documents, spreadsheets, or databases. Sometimes, graphical notations like the Goal Structuring Notation (GSN) (Birch, et al., 2013) and UML (Object Management Group, 2008) are used to provide a more structured overview. Nevertheless, the lack of an underlying formalism of these approaches is a key factor that leads to their incompleteness and inconsistency. For instance, it is still challenging to ensure consistency between artifacts such as safety requirements, failure models, architecture specifications, source code, and test cases. The consequences of this incompleteness and inconsistency become more evident when the system has to be maintained because of, for instance, requirements changes, or system element reuse or modification (Adler, 2013).

One major aspect of this challenge is inconsistency between safety requirements, failure models, and architecture (Antonino & Trapp, 2014). As these are usually created by different teams and in different moments and environments of the system development, they are very often completely disassociated. However, the safety requirements often result from a safety analysis of the architecture, and, lately, must be allocated to elements of the architecture (International Organization for Standardization, 2011). In this regard, the existing inconsistencies and incompleteness result in intensive efforts to update the artifacts impacted by the changes, and, consequently, significantly decrease the efficiency of the safety assurance architecture (Hatcliff, et al., 2014).

To contribute to overcoming this challenge, in this paper we introduce I-SafE: Integrated Safety Engineering, which is an Enterprise Architect[1] extension for supporting safety analysis and the establishment of traceability among safety requirements, failure models, and architecture specification.

With respect to the safety analysis features, I-SafE supports (i) the creation of failure models of the types Component Fault Trees (Domis & Trapp, 2009), Failure Mode and Effect Analysis (Stadler & Seidl, 2013), and Markov Chains (Grinstead & Snell, 2006), which are tightly integrated with elements of the architecture specification, and (ii) the automated execution of safety analyses based on minimal cut sets and top-event probability calculations.

With respect to the traceability establishment, it offers (i) decision support for specifying safety requirements with natural language that are traceable to failure models and to the architecture, (ii) visualization mechanisms for identifying failure models and architecture elements impacted by safety requirements, and (iii) a set of automatic consistency and completeness checks that, among other things, detect inconsistencies in Safety Integrity Levels (SIL) (International Organization for Standardization, 2011).

The remainder of this paper is organized as follows: Section 2 briefly presents a running example. Section 3 presents how I-SafE supports the creation of failure models. Section 4 subsequently describes how safety analyses based on these models can be performed. Section 5 introduces an I-SafE feature enabling the creation of natural language safety requirements that are linked to elements of the architecture and of failure models. Section 6 presents the I-SafE Visual Trace feature and Section 7 the completeness and consistency checks. Section 8 provides a brief overview of related work and Section 9 concludes the paper.

[1] http://www.sparxsystems.com

10.1016/j.ifacol.2015.06.468

2. RUNNING EXAMPLE

The I-SafE features described in this paper are illustrated using a simplified version of a fictitious electric motor drive (E-Drive) system. More specifically, the logical model of the system depicted in **Fig. 1** is used. This simplified system is composed of three sensors: Phase Current Sensor, Rotor Angle Sensor, and Accelerator Pedal Sensor, a component responsible for performing emergency shut-offs, a driver controller, and a microcontroller that performs the central management of the components.

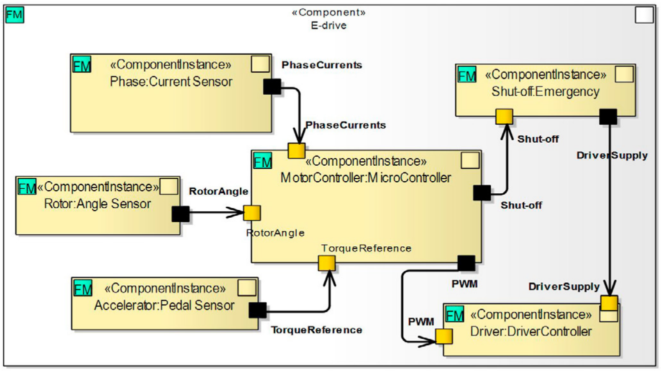


Fig. 1. Logical model of the E-Drive system.

3. CREATION OF FAILURE MODELS

Using architecture component models such as the one depicted in **Fig. 1**, I-SafE supports creating and associating with architecture components failure models of the following types: Component Fault Trees, Failure Modes and Effects Analysis, and Markov Chains, which, in turn, will be presented in this section.

3.1 Component Fault Trees

Component Fault Trees (CFT) extend standard fault trees with the concept of modularity in order to facilitate the fault tree analysis of large model-based systems. For the E-Drive example, **Fig. 2** depicts a CFT for the emergency shut-off component. In a nutshell, it can be seen that the driver supply commission failure mode can be caused by a *faulty emergency shutdown mechanism* basic event, or by the omission of the *shut-off* signal received from the MicroController.

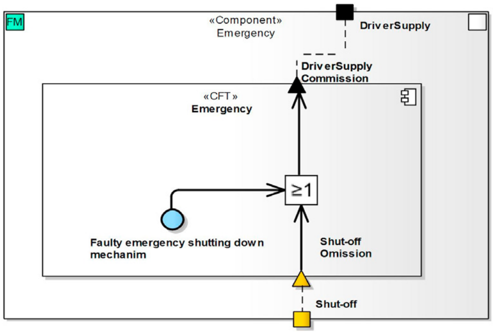


Fig. 2. CFT for the E-Drive's emergency shut-off

3.2 Failure Modes and Effect Analysis

FMEA is a systematic safety analysis method that identifies the possible system failure modes within a system and evaluates the effects on the operation of the system if the analyzed failure mode occurs. I-SafE allows creating interface-focused IF-FMEA (Papadopoulos, et al., 2001) for each system component. For instance, **Fig. 3** depicts an FMEA for the E-Drive's Pedal Sensor; depicted are the failure mode *value measured too high*, its cause *driver acceleration measured too high*, and effects, which are *undesired vehicle acceleration*, and t*orque reference too high*.

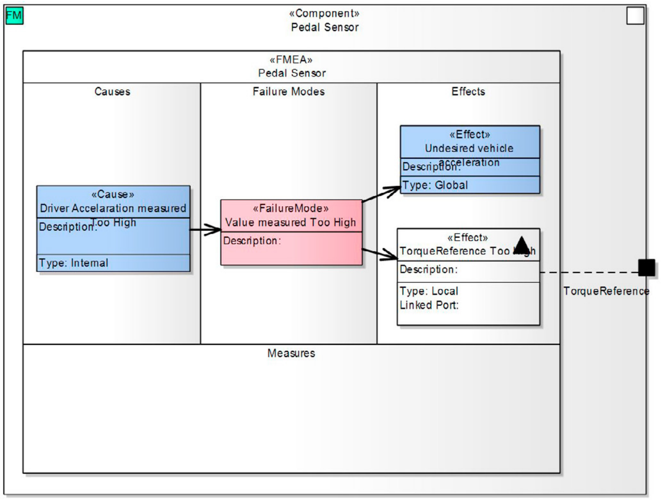


Fig. 3. FMEA for the E-Drive's Pedal Sensor.

3.3 Markov Chains

In order to support the specification of probabilistic relationships between different states of a component and associated failure modes, I-SafE supports the creation of Markov Chains. An example Markov Chain created with I-SafE for the Angle Sensor component of the E-Drive system is depicted in **Fig. 4**.

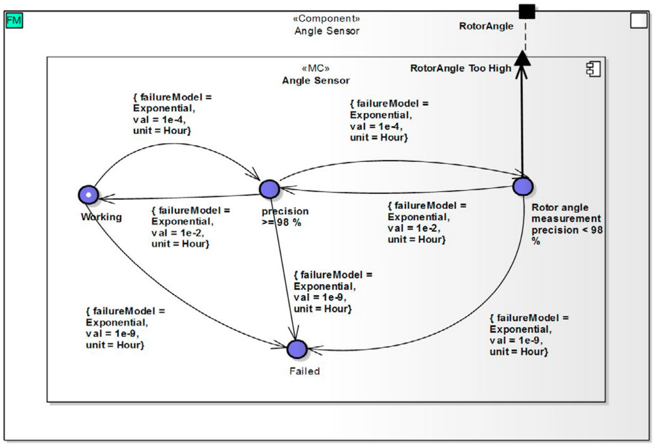

FM
«Component»
Angle Sensor
RotorAngle
«MC»
Angle Sensor
RotorAngle Too High
{ failureModel = Exponential, val = 1e-4, unit = Hour}
{ failureModel = Exponential, val = 1e-4, unit = Hour}
Working
{ failureModel = Exponential, val = 1e-2, unit = Hour}
precision >= 98 %
{ failureModel = Exponential, val = 1e-2, unit = Hour}
Rotor angle measurement precision < 98 %
{ failureModel = Exponential, val = 1e-9, unit = Hour}
{ failureModel = Exponential, val = 1e-9, unit = Hour}
{ failureModel = Exponential, val = 1e-9, unit = Hour}
Failed

Fig 4. Angle Sensor failure model represented with a Markov Chain.

3.4 Structural Propagation Model

Once the modeling of the failure behavior of each component is finished, I-SafE allows modeling the failure behavior of the overall system with the help of the Structural Propagation Model, which corresponds to the composition of the different types of failure models of the components into one higher-level diagram, as can be seen in **Fig. 5**. Thus, with I-SafE, practitioners have the advantage of being able to choose the best-suited technique for specifying the failure behavior of a given component while still being able to work modularly.

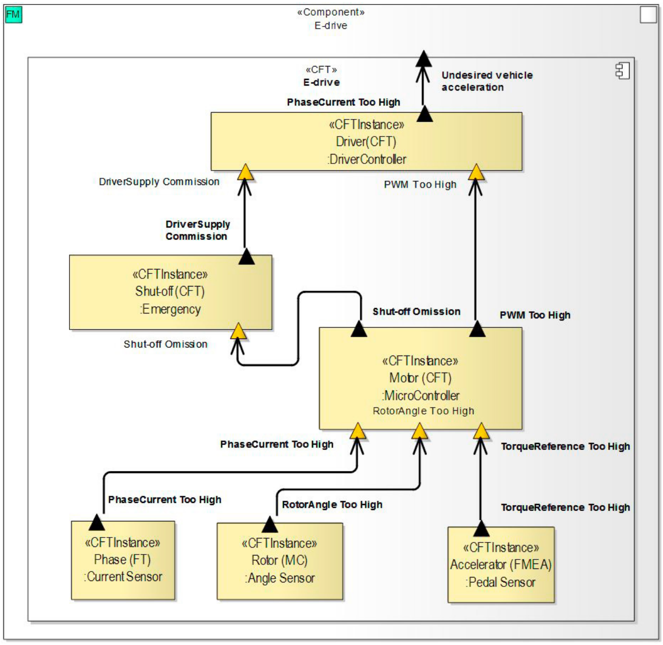


Fig. 5. Heterogeneous failure models composing the failure.

4. PERFORMING SAFETY ANALYSES

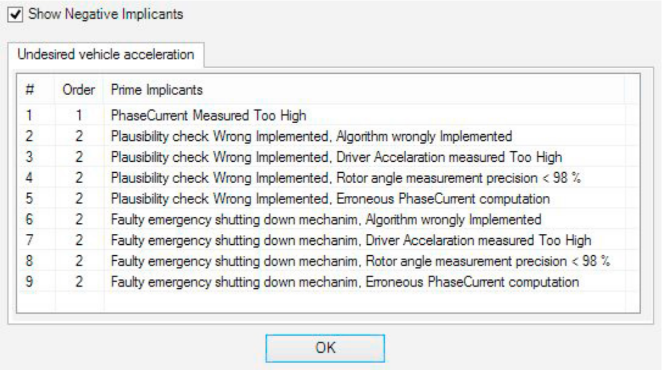
This section introduces two analysis features supported by I-SafE: minimal cut sets analysis and the calculation of the top-event probability. Both types of analysis can generally be performed with different computation tools. The current version of I-SafE supports calculation methods developed at Fraunhofer IESE as well as calculations provided by the tool Fault Tree + [2], which offers support for a large set of probability distributions.

4.1 Minimal Cut Sets Analysis

Minimal Cut Sets is an analysis technique developed for Fault Tree Analysis in which the combinations of basic events resulting in the occurrence of the top event are identified. The focus of the analysis is on obtaining the minimal combinations since these cut sets should be considered with higher priority within the safety lifecycle. **Fig. 6** depicts the results for the *undesired vehicle acceleration* failure mode.

4.2 Top-Event Probability Calculation

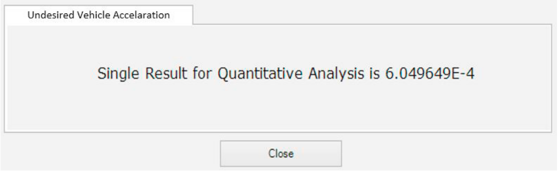
Determination of the top-event probability is one of the most natural and obvious uses of a fault tree. The calculation can be done straightforward based on the fault tree structure and annotated probabilities (and probability distributions) of the base events. As an example, **Fig. 7** depicts the probability of occurrence for the top-level event *undesired vehicle acceleration,* which was computed for the E-Drive system.


☑ Show Negative Implicants

Undesired vehicle acceleration

#	Order	Prime Implicants
1	1	PhaseCurrent Measured Too High
2	2	Plausibility check Wrong Implemented, Algorithm wrongly Implemented
3	2	Plausibility check Wrong Implemented, Driver Accelaration measured Too High
4	2	Plausibility check Wrong Implemented, Rotor angle measurement precision < 98 %
5	2	Plausibility check Wrong Implemented, Erroneous PhaseCurrent computation
6	2	Faulty emergency shutting down mechanim, Algorithm wrongly Implemented
7	2	Faulty emergency shutting down mechanim, Driver Accelaration measured Too High
8	2	Faulty emergency shutting down mechanim, Rotor angle measurement precision < 98 %
9	2	Faulty emergency shutting down mechanim, Erroneous PhaseCurrent computation

Fig. 6. I-SafE's output for Minimal Cut Set Analysis.


Undesired Vehicle Accelaration

Single Result for Quantitative Analysis is 6.049649E-4

Fig. 7. Quantitative analysis results.

5. TRACING SAFETY REQUIREMENTS SPECIFIED WITH NATURAL LANGUAGE TO FAILURE MODELS AND TO THE ARCHITECTURE

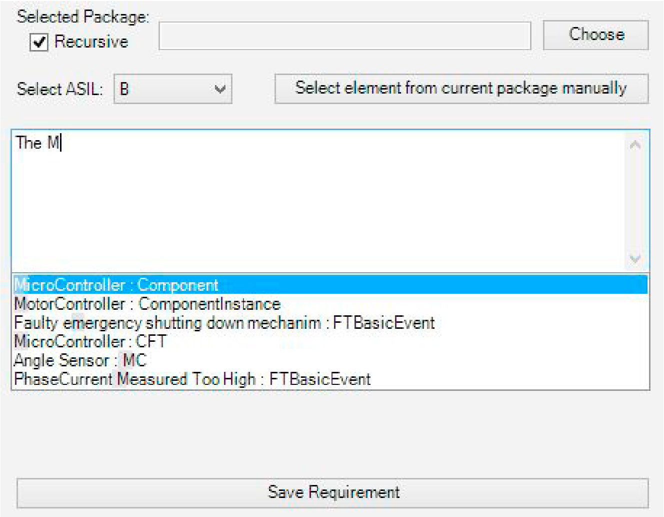
In order to conveniently support the creation of trace links, I-SafE provides an autocomplete mechanism that suggests elements that should be referenced in the safety requirement being specified. These suggestions are made when the text being written has similarities with the names of elements present in the failure models or architecture models. For instance, as shown in **Fig. 8**, as soon as the user starts to type the text fragment “*The M*”, the suggestions of the architecture component “MicroController” (cf. **Fig. 1**), along with other elements that have similarities with this string, such as the MicroController CFT (cf. **Fig. 2**), are shown in the suggestion list.

One benefit of the autocomplete mechanism is that even if a model–based approach is used to specify the safety requirements, free text can still be used; moreover, the writing flow of the specification does not need to be interrupted to select elements of the architecture model to be referenced in the safety requirement. With that, we also support the seamless integration of safety requirements and architectural design without the need to use strict formal

[2] http://www.isograph.com

specification languages like Z, thus preserving intuitiveness during the specification of safety requirements.

Another important aspect is the possibility to reference multiple elements from heterogeneous models, such as architecture and failure models, in one single specification. With that, the safety requirement explicitly references the reason for its existence and the elements that address its demands.



Selected Package:
☑ Recursive

Select ASIL: B

The M|

MicroController : Component
MotorController : ComponentInstance
Faulty emergency shutting down mechanim : FTBasicEvent
MicroController : CFT
Angle Sensor : MC
PhaseCurrent Measured Too High : FTBasicEvent

Fig. 8. I-SafE autocomplete mechanism.

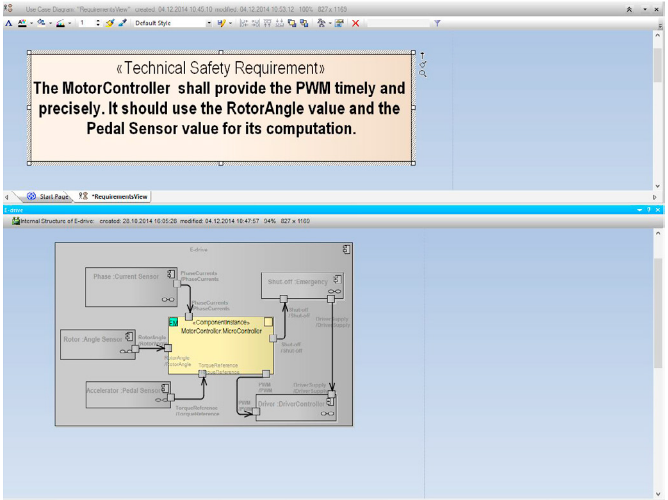
Last but not least, it is also possible to specify the Safety Integrity Level (SIL) of the safety requirement being specified. In the example shown in **Fig. 8**, the SIL specialization for the road vehicle domain (Automotive Safety Integrity Level (ASIL)) is used. Of course, I-SafE was designed to support integrity levels of any domain, including avionics and medical devices, where the terms Development Assurance Levels (DAL) and Software Safety Classes are used, respectively. One important aspect regarding integrity levels is that consistency is assured between the integrity level of the architectural elements and the associated safety requirement.

6. I-SAFE VISUAL TRACE

I-SafE provides a visual trace mechanism that allows engineers to visualize all elements related to each safety requirement specification. It allows visualizing:

- **Directly impacted engineering artifacts**: architectural elements explicitly referenced in textual safety requirements specifications;
- **Indirectly impacted engineering artifacts:** architectural elements that are not explicitly mentioned in safety requirements specifications, but that are related (over a series of indirections) to those that are explicitly referenced. As an example consider components that receive inputs from others or deployment units that deploy a mix of directly and indirectly impacted components;
- **Failure models of directly and indirectly impacted components:** I-SafE allows visualizing the failure models associated with architecture elements created with the I-SafE features described in Section 3;
- **Other specifications related to safety requirements being analyzed:** for instance, there may be guarantees or demands of a Conditional Safety Certificates (ConSerts; Schneider & Trapp, 2013) of a given component that relate to the safety requirement that is being analyzed. These would be highlighted correspondingly.

The visual trace mechanism of I-SafE works as follows: the user activates the Visual Trace mode and from this point on, whenever he/she clicks on a safety requirement, the diagrams that contain elements referenced by it will open and only the referenced elements will be highlighted. In the example shown in **Fig. 11**, the safety requirement is as follows: *"The MotorController shall provide the PWM timely and precisely. It should use the RotorAngle value and the Pedal Sensor value for its computation"*. In this case, only the MotorController element is referenced (cf. **Fig. 1**). After activating the Visual Trace mode of I-SafE, whenever the user clicks on the safety requirement element in the model (cf. **Fig. 9**), the diagram shown in **Fig. 1** opens and only the referenced element MicroController is highlighted. It is important to mention that the user can explicitly choose to display directly and/or indirectly impacted engineering artifacts. In this case, we chose to show only directly impacted engineering artifacts.



Use Case Diagram: "RequirementsView" created: 04.12.2014 10:45:10 modified: 04.12.2014 10:53:12 100% 827 x 1169

«Technical Safety Requirement»
The MotorController shall provide the PWM timely and precisely. It should use the RotorAngle value and the Pedal Sensor value for its computation.

Internal Structure of E-drive: created: 28.10.2014 16:05:28 modified: 04.12.2014 10:47:57 94% 827 x 1169

Fig. 9. I-SafE mechanism for visualizing artifacts related to a safety requirement.

7. AUTOMATED COMPLETENESS AND CONSISTENCY CHECKS

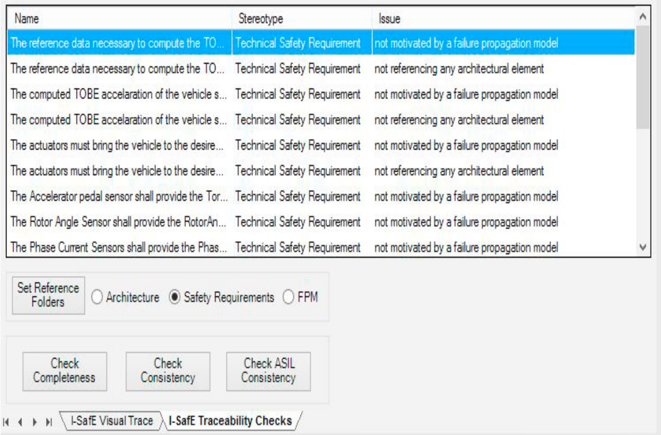
I-SafE also provides support for executing completeness and consistency checks between safety requirements and architecture design, aiming at detecting and alerting engineers to existing inconsistencies.

7.1 Completeness Checks

In I-SafE, we consider a safety requirement to be complete if (i) its motivation is described in failure propagation models, which, in turn, describe the potential failures of each architectural element that might lead to a concrete hazard occurrence and (ii) its content describes mitigation strategies that are realized by performing modifications in the architecture, such as rearrangement of existing elements and connections and insertions of new ones. In this regard, I-SafE's checks whether:

- Every safety requirement describes mitigation strategies for failures that are described in at least one failure propagation model;
- Every failure propagation model describes the failures of at least one safety-critical architecture element; and
- Every safety requirement describes failure mitigations referencing at least one safety-critical architecture element.

The completeness checks are displayed to the user as shown in **Fig. 10**, where the list of safety requirements is displayed, along with their types and the completeness violation.



Name	Stereotype	Issue
The reference data necessary to compute the TO...	Technical Safety Requirement	not motivated by a failure propagation model
The reference data necessary to compute the TO...	Technical Safety Requirement	not referencing any architectural element
The computed TOBE accelaration of the vehicle s...	Technical Safety Requirement	not motivated by a failure propagation model
The computed TOBE accelaration of the vehicle s...	Technical Safety Requirement	not referencing any architectural element
The actuators must bring the vehicle to the desire...	Technical Safety Requirement	not motivated by a failure propagation model
The actuators must bring the vehicle to the desire...	Technical Safety Requirement	not referencing any architectural element
The Accelerator pedal sensor shall provide the Tor...	Technical Safety Requirement	not motivated by a failure propagation model
The Rotor Angle Sensor shall provide the RotorAn...	Technical Safety Requirement	not motivated by a failure propagation model
The Phase Current Sensors shall provide the Phas...	Technical Safety Requirement	not motivated by a failure propagation model

○ Architecture ● Safety Requirements ○ FPM

Fig. 10. I-SafE completeness checks outputs.

7.2 Consistency Checks

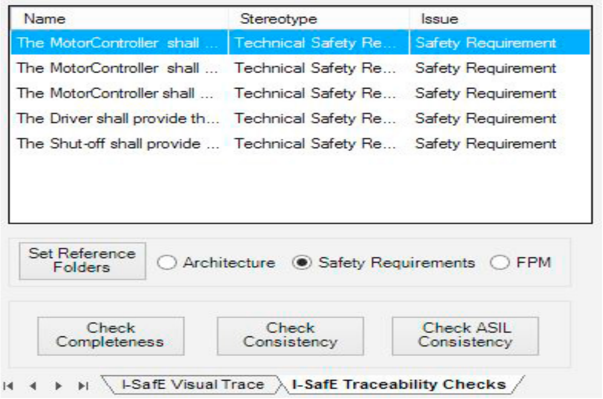
Consistency is a quality attribute that is ensured when there are no contradictions among development artifacts (Zowghi & Gervasi, 2004) (Glinz & Wieringa, 2013). In the scope of this work, we understand consistency to be preserved as long as there are no contradictions among safety requirements, safety-critical architecture elements, and failure propagation models. We understand that these contradictions are triggered by updates or deletions of elements that are already traced. In this regard, I-SafE consistency checks detects:

- For every updated or deleted safety requirement, the safety-critical architecture elements, failure propagation models, and other safety requirements that are impacted;

- For every updated, deleted, or substituted safety-critical architecture element, the safety requirements, failure propagation models, and other safety-critical architecture elements that are impacted;

- For every updated or deleted failure propagation model, the safety requirements and safety-critical architecture elements that are impacted.

For instance, consider the MicroCrontroller, which is already referenced by safety requirements, such as the one depicted in **Fig. 1**. If, for example, a new port is added to this component, the execution of the I-SafE consistency check will display the safety requirements elements that reference this component (cf. **Fig. 11**), which, in turn, must be analyzed by the engineers because, due to a change in the referenced component, they might have become invalid.

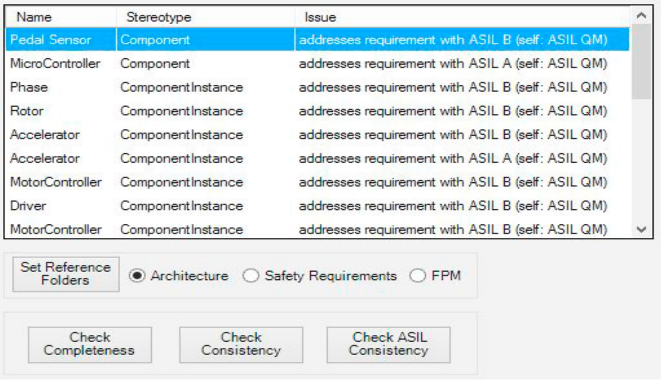


Name	Stereotype	Issue
The MotorController shall ...	Technical Safety Re...	Safety Requirement
The MotorController shall ...	Technical Safety Re...	Safety Requirement
The MotorController shall ...	Technical Safety Re...	Safety Requirement
The Driver shall provide th...	Technical Safety Re...	Safety Requirement
The Shut-off shall provide ...	Technical Safety Re...	Safety Requirement

○ Architecture ● Safety Requirements ○ FPM

Fig. 11. Consistency check result after modifying MicroController.

Another type of consistency check performed by I-SafE regards possible SIL violations, which are caused when safety requirements and the safety-critical architecture elements that address them have incompatible safety integrity levels. In this regard, I-SafE also checks whether the safety requirements are addressed by safety-critical architecture elements with an equal or more stringent safety integrity level. For instance, **Fig. 12** shows a list of architecture elements that have ASIL incompatibility.



Name	Stereotype	Issue
Pedal Sensor	Component	addresses requirement with ASIL B (self: ASIL QM)
MicroController	Component	addresses requirement with ASIL A (self: ASIL QM)
Phase	ComponentInstance	addresses requirement with ASIL B (self: ASIL QM)
Rotor	ComponentInstance	addresses requirement with ASIL B (self: ASIL QM)
Accelerator	ComponentInstance	addresses requirement with ASIL B (self: ASIL QM)
Accelerator	ComponentInstance	addresses requirement with ASIL A (self: ASIL QM)
MotorController	ComponentInstance	addresses requirement with ASIL B (self: ASIL QM)
Driver	ComponentInstance	addresses requirement with ASIL B (self: ASIL QM)
MotorController	ComponentInstance	addresses requirement with ASIL B (self: ASIL QM)

● Architecture ○ Safety Requirements ○ FPM

Fig. 12. ASIL violations detected by I-SafE.

8. RELATED WORK

I-SafE provides functionalities that support safety analysis and creation as well as verification of traceability among failure models, safety requirements, and the architecture.

Most of the tools on the market offer several techniques for performing safety analysis. However, they mostly have implemented traditional monolithic approaches, which do not work well anymore with today's complex systems. In this sense, I-SafE offers a modular and compositional alternative to this problem. Furthermore, through the tight integration with the architecture, it creates the basis for better traceability and reusability.

With respect to traceability features, PREEvision[3] and Medini Analyze[4] are tools that also offer such support. However, with respect to allocation, none of them offers automated decision support to create dynamic traces within textual safety requirements specifications, which, as we have observed, improves not only the completeness and consistency of safety requirements, but also the efficiency of the specification process since the engineers do not have to interrupt the flow of their specification writing to create the traces.

Regarding the visual trace, we have observed that the intuitive navigation mechanism of I-SafE, which displays only diagrams that contain the elements referenced by the safety requirements, is a considerable step towards improvements in terms of efficiency of safety analysis activities. In this regard, neither PREEvision nor Medini or any other tool on the market provides such facilities.

In terms of completeness and consistency checks, there is no other tool that supports automatic detection of orphan elements in the safety architecture or automatic detection of updates in architectural elements already referenced by safety requirements. With respect to ASIL consistency checks, Medini offers some support in this regard; the advantage of I-SafE is that ASIL violations are identified directly when a safety requirement is allocated.

9. CONCLUSION

Even though I-SafE is a non-commercial tool from academia, it provides a range of features that can rarely be found in professional tools. Amongst the features of I-SafE presented throughout this paper, we deem the aspects of integration and traceability particularly important. Integration between different (types of) modular analysis models in the context of a larger system and traceability between safety requirements and related artifacts along the safety engineering chains are features that are bound to make life much easier for engineers. Of course, as an academic tool, the maturity of I-SafE is not on par with professional tools – but this is obviously also not the goal. With I-SafE we hope to demonstrate the feasibility and effectiveness of these concepts so that they will be recognized and maybe eventually be adopted by the state of the practice.

ACKNOWLEDGEMENT

This work is supported by the Fraunhofer Innovation Cluster Digitale Nutzfahrzeugtechnologie. We would also like to thank Sonnhild Namingha for proofreading.

REFERENCES

Adler, R., 2013. *Introducing Quality Attributes for a Safety Concept.* Detroit, Michigan, USA, s.n.

Antonino, P. O. & Trapp, M., 2014. *Improving Consistency Checks between Safety Concepts and View Based. Architecture Design.* Honolulu, Hawaii, USA, s.n.

Birch, J. et al., 2013. *Safety Cases and Their Role in ISO 26262 Functional Safety Assessment.* Toulouse, France, Springer.

Domis, D. & Trapp, M., 2009. *Component-Based Abstraction in Fault Tree Analysis.* Hamburg, Germany, Springer-Verlag, pp. 297-310.

Easterbrook, S. & Nuseibeh, B., 1995. *Managing inconsistencies in an evolving specification.* s.l., s.n., pp. 44-55.

Glinz, M. & Wieringa, R., 2013. *RE@21 spotlight: Most influential papers from the requirements engineering conference.* Rio de Janeiro, Brazil, s.n., pp. 368 -370.

Hatcliff, J. et al., 2014. *Certifiably safe software-dependent systems: challenges and directions.* Hyderabad, India, s.n., pp. 182-200.

International Organization for Standardization, 1998. *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems,* Geneva, Switzerland: The International Electrotechnical Commission.

Leveson, N. G., 2000. *Completeness in formal specification language design for processcontrol systems.* Portland, Oregon, s.n.

Object Management Group, 2008. *UML profile for modeling QoS and FT characteristics and mechanisms.* [Online].

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

Ramamoorthy, C., Ho, G. & Han, Y., 1977. *Fault tree analysis of computer system.* New York, NY, USA, ACM, pp. 13-17.

Reifer, D., 1979. Software Failure Modes and Effects Analysis. *IEEE Transactions on Reliability,* R-28(3), pp. 247-249.

Schneider, D. & Trapp, M., 2013. Conditional Safety Certification of Open Adaptive Systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS),* 8(2), pp. 1-20.

Stadler, J. J. & Seidl, N. J., 2013. *Software failure modes and effects analysis.* s.l., s.n., pp. 1-5.

Yu, Y., 2006. *The Quantitative Safety Assessment for Safety-Critical Computer Systems.* Charlottesville, VA, USA.: Ph.D. Dissertation. University of Virginia.

Zowghi, D. & Gervasi, V., 2002. *The Three Cs of Requirements: Consistency, Completeness, and Correctness.* Essen, Germany, s.n., pp. 55-164.

Zowghi, D. & Gervasi, V., 2004. On the Interplay between Consistency, Completeness, and Correctness in Requirements Evolution. *Journal of Information and Software Technology,* 46(11), pp. 763-779.

[3] www.vector.com/preevision

[4] www.ikv.de/medinianalyze