Delft University of Technology
Software Engineering Research Group
Technical Report Series

Establishing Evidence for Safety Cases in Automotive Systems – A Case Study

Willem Ridderhof, Hans-Gerhard Gross, Heiko Doerr

Report TUD-SERG-2007-008

TUD-SERG-2007-008

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
`http://www.se.ewi.tudelft.nl/techreports/`

For more information about the Software Engineering Research Group:
`http://www.se.ewi.tudelft.nl/`

Establishing Evidence for Safety Cases in Automotive Systems – A Case Study

Willem Ridderhof[1], Hans-Gerhard Gross[2], and Heiko Doerr[3]

[1] ISPS Medical Software, Rotterdamseweg 145, 2628 AL Delft
Email: willem.ridderhof@isps-medical-software.nl
[2] Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
Email: h.g.gross@tudelft.nl
[3] CARMEQ GmbH, Carnotstr. 4, 10587 Berlin, Germany
Email: heiko.doerr@carmeq.com

Abstract. The upcoming safety standard ISO/WD 26262 that has been derived from the more general IEC 61508 and adapted for the automotive industry, introduces the concept of a safety case, a scheme that has already been successfully applied in other sectors of industry such as nuclear, defense, aerospace, and railway. A safety case communicates a clear, comprehensive and defensible argument that a system is acceptably safe in its operating context. Although, the standard prescribes that there should be a safety argument, it does not establish detailed guidelines on how such an argument should be organized and implemented, or which artifacts should be provided.
In this paper, we introduce a methodology and a tool chain for establishing a safety argument, plus the evidence to prove the argument, as a concrete reference realization of the ISO/WD 26262 for automotive systems. We use the goal structuring notation to decompose and refine safety claims of an emergency braking system (EBS) for trucks into subclaims until they can be proven by evidence. The evidence comes from tracing the safety requirements of the system into their respective development artifacts in which they are realized.

1 Introduction

Safety critical systems have to fulfill safety requirements in addition to functional requirements. Safety requirements describe the characteristics that a system must have in order to be safe [12]. This involves the identification of all hazards that can take place, and that may harm people or the environment. Safety-related issues are often captured in standards describing products and processes to be considered throughout the life-cycle of a safety critical system. The upcoming safety standard ISO/WD 26262 [2] is an implementation of the more general IEC 61508 standard that addresses safety issues in the automotive industry. The standard prescribes that a safety case should be created for every system that has safety-related features, and it says that part of the system documentation should provide evidence for the fulfillment of safety requirements, thus guaranteeing functional safety. However, the standard does not provide any details about which artifacts should be produced in order to prove functional safety, nor does it say how such a proof may be devised.

In this paper, we develop a generic safety case that may act as a reference realization for the automotive industry. In section 2, we describe how a safety case may be constructed based on the Goal-Structuring-Notation. Section 3 shows that constructing a safety argument is, to a large extent, a traceability effort and dealing with the construction of trace tables. Section 4 presents the case study, a safety critical system, for which we have devised part of a safety argument. Here, we concentrate on the traceability part. Section 5 summarizes and concludes the paper.

2 The Safety Case

Part of the certification process in the automotive domain is the assessment of a system through an inspection agency. To convince inspectors that a system is safe, a safety case should be created. The safety case communicates a clear, comprehensive and defensible argument that a system is acceptably safe in its operating context [7]. The argument should make clear that it is reasonable to assume the system can be operated safely. It is typically based on engineering judgment rather than strict formal logic [12], and it provides evidence that the risks (hazards) associated with the operation of the system have been considered carefully, and that steps have been taken to deal with the hazards appropriately.

The safety argument (SA) must identify all matters significant to the safety of the system and demonstrate how these issues have been addressed. A convenient way to define a safety argument is through the goal structuring notation (GSN) devised by Kelly [7] which is based on earlier work by Toulmin on the construction of arguments [13]. An argument consists of claims whose truth should be proven. The facts used to prove the claims are referred to as data, and the justification for why data prove a claim is described by warrants. If it is possible to dispute a warrant, backing can be used to show why the warrant is valid. The structure of this argument is depicted in Fig. 1.

The main elements of the GSN are goals and solutions. Goals correspond to Toulmin's claims whereas solutions relate to Toulmin's data, also termed evidence. For constructing a safety case, we have to determine which evidence is required for a particular safety argument, and why the evidence supports the claim. According to the GSN, the safety case starts with a top-level claim, or a goal, such as "the system is safe" or "safety requirements have been realized." The top-level claim is then decomposed into sub-ordinate claims down to a level that a sub-claim can be proven by evidence. The concepts of the GSN are displayed in the example in Fig. 2. Claims and sub-claims, or goals, are represented as rectangular boxes and evidence, or solutions, as circles. A strategy-node, represented as rhomboid, contains the explanation why a goal has been decomposed. The argument can be constructed by going through the following steps [3, 7]:

1. Identification of the goals to be supported.

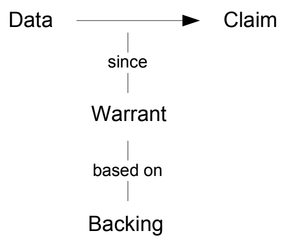


Fig. 1. Structure of Toulmin's Argument [13].

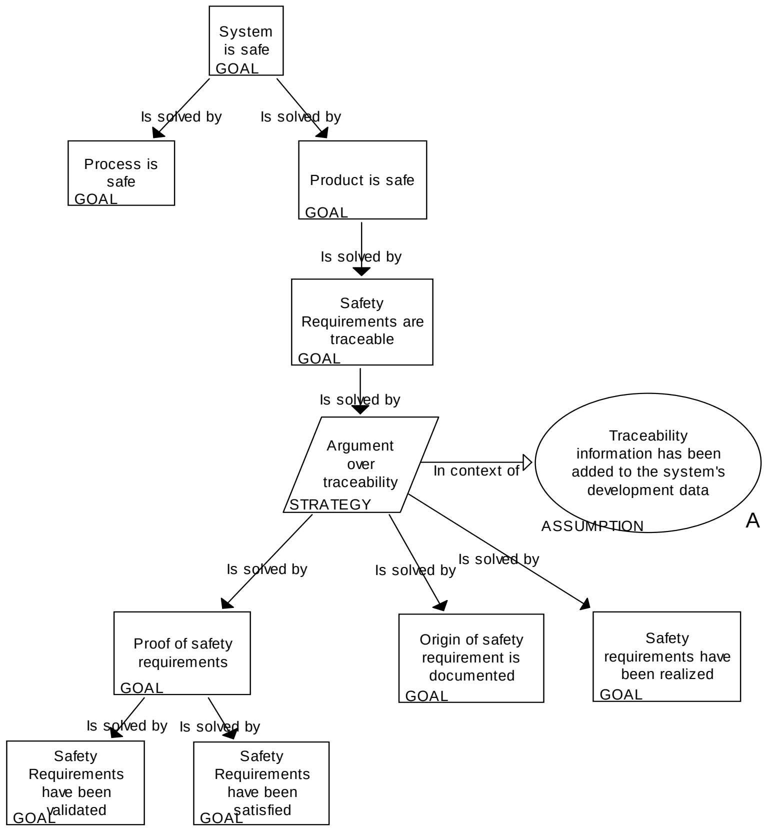

System is safe GOAL
Is solved by
Is solved by
Process is safe GOAL
Product is safe GOAL
Is solved by
Safety Requirements are traceable GOAL
Is solved by
Argument over traceability STRATEGY
In context of
Traceability information has been added to the system's development data
ASSUMPTION
A
Is solved by
Is solved by
Is solved by
Proof of safety requirements GOAL
Origin of safety requirement is documented GOAL
Safety requirements have been realized GOAL
Is solved by
Is solved by
Safety Requirements have been validated GOAL
Safety Requirements have been satisfied GOAL

Fig. 2. Example GSN tree, decomposition of the goal "the product is safe."

2. Definition of the basis on which the goals are stated.
3. Identification of the strategy to support the goals.
4. Definition of the basis on which the strategies are stated.
5. Elaboration of the strategy including the identification of new goals and starting from step 1, or moving to step 6.
6. Identification of a basic solution that can be proven.

It is difficult to propose a standard safety case structure that may be valid for most systems. However, some of the argumentation will be the same for many systems, such as "all safety requirements have been realized" or the like. Such argumentation structures, or so-called safety case patterns [6, 15], may be reused in several safety cases for different systems. By using such patterns, safety cases can be devised much faster. Similarly, safety case anti-pattern can be used to express weak and flawed safety arguments [7, 14].

A particular GSN decomposition proposed by the EU project EASIS [3] organizes the argumentation into a product branch and a process branch, claiming that "a system is safe" if "the process is safe" and "the product is safe". The safety of the process can be assured through application of certified development standards, such as the IEC 61508 [4] or the V-model [11]. Here, questions should be asked about how the product is developed, such as "did we perform hazard analysis?", "do we have a hazard checklist?", "did we perform a preliminary hazard identification?", "did we implement the results of the preliminary hazard identification?", and so on [3].

On the product side, we can decompose the claim "the product is safe" into the sub-goal "the safety requirements are traceable" which turns the satisfaction of our safety case into a traceability problem. We can argue, that if all safety related aspects of our system can be traced to their origin and to their realization, the system is safe, given the process is safe (proof of the process branch). Traceability is a prerequisite for assessment and validation of the safety goals, which we refer to as "proof of safety requirements". We can then decompose the traceability goal further into "proof of safety requirements", "origin of safety requirements documented", and "safety requirements realized". This extended organization of the safety case is depicted in Fig. 2 and, through its general nature, it can be used as a pattern for all systems.

3 Traceability of Safety Requirements

In the previous section, we argued that part of the proof of a safety case can be achieved through tracing all safety requirements to the respective development documents. This is fully in line with the ISO/WD 26262 [2] since it demands that "the origin, realization and proof for a requirement are clearly described in the documentation" of a system. A requirement is a condition or an ability that the system should fulfill. The origin of a requirement is a rationale why this requirement has been elicited for the system. The realization demonstrates how/where the requirement is implemented in the final system. A proof for a requirement means that it should be demonstrated that the requirement has an origin and that it is implemented, in other words, that the requirement is traceable across all development documents in both directions, forwards and backwards. The documents comprise the hazards possible, the safety goals, the safety requirements, design elements, and implementation elements, plus associated review documents.

As shown in Fig. 2, the "product is safe"-branch is decomposed into a traceability sub-goal that is split into various traceability claims, i.e., "origin of safety requirements documented", "safety requirements realized", and "proof of safety requirements". This last goal is decomposed into two sub-goals, "safety requirements validated" and "safety requirements satisfied" which can be traced to the respective documents that deal with those issues.

The origin of a safety requirement can be demonstrated by backward traceability. Safety requirements are derived from hazards and safety goals. Every safety requirement should be linked to at least one safety goal, expressed through $\forall srSR \longrightarrow \exists sgSG$, and every safety goal should be linked to a hazard, expressed through $\forall sgSG \longrightarrow \exists hH$. But also forward traceability is important, so that for every hazard, there is a safety goal ($\forall hH \longrightarrow \exists sgSG$), and for every safety goal, there should be an associated safety requirement ($\forall sgSG \longrightarrow \exists srSR$). Consequently, we can extend our safety case as depicted in Fig. 3.

For the lowest-level goals, we can then come up with solutions in the form of trace tables. These are now product-specific. Once all trace relations have been established in a development project we can claim that the system is acceptably safe with respect to those safety goals, e.g. with respect to "origin of safety requirements is documented." We have to do this for all safety goals defined, and we demonstrate how this may be done for a specific case using specific tools in the next section.

4 Case Study

We have devised a partial safety case for an emergency brake system (EBS). This innovative assistance functionality becomes part of modern vehicles and lorries. DaimlerChrysler, for instance, markets that type of application as Active Brake Assist. An emergency brake system warns the driver of a likely crash, and, if the driver does not react upon the warning, initiates and emergency braking. It is a distributed system incorporating the braking system, a distance sensor, the hifi-system, a control system, as well as the driver's interface. The individual subsystems are usually interconnected through a vehicle's CAN bus.

In order to devise the traceability part of the safety case, first, we have to take a look at the development processes and tools deployed. The safety requirements (SR) for this system are coming from a preliminary hazard analysis (PHA), or from a hazards and operability analysis (HAZOP) [10]. Once all the potential hazards have been identified, they are associated with safety goals (SG). Safety goals are comparable to top level functional requirements. The requirements are decomposed into sub-system requirements, and eventually, into component requirements. All requirements are managed within Telelogic's DOORS, a widely used requirements management tool (http://www.telelogic.com). They are associated with different levels such as safety requirement, safety concept, system requirement, component specification, etc, and the tool maintains also traces between those levels. These traces are shown in Fig. 4.

Apart from DOORS for the requirements management, various other tools should be used throughout the other development phases in order to comply to standard's requirements. A typical tool chain could consist of:

– Matlab's Simulink and Stateflow (http://www.matlab.com) are used for system and component design,
– DSpace's TargetLink (http://www.dspace.com) is used for the implementation and automatic code generation out of the Matlab models.
– Tessy (by Hitex) is used for automated unit testing,
– the Classification Tree Editor (CTE by Hitex) is used to support input domain-based testing,
– Time Partition Testing is employed for generating test cases with continuous input data streams [8, 9],
– DSpace's MTest generates test cases automatically based on the Simulink and TargetLink models (http://www.dspace.com),
– QA-C/Misra can be used to analyze the resulting C-code (http://www.qasystems.de),
– PolySpace is a tool that can detect run-time errors during compile time (http://www.polyspace.de), and
– Mercury's Quality Center is a global suite of tools for ensuring software quality (http://www.mercury.com).

4.1 ToolNet

The tools in the previous list are dealing with different types of work products, and the trace tables that we have to devise for the safety case have to refer to the artifacts stored in these various tools. In other words, in order to realize a full tracing between the different work products, we have to gain access to the tools' various data representations. This is done through another tool, called ToolNet [1] which enables us to create traceability links between various development artifacts independent from the type of tool through which they have been created. ToolNet is based on a bus-architecture, the so-called information backbone that connects each tool via an adapter [1] (Fig. 5). Every single development object recognized by ToolNet is assigned a unique ID (object reference) which is based on its data source (a tool or part of a tool). The development objects must be defined unambiguously, according to a product-specific information model [5]. It describes the available and traceable development objects such as hazard, safety concept, safety requirement, etc. An example information model for our project is displayed in Fig. 6. The development object models and the ToolNet structure permit the tools to interact witch each other through services implemented in their respective adapters. A user can select a specific development object from one tool, e.g., a requirement within DOORS, and associate it with another development object, e.g., from Matlab/Simulink, and eventually link that to an

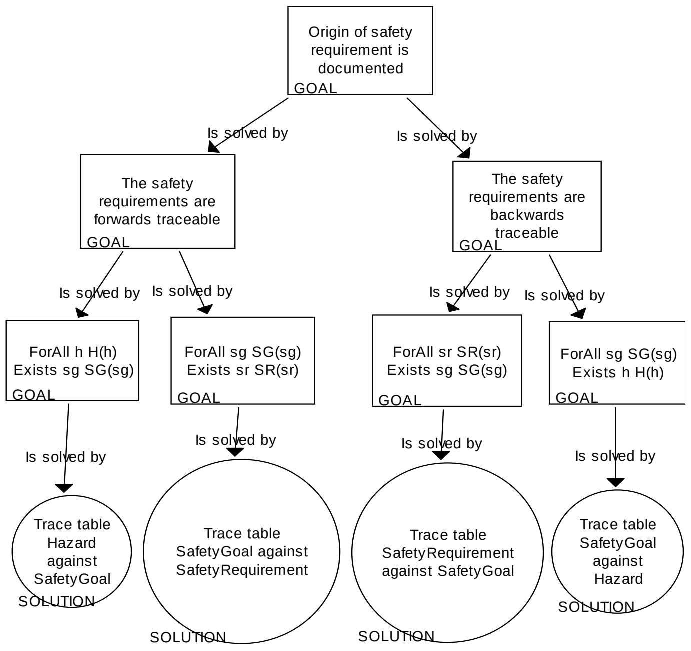

Origin of safety requirement is documented
GOAL
Is solved by
Is solved by
The safety requirements are forwards traceable
GOAL
The safety requirements are backwards traceable
GOAL
Is solved by
Is solved by
Is solved by
Is solved by
ForAll h H(h) Exists sg SG(sg)
GOAL
ForAll sg SG(sg) Exists sr SR(sr)
GOAL
ForAll sr SR(sr) Exists sg SG(sg)
GOAL
ForAll sg SG(sg) Exists h H(h)
GOAL
Is solved by
Is solved by
Is solved by
Is solved by
Trace table Hazard against SafetyGoal
SOLUTION
Trace table SafetyGoal against SafetyRequirement
SOLUTION
Trace table SafetyRequirement against SafetyGoal
SOLUTION
Trace table SafetyGoal against Hazard
SOLUTION

Fig. 3. Further decomposition of the goal "origin of safety requirement is documented".

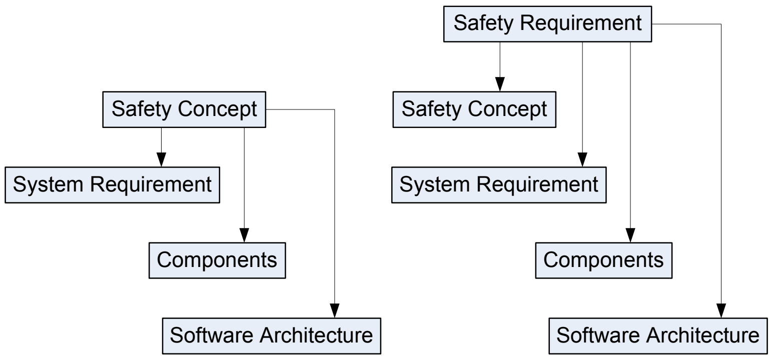


Fig. 4. Classes of development artifacts and their (forward) traceability relations that we implemented for the EBS case study.

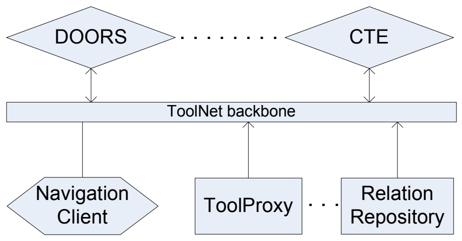


Fig. 5. ToolNet Architecture.

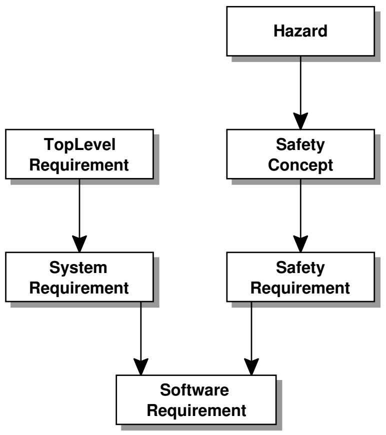


Fig. 6. Information model for the requirements engineering phase.

implementation and a number of test cases. In that way, engineers can build up a network of associations between the many different development objects, and thus, construct a trace table for an entire project. The tracing data is stored within ToolNet.

4.2 Traceability of the EBS-System

When we created the sample safety case for the emergency braking system, we could, to a large extent, reuse the structure of our generic safety case described in Sect. 2. The only difference is that we had to apply our claim "safety requirements are traceable" to all subsystems of the EBS, the hifi-system, the breaking system, the distance sensor and the control system. We could, therefore, extend the safety case displayed in Fig. 2 with a corresponding structure for each of the four sub-systems, leading to four separate sub-goals of the super-ordinate goal "the system requirements are traceable." This extended product-specific safety case is displayed in Fig. 7. We show the extended safety case for the control system only. Figure 4 shows the forward traceability links between the various types of artifacts that we implemented for the EBS system. Each artifact within the various tools had to be assigned one of these classes. We implemented that through adding a type attribute to the artifacts. For future projects, we propose to create an individual DOORS module for each of the artifact types in order to facilitate their management. After we had established all traceability links, the final step was the evaluation of our coverage criterion, that is, can all development artifacts be linked to artifacts in the respective other classes of artifacts, and can they be linked to an implementation? In other words, we had to verify the relations stated in our (forward) traceability claims (Figure 7).

4.3 Verification of the Traceability Links

The aim of a safety argument, following the goal structuring notation, is to associate goals with solutions. A goal is a claim corresponding to a required safety property of a system, and a solution is evidence supporting this claim. By associating every claim with a solution, we demonstrate that a system fulfills its safety requirements. Initially, we will have no associations stored in the trace tables, but, eventually, throughout the system development, the trace tables will be filled. ToolNet can be used to depict the information contained in the trace tables. An example is shown in Fig. 9.

Assessing to which extent the claims are proven by evidence in a large system such as the EBS control system, can be daunting. Identifying missing links manually is very tedious and time consuming, and the task must be repeated every time the system requirements are amended. This is why we are currently devising an automatic solution generator that can compile the various trace tables based on the development data found in ToolNet, and then make a traceability analysis. It works based on evaluating OCL constraints expressing the traceability relations. The outcome of such an analysis is twofold. First, we can highlight missing links in the desktop view of ToolNet for the developers (displayed in

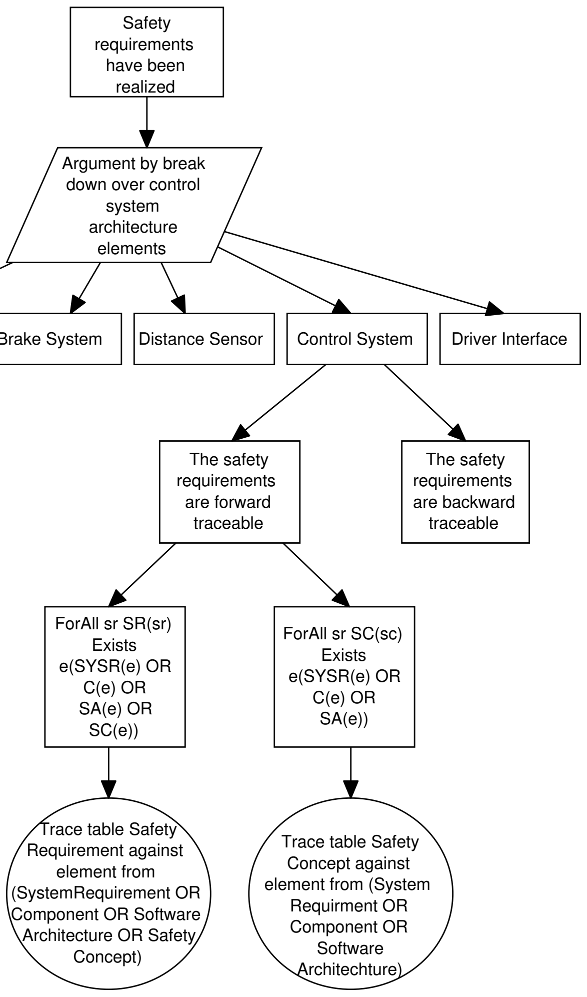
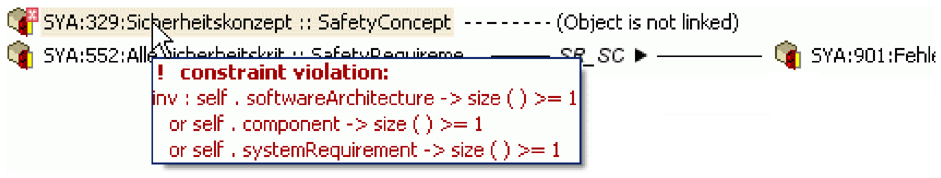

Safety requirements have been realized
Argument by break down over control system architecture elements
Hifi System
Brake System
Distance Sensor
Control System
Driver Interface
The safety requirements are forward traceable
The safety requirements are backward traceable
ForAll sr SR(sr) Exists e(SYSR(e) OR C(e) OR SA(e) OR SC(e))
ForAll sr SC(sc) Exists e(SYSR(e) OR C(e) OR SA(e))
Trace table Safety Requirement against element from (SystemRequirement OR Component OR Software Architecture OR Safety Concept)
Trace table Safety Concept against element from (System Requirment OR Component OR Software Architechture)

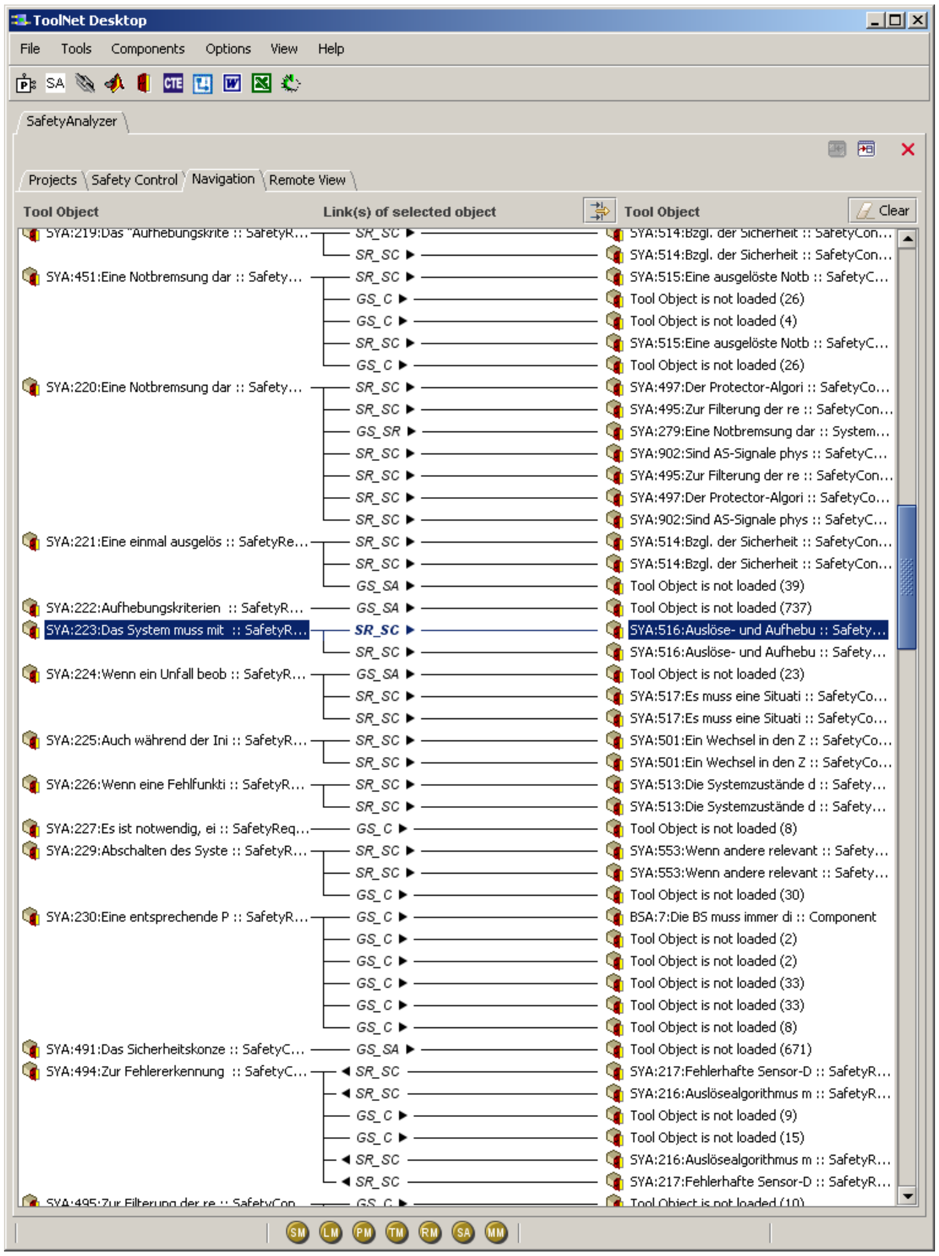
Fig. 7. Product-specific safety case for the emergency braking control system.


SYA:329:Sicherheitskonzept :: SafetyConcept -------- (Object is not linked)

SYA:552:Alle Sicherheitskrit :: SafetyRequireme —— SR_SC ▶ —— SYA:901:Fehl

! constraint violation:
inv : self . softwareArchitecture -> size ( ) >= 1
or self . component -> size ( ) >= 1
or self . systemRequirement -> size ( ) >= 1

Fig. 8. Evaluation of an OCL constraint in ToolNet.



Tool Object	Link(s) of selected object	Tool Object
SYA:219:Das "Aufhebungskrite :: SafetyR...	SR_SC ▶	SYA:514:Bzgl. der Sicherheit :: SafetyCon...
	SR_SC ▶	SYA:514:Bzgl. der Sicherheit :: SafetyCon...
SYA:451:Eine Notbremsung dar :: Safety...	SR_SC ▶	SYA:515:Eine ausgelöste Notb :: SafetyC...
	GS_C ▶	Tool Object is not loaded (26)
	GS_C ▶	Tool Object is not loaded (4)
	SR_SC ▶	SYA:515:Eine ausgelöste Notb :: SafetyC...
	GS_C ▶	Tool Object is not loaded (26)
SYA:220:Eine Notbremsung dar :: Safety...	SR_SC ▶	SYA:497:Der Protector-Algori :: SafetyCo...
	SR_SC ▶	SYA:495:Zur Filterung der re :: SafetyCon...
	GS_SR ▶	SYA:279:Eine Notbremsung dar :: System...
	SR_SC ▶	SYA:902:Sind AS-Signale phys :: SafetyC...
	SR_SC ▶	SYA:495:Zur Filterung der re :: SafetyCon...
	SR_SC ▶	SYA:497:Der Protector-Algori :: SafetyCo...
	SR_SC ▶	SYA:902:Sind AS-Signale phys :: SafetyC...
SYA:221:Eine einmal ausgelös :: SafetyRe...	SR_SC ▶	SYA:514:Bzgl. der Sicherheit :: SafetyCon...
	SR_SC ▶	SYA:514:Bzgl. der Sicherheit :: SafetyCon...
	GS_SA ▶	Tool Object is not loaded (39)
SYA:222:Aufhebungskriterien :: SafetyR...	GS_SA ▶	Tool Object is not loaded (737)
SYA:223:Das System muss mit :: SafetyR...	SR_SC ▶	SYA:516:Auslöse- und Aufhebu :: Safety...
	SR_SC ▶	SYA:516:Auslöse- und Aufhebu :: Safety...
SYA:224:Wenn ein Unfall beob :: SafetyR...	GS_SA ▶	Tool Object is not loaded (23)
	SR_SC ▶	SYA:517:Es muss eine Situati :: SafetyCo...
	SR_SC ▶	SYA:517:Es muss eine Situati :: SafetyCo...
SYA:225:Auch während der Ini :: SafetyR...	SR_SC ▶	SYA:501:Ein Wechsel in den Z :: SafetyCo...
	SR_SC ▶	SYA:501:Ein Wechsel in den Z :: SafetyCo...
SYA:226:Wenn eine Fehlfunkti :: SafetyR...	SR_SC ▶	SYA:513:Die Systemzustände d :: Safety...
	SR_SC ▶	SYA:513:Die Systemzustände d :: Safety...
SYA:227:Es ist notwendig, ei :: SafetyReq...	GS_C ▶	Tool Object is not loaded (8)
SYA:229:Abschalten des Syste :: SafetyR...	SR_SC ▶	SYA:553:Wenn andere relevant :: Safety...
	SR_SC ▶	SYA:553:Wenn andere relevant :: Safety...
	GS_C ▶	Tool Object is not loaded (30)
SYA:230:Eine entsprechende P :: SafetyR...	GS_C ▶	BSA:7:Die BS muss immer di :: Component
	GS_C ▶	Tool Object is not loaded (2)
	GS_C ▶	Tool Object is not loaded (2)
	GS_C ▶	Tool Object is not loaded (33)
	GS_C ▶	Tool Object is not loaded (33)
	GS_C ▶	Tool Object is not loaded (8)
SYA:491:Das Sicherheitskonze :: SafetyC...	GS_SA ▶	Tool Object is not loaded (671)
SYA:494:Zur Fehlererkennung :: SafetyC...	◀ SR_SC	SYA:217:Fehlerhafte Sensor-D :: SafetyR...
	◀ SR_SC	SYA:216:Auslösealgorithmus m :: SafetyR...
	GS_C ▶	Tool Object is not loaded (9)
	GS_C ▶	Tool Object is not loaded (15)
	◀ SR_SC	SYA:216:Auslösealgorithmus m :: SafetyR...
	◀ SR_SC	SYA:217:Fehlerhafte Sensor-D :: SafetyR...
SYA:495:Zur Filterung der re :: SafetyCon...	GS_C ▶	Tool Object is not loaded (10)

Fig. 9. ToolNet trace table highlighting a trace between a safety requirement and safety concept.

Fig. 8), and, second, we can generate reports about the safety status of an entire system, i.e. for management or, eventually, the certification authority.

5 Summary, Conclusions and Future Work

The upcoming safety standard ISO/WD 26262 for the automotive industry, introduces the concept of a safety case, which communicates a clear, comprehensive and defensible argument that a system is acceptably safe. In the future, it will be required by safety inspectors in order to assess how and to which extent all safety-related issues have been addressed and treated by the vendor of a safety-critical system in the automotive domain.

In this paper, we demonstrated how a safety case can be constructed based on the goal structuring notation that proposes to create a defensible argument out of goals that are decomposed recursively into subgoals until a subgoal can be proven by evidence. The evidence is provided by documents addressing the safety issues. The standard prescribes that part of the safety case should demonstrate that the origin, realization and proof for every requirement is clearly documented. In other words, all requirements should be traceable to their respective implementations (forward and backward).

We developed a generic safety case that could be applied to emergency braking systems like the Active Brake Assist for lorries of the brand Mercedes-Benz. This safety case may act as a template, or a reference realization for other systems in the automotive domain. The greatest challenge was to incorporate the traceability features of our safety case into the existing development process that employs a number of different tools. We solved that through adding trace information to ToolNet, a framework that integrates many different software engineering tools. Within ToolNet, we can now navigate along the traceability paths and assess which safety requirements have been treated sufficiently.

Current work is now focused on the development of an automatic solution generator that will compile the trace tables required for the safety argument automatically. In the future, this generator can be used to provide the current safety status of a project on the punch of a button, i.e. for project management, display a colored safety argument, with green and red indicating the safety status of system parts, and, eventually, compile the safety reports for the inspection agency.

References

1. F. Altheide, S. Dörfel, H. Dörr and J. Kanzleiter. An Architecture for a Sustainable Tool Integration. Workshop on Tool Integration in System Development, pp. 29–32, Helsinki, Finland, September 2003.
2. Automotive Standards Committee of the German Institute for Standardization. ISO/WD 26262: Road Vehicles – Functional Safety. Preparatory Working Draft, Technical Report, October 2005.
3. O. Bridal and others. Deliverable D3.1 Part 1 Appendix E: Safety Case, Version 1.1. Technical Report, EASIS Consortium (www.easis-online.org), February 2006.

4. Intl. Electrotechnical Commission. IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Technical Report, April 1999.
5. G. John, M. Hoffmann, and M. Weber. EADS-Methodenrichtlinie zur Traceability zwischen Anforderungen und Entwurfsobjekten. Technical Report RM-008, DaimlerChrysler AG, November 2000.
6. T.P. Kelly and J. McDermid. Safety Case Construction and Reuse using Patterns. In: Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP'97), Springer-Verlag, September 1997.
7. T.P. Kelly. Arguing Safety: A Systematic Approach to Managing Safety Cases. PhD Thesis, University of York, UK, September 1998.
8. E. Lehmann. Time Partition Testing: A Method for Testing Dynamic Functional Behaviour. Proceedings of TEST2000, London, Great Britain, May 2000.
9. E. Lehmann. Time Partition Testing. PhD Thesis, Technical University of Berlin, February 2004.
10. N.G. Leveson. Safeware: System Safety and Computers. Addison-Wesley, Boston, MA, 1995.
11. Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung. V-Modell©XT, 2004 (http://www.kbst.bund.de).
12. N. Storey. Safety Critical Computer Systems. Addison-Wesley, 1996.
13. S.E. Toulmin. The Uses of Argument. Cambridge University Press, 1958.
14. R. A. Weaver. The Safety of Software – Constructing and Assuring Arguments. DPhil Thesis, Department of Computer Science, University of York, UK, 2003.
15. R. Weaver, G. Despotou, T. Kelly and J. McDermid. Combining Software Evidence: Arguments and Assurance. Proceedings of the 2005 workshop on Realising evidence-based software engineering, pp. 1–7, St. Louis, Missouri, 2005.

TUD-SERG-2007-008
ISSN 1872-5392

SERG