

A Safety Case Approach to Assuring Configurable Architectures of Safety-Critical Product Lines

Ibrahim Habli and Tim Kelly

Department of Computer Science, University of York, United Kingdom
{Ibrahim.Habli, Tim.Kelly}@cs.york.ac.uk

Abstract. Companies are increasingly adopting a product-line approach to the development of safety-critical systems. A product line offers large-scale reuse by exploiting common features and assets shared by systems within a specific domain. In this paper, we discuss the challenges of justifying the safety of architectural configurations and variation when developing product-line safety cases. We then address these challenges by defining an approach to developing product-line safety cases using the patterns and modular extensions of the Goal Structuring Notation (GSN). In this approach, we use the GSN patterns extension for explicitly capturing safety case variations and tracing these variations to their extrinsic source in the architectural model. Further, we use the GSN modular extension to organise the safety case into core and variable argument modules which are loosely coupled by means of argument contracts. We demonstrate this approach in a case study based on a product line of aero-engine control systems.

Keywords: Safety Cases, Architectures, Product Lines, Variation Management.

1 Introduction

To reduce the engineering costs of safety-critical systems, companies are increasingly adopting a product-line approach which offers large-scale reuse by exploiting common features and assets shared by systems within a specific domain. In particular, the safety case in a safety-critical product line is a valuable asset which should be systematically documented, reused and maintained. Otherwise, the value of a safety-critical product line can be easily undermined if the safety case is developed from scratch, or in an ad-hoc manner, for each product within the product line. Given that products in a product line share most of their functional features, components, failure modes and risk mitigation measures, it is reasonable to expect that these products also share strategies which can be used to argue why they are acceptably safe to operate within certain environments. For example, if products derived from a product-line share a set of functional configurations, which pose similar risks managed using common risk-mitigation measures, it would be sensible to expect that the safety case for these products share a set of core (i.e. common) argumentation strategies too.

However, like most reusable product-line assets, the challenges do not lie simply in exploiting and managing commonalities. Rather, the key challenges often lie in the

management of the way in which assets may vary, according to predefined architectural constraints. In other words, it is important to identify and manage both the *common* and the *variable* structures of the product-line safety case and the ways in which these structures may be reused and composed in order to derive a compelling, comprehensible and traceable safety case for each individual product.

In this paper, we describe and evaluate a safety case approach to justifying the architectures of safety-critical product lines. We start by introducing product-lines. We then discuss the challenges of managing variations within product-line safety cases and how they can correspond to architectural configurations. We then present an approach to developing product-line safety cases using the patterns and modular extensions of the Goal Structuring Notation (GSN) [3]. Finally, this approach is evaluated in an aerospace case study.

2 Product-Line Engineering

A product line comprises a configurable architecture and a set of reusable core assets. Products can be derived from the product-line architecture and core assets based on a predefined process that manages and controls permitted configurations and *variations*. These variations in a product line are not an indication that the development is unstable. Instead, they indicate that differences between products are identified, analysed and controlled. Product-line development is an integrated approach in that any development or assessment artefacts, particularly early-lifecycle artefacts such as requirements and analysis models, can be reused as long as they adhere to the context, architectural constraints and variation rules defined in the product line. To reap the benefits of product lines, the development should be carried out according to defined processes and within certain technical and business constraints. The following definition by the Software Engineering Institute (SEI) provides a comprehensive description of product lines [1]: “A *software product line* is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”. What distinguishes the SEI’s definition from many other accounts of product lines is that it explicitly addresses the need to develop the product-line core assets and derive new products “*in a prescribed way*”, i.e. according to defined processes. There are two essential processes in any product line, namely:

- Product-line establishment process (domain engineering)
- Product derivation process (application engineering)

In the domain engineering process, the product-line scope is defined, permitted variations are identified and the reference architecture and core assets are developed [2]. Subsequently, in the product derivation process, products within the scope of the product line are developed from the reusable assets, according to permitted variations.

3 Variation in Product-Line Safety Cases

The safety case of a product line is not immune from the impact of environmental and system variations. Ignoring or underestimating the impact of these variations may

seriously weaken confidence in the product-line safety case. For example, any mismatches between the product line’s permitted architectural options and the assumed architectural options considered in the safety case can invalidate certain safety case claims and evidence. Fig. 2 shows an example safety argument, represented in GSN, for a safety-critical system. GSN represents safety arguments in terms of basic elements such as goals, solutions, and strategies (Fig. 1). Arguments are created in GSN by linking these elements using two main relationships, ‘supported by’ and ‘in context of’ to form a goal structure. The principal purpose of any goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach and the context in which goals are stated (e.g. the system scope or operational role). GSN has been adopted by a growing number of companies within safety-critical industries (such as aerospace, railways and defence) for the presentation of safety arguments within safety cases. The key benefit experienced by those companies adopting GSN is that it improves the comprehension of the safety argument amongst all of the key project stakeholders (i.e. system developers, safety engineers, independent assessors and certification authorities). In turn, this has improved the quality of the debate and discussion amongst the stakeholders and has reduced the time taken to reach agreement on the argument approaches being adopted.

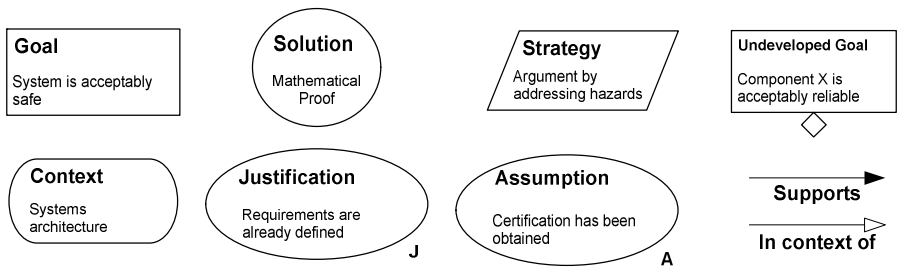


Fig. 1. Core Elements of GSN

The style of the reasoning in the argument in Fig. 2 is hazard/risk directed. Briefly, the top-level claim, that the system is safe to operate within the specified environment (*SysSafe*), is substantiated by arguing that the risk posed by the identified hazards is acceptable. The top-level claim is made in the context of some definition of the system and its environment (*CSys* and *CEnvironment*). Next, the argument considers the acceptability of the risk posed by each identified hazard in the context of specific risk tolerability criteria (*CTolerabilityCri*). Finally, the risk acceptability of each hazard is supported by appealing to the deployment of sufficient risk-mitigation measures (*CRiskMitig*) which are implemented by a number of components and interactions (*CompSafe1*, *CompSafe2*, *CompSafe3* and *ComInter*).

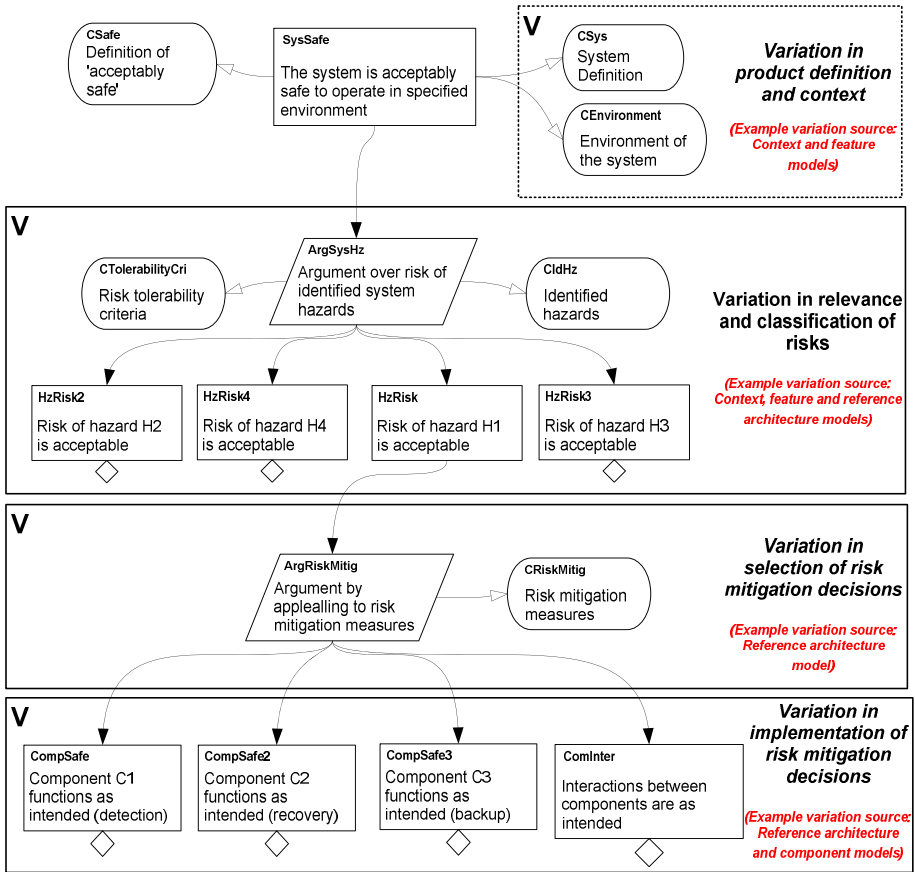


Fig. 2. Impact of Product-Line Variation on the Safety Argument

Product-line variation can potentially affect the argument in Fig. 2 in any of the following ways:

- Context and system definition:** In a product line, the context and feature models define how products, derived from the product line, vary from one another. To this end, the safety argument context elements (*CSys* and *CEnvironment*) have to vary to correspond accurately to the way in which each product is configured, based on the context and feature models.
- Hazard identification and risk assessment:** Depending on the features selected and the environmental conditions assumed for each derived product, the way in which the argument considers the risk posed by hazards could vary (*ArgSysHz*). Not all hazards may be relevant to all product configurations. Also, the risk assessment results for each applicable hazard may vary due to some variable external or system features. Further, the risk tolerability criteria may vary across products if these products are deployed in different environments with different certification requirements, e.g. civil vs. military applications.

- **Risk mitigation measures:** The product-line reference architecture may offer a number of options for mitigating certain risks (*CRiskMitig*). Not all these options may be selected for each product. Consequently, the product-line safety argument needs to accommodate variation concerning how each derived product may mitigate its associated risks.
- **Implementation of risk mitigation measures:** Two derived products may share the same risk-mitigation strategy, but may vary in how they implement such a strategy. For example, two products may adopt a strategy for fault-detection by means of monitoring. However, the product-line reference architecture may offer two or more alternatives for implementing a monitor, e.g. either in software or in hardware. The product-line safety case should therefore explicitly address possible variation in how risk mitigation measures may be implemented.

The above example safety case variations show the importance of developing the product-line safety case in such a way that permitted safety case variations are explicitly defined and traced to other variations in the environment and architectural configuration of a derived product.

4 Managing Product-Line Safety Cases Using GSN

In this paper, we address the challenges presented in the previous section by defining an approach to developing product-line safety cases using the patterns and modular extensions of GSN. The patterns and modular extensions of GSN are introduced in the next two sections, followed by a detailed description and analysis of how they can be used to create and manage product-line safety cases.

4.1 GSN Patterns Extension

Based in part on the principles of the work on design patterns by Christopher Alexander [9] and the ‘Gang of Four’ (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides) [10], the concept of a safety case pattern was developed as “*a means of documenting and reusing successful safety argument structures*” [3]. A safety case pattern captures argument structures observed to be common in forming the backbone of certain safety cases in a particular domain or across different domains. To create safety case patterns, the core of GSN was extended by adding the following types of abstraction (Fig. 3):

- **Structural Abstraction** – supporting generalised n-ary, optional and alternative relationships between GSN elements
- **Entity Abstraction** – supporting generalisation/specialisation of GSN elements

Safety case patterns are created in GSN by abstracting the details of commonly-used safety argument structures. These patterns describe a successful and a proven style of argumentation rather than a concrete argument for a particular system.

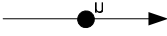
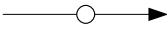




Multiplicity Extensions	
	A solid ball is the symbol for many (meaning zero or more). The label next to the ball indicates the cardinality of the relationship
	A hollow ball indicates "optional" (meaning zero or one)
	A line without multiplicity symbols indicates a one to one relationship (as in conventional GSN)
Optionality Extension	
	This symbol is defined for use over the existing relation types. Choice can be used to denote possible alternatives in satisfying a relationship. It can represent a 1-of-n and m-of-n selection.
Entity Abstraction Extensions	
 Uninstantiated Entity	Entity remains to be instantiated i.e. at some later stage the 'abstract' entity needs to be replaced (instantiated) with a more concrete instance.
 Undeveloped Entity	This placeholder denotes that the attached entity requires further development, i.e. at some later stage the entity needs to be (hierarchically) decomposed and further support by sub-entities.

Fig. 3. GSN Pattern Notation – Entity Abstraction Extensions [3]

4.2 GSN Modular Extension

To support the certification of highly-integrated, modular and reconfigurable systems such as Integrated Modular Avionics (IMA), the core of GSN was extended with modular features [4]. These features support the development of *modular and compositional safety cases*. One fundamental objective of modular and compositional safety cases is to reduce the amount of effort needed for the reassessment of a safety case after system changes or reconfiguration. Rather than considering the safety case as a *single monolithic* structure, it can be viewed as a set of well-defined and scoped modules, the composition of which defines the system safety case. Each argument module is specified by an interface, comprising [4]:

1. Goals addressed by the module
2. Evidence presented within the module
3. Context defined within the module
4. Arguments requiring support from other modules

Inter-module dependencies:

5. Reliance on goals addressed in other modules
6. Reliance on evidence presented within other modules
7. Reliance on context defined within other modules

Elements 5, 6 and 7 are called *Away Goals*, *Away Context* and *Away Solutions* (Fig. 4). In particular, an *Away Goal* is a goal reference which is used to support, or provide contextual backing for, an argument presented in one module. However, the argument supporting that goal is presented in another module (hence creating interdependencies between the safety case modules).

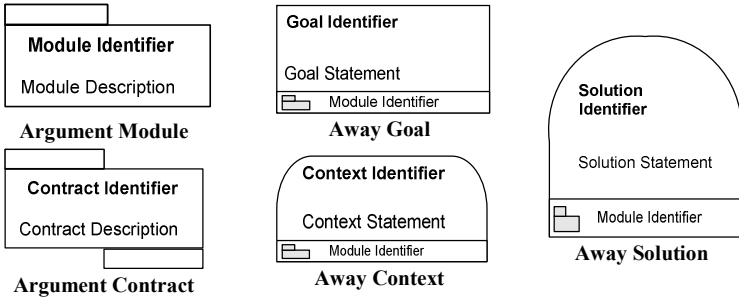


Fig. 4. GSN Modular Notation [4] [5]

Further, in order to promote loose coupling and therefore minimise the impact of change between interrelated safety case modules, the concept of an argument *contract* was created in [4], and later refined in [5] [7] (Fig. 4). Argument contracts preserve the overall integrity of the modular safety case when the internal details of one or more argument modules are modified. This is mainly because these contracts are specified using the interfaces of the interrelated argument modules rather than using the internal details of these modules (hence protecting interdependent modules from changes to the internal details of the arguments contained in these modules). Essentially, a safety case contract captures a ‘rely-guarantee’ relationship between two argument modules. In a contract, items 4 to 7 in a module interface define the ‘rely’ conditions whereas item 1 defines the guarantee conditions (items 2 and 3 should hold during the composition of the two or more argument modules).

5 Capturing Safety Case Variations Using the GSN Patterns Extension

In this section, we show how the GSN patterns extension can be used to capture variation in the product-line safety case. A product-line safety case comprises reusable argument structures and evidence used as the basis for the definition of the safety case for each product derived from the product-line assets. Some of these argument structures and evidence are shared between all product safety cases, and therefore are *core*, while others differ from one product safety case to another, and therefore are *variable*. Generally, there are two types of product-line safety case variation: *intrinsic* and *extrinsic variation*.

Intrinsic variation exists whenever there is more than one argumentation style to support the safety claims of a *particular* product-line instance. *Extrinsic* variation, in contrast, is more peculiar to product-line safety cases. The source of this type of variation is not the product-line safety case itself but rather the reusable assets referenced in the safety case from product-line models such as the feature and reference architectural models. This is because many of these assets are expected to vary in how they are developed, configured and composed. This variation may change the contribution of these assets to safety and therefore may change the way in which the safety of the system is justified in the safety case. To this end, it is important that extrinsic

variation in a product-line safety case be explicitly linked to, and constrained by, its source, be it a variation in the context model, the feature model or the reference architecture model.

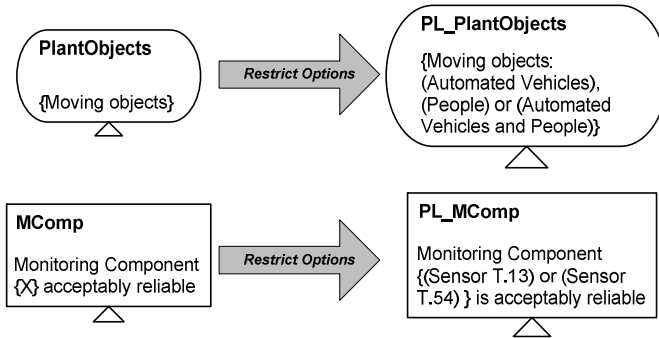


Fig. 5. Defining Safety Case Variations using Entity Abstraction Extensions

As discussed in Section 4.1, there are two types of abstraction supported in the GSN patterns extension: *Entity Abstraction* (supporting generalisation/specialisation) and *Structural Abstraction* (supporting optionality and multiplicity). Here, we use *Entity Abstraction* to capture and restrict variation in GSN elements (goals, strategies, context, assumptions, justifications and evidence). On the other hand, we use *Structural Abstraction* to capture and restrict variation in the relationships between the GSN elements (*supported by* and *in the context of* relationships). The left hand side of Fig. 5 shows example un-instantiated GSN elements (types of *Entity Abstraction*). The items in the curly brackets represent types of un-instantiated information (*‘Moving objects’*, *‘X’* and *‘Coverage Analysis Data’*). These items need to be instantiated before they can be used as part of a concrete safety argument. When representing extrinsic variation in the product-line safety case, the un-instantiated GSN elements should be explicitly associated with their source of variation. As shown on right hand side of Fig. 5, *‘Moving objects’* are limited to *‘Automated Vehicles’*, *‘People’* or *‘Automated Vehicles and People’*, which are objects defined in the product-line context model. In other words, these are the moving objects which are assumed to be within the scope of the product-line and, as such, the safety argument is only valid in the context of one of these sets of moving objects. Similarly for the *‘Monitoring Component {X}’* in the *‘MComp’* Goal, two variants of this component exist in the *‘PL_MComp’* Goal: *‘Sensor T.13’* and *‘Sensor T.54’*. These variants are alternative design components offered by the reference architecture. Variations in the safety case do not only affect the GSN elements, but they also affect the way in which these elements are connected. To capture and restrict variation in the relationships between the GSN elements of a product-line safety case, we use the *Multiplicity* and *Optionality* symbols which are part of the GSN patterns extension (*Structural Abstraction*).

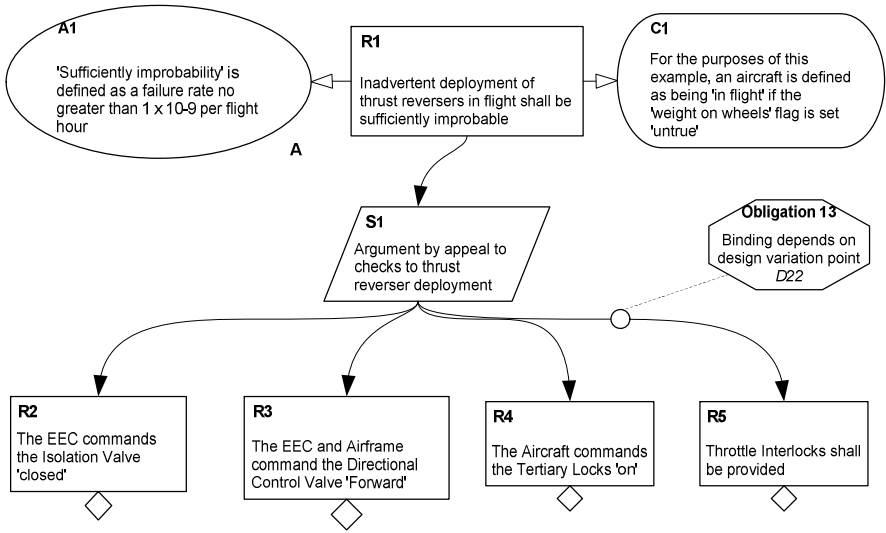


Fig. 6. Thrust Reverse Deployment Protection Argument (adapted from [6])

Fig. 6 illustrates how variation in the way in which GSN elements may be connected can be captured and traced to the product-line models. For each multiplicity and optionality, we attach a GSN element called ‘Obligation’ (octagon symbol), which describes the basis on which options and alternatives may be selected. More specifically, each obligation element that is based on extrinsic variations should be traced to one or more product-line variations in the context, feature or reference architectural models. The argument in Fig. 6 is an adaptation of the thrust reverse deployment defined in [6]. In order to prevent an inadvertent deployment of thrust reversers in flight, three core checks should be performed: ‘The EEC commands the Isolation Valve closed’ (R2), ‘The EEC and Airframe command the Directional Control Valve Forward’ (R3) and ‘The Aircraft commands the Tertiary Locks on’ (R4). The use of throttle interlocks on certain engines is optional (labelled in Fig. 6 as design variation point ‘D22’). In other words, the use of throttle interlocks is only required by some airframe manufacturers. As such, ‘Obligation 13’ is used in Fig. 6 to indicate that ‘R5’ is optional and its selection depends on the instantiation of the design variation point ‘D22’ (the design assumption here is that, regardless of whether ‘D22’ is instantiated or not, as a minimum requirement, the overall failure rate is not greater than 10^{-9} per flight hour).

6 Composing Product-Line Safety Cases Using the GSN Modular Extension

The product-line safety case should be highly reconfigurable in order to support the derivation of a safety case for each product developed from the product-line assets. In the previous section, we described how to embed variations into the product-line safety case using the GSN patterns extension. These variations need to be instantiated

for each derived *product* safety case. However, a key constraint is to instantiate these variations in a way that minimises the effort needed to assess the suitability of a derived safety case used for the assurance of a particular product. In this section, we propose that the effort needed for the assessment of each product safety case could be reduced by the adoption of the concept of modular and compositional safety cases. Modularity is a proven design technique for improving the flexibility of software systems, particularly through promoting loose coupling and high cohesion between modules interacting through a stable set of interfaces. These interfaces hide the details of each module, exposing only information required for module integration, hence reducing unnecessary dependencies between interacting modules.

In this section, we use the GSN modular extension, reviewed in Section 4.2, to structure the safety case into core and variable *argument modules*. These argument modules are defined and composed in a way that protects core argument modules from the permitted variation in the optional argument modules by means of predefined argument *contracts*. In order to improve the process of deriving a safety case instance from the product-line safety case, we establish a clear and traceable mapping between the structures of the product-line safety argument and the structures of the product-line feature and reference architecture models. Further, it is important to define a product-line safety case in such a way that variations, e.g. options and choices, can be added or removed with little impact on the overall structure of the product-line safety case and its instances. This can be achieved by defining the safety case in terms of loosely-coupled argument modules. One mechanism for defining loosely-coupled argument modules is to reduce '*hardwired*' dependencies between the argument modules by means of predefined argument *contracts* [5]. These argument *contracts* can serve as a mediator or a proxy between interrelated argument modules, isolating the way in which one argument module is '*supported*', or '*contextually-backed*', by another argument module. More specifically, argument contracts can be used to contain the impact of variation in one argument module and prevent it from propagating to other argument modules. The way in which argument contracts can be used to contain the impact of product-line variations within a safety case is illustrated in Fig. 7. The top-left hand side of Fig. 7 shows an argument structure in which the argument module '*Function X*' is supported by either the argument module '*Redundancy*' or the argument module '*Monitoring*', i.e. variation in the form of alternativity. Here, because these two argument modules are *directly* connected to the argument module '*Function X*', a change to these modules may propagate to the argument module '*Function X*' and vice versa. The top right hand side of Fig. 7 shows how the insertion of an argument contract between the argument module '*Function X*' and the argument modules '*Redundancy*' and '*Monitoring*' creates a 'buffer zone' that masks the way in which the argument module '*Function X*' is supported. For example, if a new optional argument module is later added to the product-line safety case to support the argument module '*Function X*', any required change should be contained within the argument contract and should not propagate to the argument module '*Function X*'. Fig. 7 also shows internal details of the argument contract, showing how the public goal '*SysDesign*', which is part of the public interface of the argument module '*Function X*', is supported by a strategy which is based on either a claim of adequate redundancy, provided by the argument module '*Redundancy*', or a claim of

adequate monitoring, provided by the argument module ‘*Monitoring*’. In other words, within the scope of the argument module ‘*Function X*’, the actual way in which the goal ‘*SysDesign*’ is supported is unknown but entrusted to the argument contract, i.e. breaking any direct coupling between the argument module ‘*Function X*’ and the argument modules ‘*Redundancy*’ and ‘*Monitoring*’.

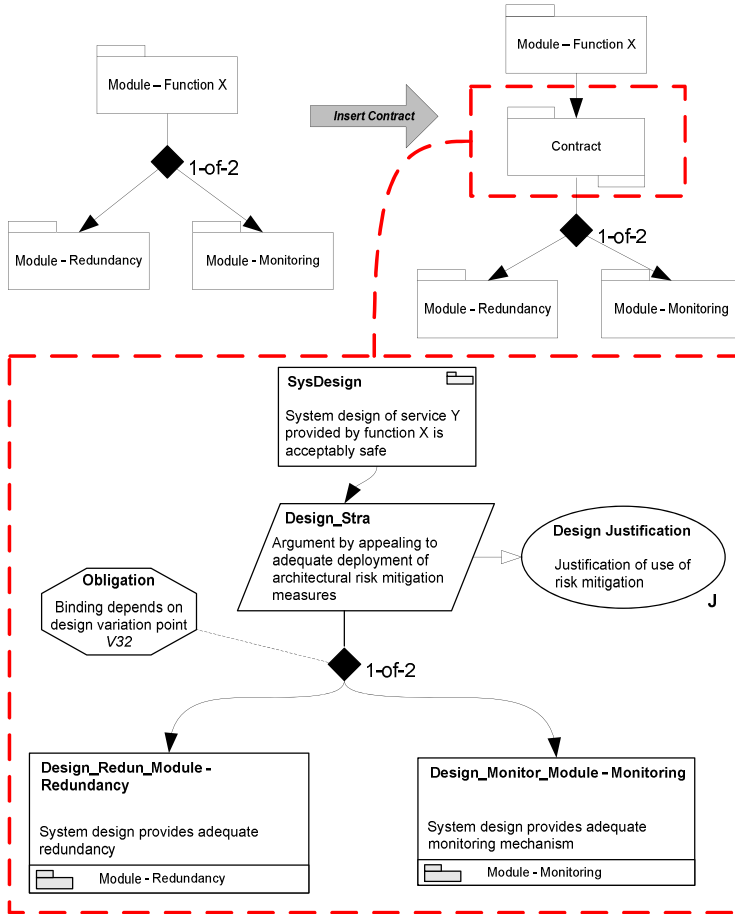


Fig. 7. Variations within an Argument Contract

In short, modularity in the product-line safety case can support the encapsulation of the arguments behind product-line functions and components in reusable and self-contained modules. Equally importantly, dependencies between these modules should be minimised in order to contain potential ripple-effects resulting from changes in the options and choices provided by the product line.

7 Case Study – Assuring Safety of Aero-Engine Sensors

We have demonstrated how a product-line safety case can be constructed using the patterns and modular extensions of GSN by means of case studies from the aerospace and automotive domains. The case study presented in this section is partly based on an aero-engine control system described in [8].

7.1 System Overview

Modern aircraft gas turbine engines are controlled by a Full Authority Digital Electronics Control (FADEC) system. This control system is a high-integrity computer system that controls and monitors the operation of the engine. Within aero-engine product lines, not only may these product lines vary at the engine level (e.g. optional/alternative physical and interface characteristics), but also they can vary at the control system level (e.g. optional/alternative functional and technological characteristics). The control system variants can be considered as part of a product line embedded within the larger engine product line. In this case study, we focus on the commonalities and variabilities within an engine control system product line.

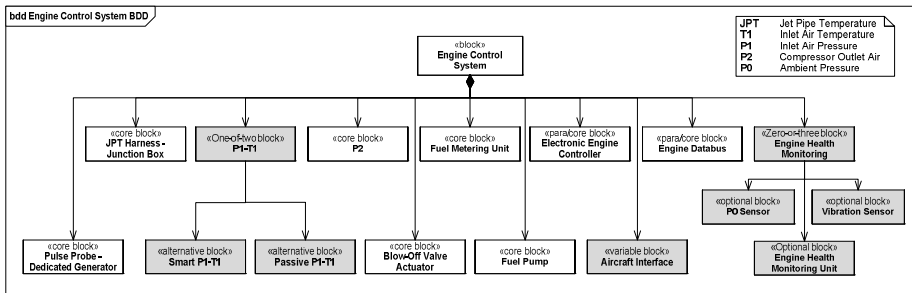


Fig. 8. Block Definition Diagram

A depiction of the reference architecture of the control system product line is shown in Fig. 8 and Fig. 9 (specified in the Systems Modelling Language (SysML) [11]). Here we focus on three different types of variations within this architecture. Firstly, the reference architecture provides two alternative means for measuring the engine’s inlet air temperature (T1) and inlet air pressure (P1) using either passive or smart sensors. The key differences between these two types of sensor are as follows. The reliability of T1 and P1 measurements is higher when smart sensor are used. Also, signal conditioning, signal selection, fault detection and fault accommodation can be carried out locally by the smart sensors T1 and P1 (as opposed to being fully performed by the Electronic Engine Controller when passive sensors are used). Further, smart sensors can communicate with the Electronic Engine Controller via the Engine Databus, and not through a dedicated pipe, and therefore can simplify “*wiring and connections, piping and reduce weight*” [8]. The second key architectural variation lies in the Aircraft Interface block. This variation supports the realisation of various interface requirements requested by different customers (e.g. special interface

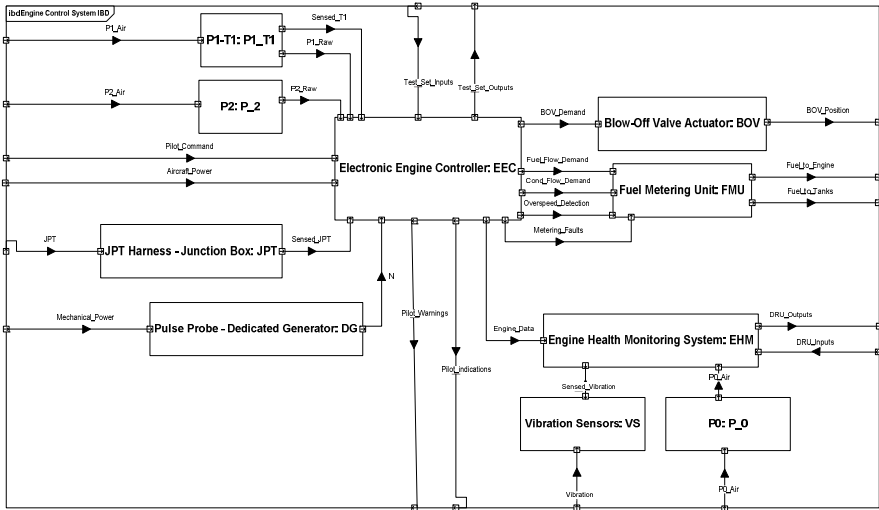


Fig. 9. Internal Block Diagram

requirements by aircraft manufacturers). Finally, the reference architecture offers engine health monitoring as an optional block. This block comprises an Engine Health Monitoring Unit and P0 (ambient pressure) and vibration sensors.

7.2 Safety Case

In this case study, we developed a safety case which considered six different hazards. In this paper, due to page constraint, we only consider the argument over the risk of hazardous overspeed. Fig. 10 shows the argument over the mitigation of the faults contributing to hazardous overspeed, e.g. faults in fuel flow, speed measurements and T1/P1 measurements. In particular, the ‘T1/P1’ argument module, referenced in the above argument concerning the contribution of the T1/P1 measurements, requires further instantiation as it embodies variations which are associated with the ability to choose between the deployment of either smart or passive sensors.

The argument within the ‘T1/P1’ module is shown in Fig. 11. In this argument, the top-level claim is made in the context of either passive or smart sensors. This choice is captured using the GSN choice pattern symbol. This choice is associated with ‘*Obligation 2*’ which references design variation concerning the selection of either passive or smart sensors in the reference architecture. Here, regardless of the selected design choice, the reasoning strategy is common – arguing over the T1/P1 reading, conditioning and transmission. The claims concerning the T1/P1 reading, conditioning and transmission are supported by three argument contracts, each of which requires instantiation based on the T1/P1 design choice. The importance of these argument contracts is that they defer the instantiation of variations associated with the T1/P1 design choice and therefore preserve the overall integrity of the T1/P1 argument, i.e. localising the impact of the T1/P1 design choice by hiding this impact behind the contracts.

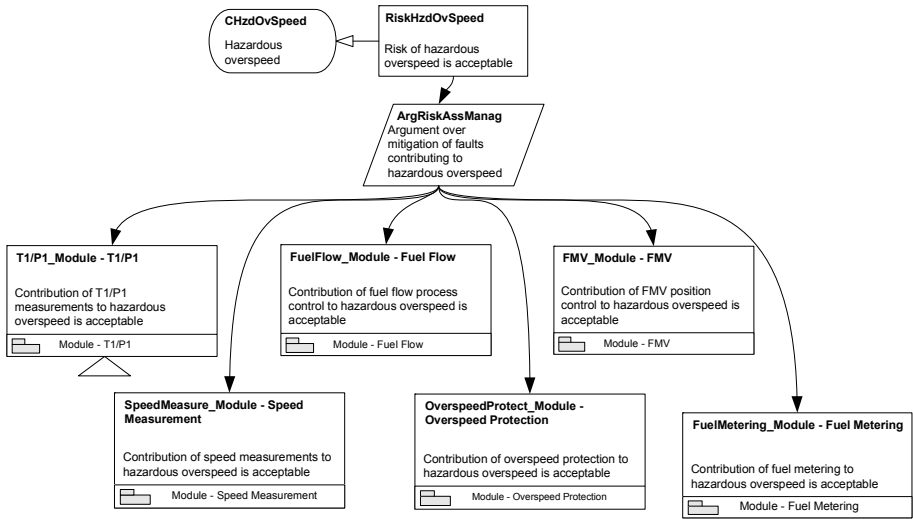


Fig. 10. Argument Module – Hzd Overspeed

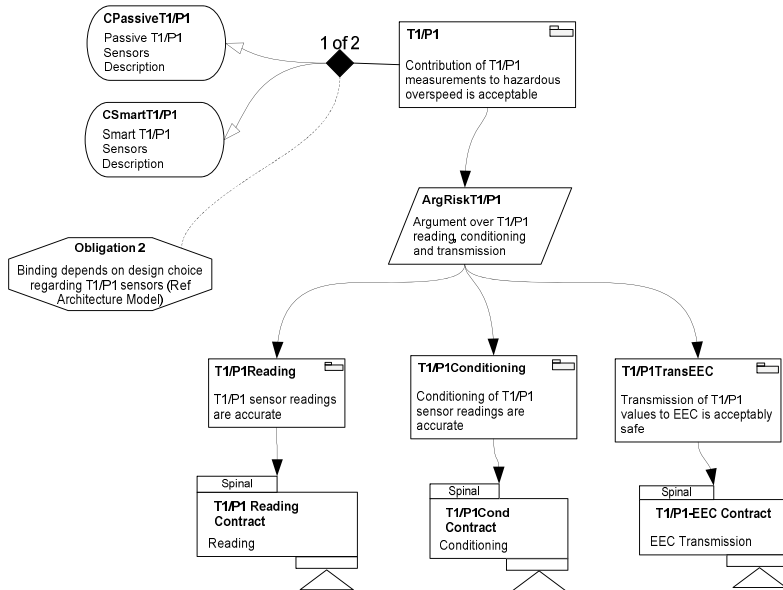


Fig. 11. Argument Module - T1/P1

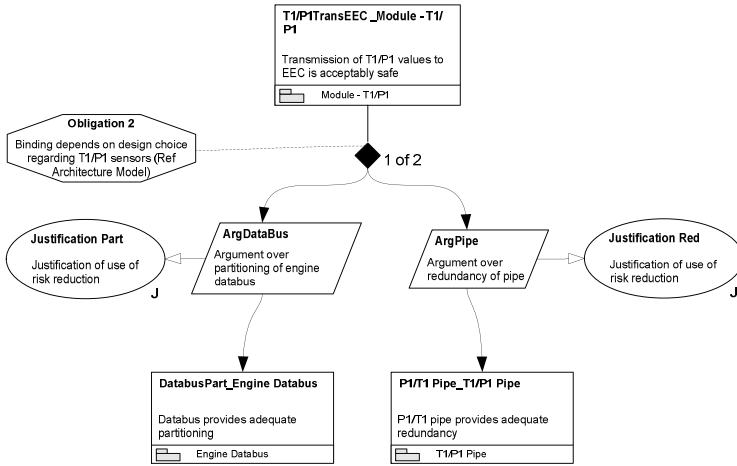


Fig. 12. T1/P1-EEC Contract

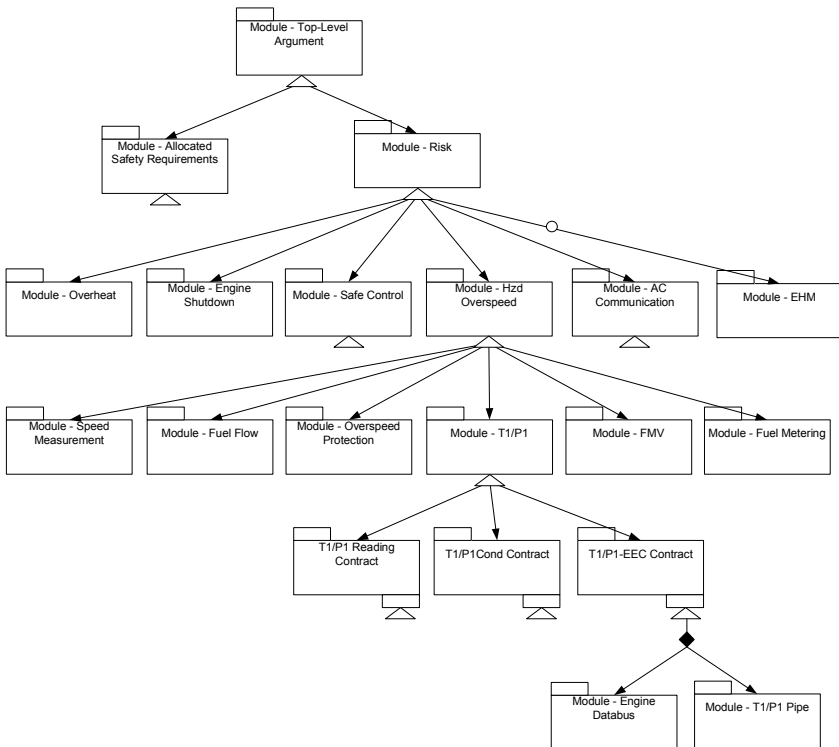


Fig. 13. Safety Case - Module Viewpoint

Fig. 12 shows the argument presented within the ‘*T1/P1-EEC*’ contract. The claim that the transmission of T1/P1 values to EEC is acceptably safe is either supported by arguing over Engine Databus partitioning if smart sensors are chosen or arguing over pipe redundancy if passive sensors are chosen. The choice is made based on the instantiated design variation in the reference architecture regarding the T1/P1 sensor alternatives. Subsequently, depending on the T1/P1 sensor choice, the instantiated contract relies either on the ‘*Engine Databus*’ argument module or the ‘*T1/P1 Pipe*’ argument module to fulfill the guarantee in supporting the claim that the transmission of T1/P1 values to EEC is acceptably safe (‘*T1/P1TransEEC*’).

Finally, in order to show a modular viewpoint of the safety case developed in this case study, we have generated the safety case model shown in Fig. 13. This viewpoint represents a high-level depiction of the top-level safety case in terms of the safety case’s argument modules and contracts. This viewpoint also shows the distinction between *core* argument modules such as the ‘*Engine Shutdown*’ and ‘*Fuel Metering*’ modules and *variable* argument modules such as the ‘*AC Communication*’ and ‘*T1/P1*’ modules (variability indicated using the un-instantiation symbol). Further, this viewpoint reveals some ‘structural stability’ in the way in which the safety case is organised where the higher-level modules address individual hazards and lower-level modules address how the architectural blocks and properties contribute to these hazards. Also, at these two levels, the impact of product-line variation is contained within the argument modules and contracts.

8 Related Work

Safety standards acknowledge the economic need to employ previously developed systems, functions and components [12] [13] [14] [15]. In civil aerospace for example, systems may be reused across different aircraft types, without the need for additional assessment, provided that evidence of similar design, installation, application and operation can be produced [15]. Otherwise, the safety assessment process should be performed to examine the impact of the reusable systems on the aircraft functions. Also in civil aerospace, particularly for airborne software, the American Federal Aviation Administration (FAA) created an Advisory Circular (AC), offering means to satisfy the requirements of the aerospace software guidance DO-178B regarding the use of reusable software components [16].

Generally, most standards place additional constraints on the way in which reusable components are developed, verified, integrated, and maintained. The goal of such constraints is to ensure that the safety of the overall system is not compromised as a result of incorporating reusable components. For example, the flexibility developed into reusable software components may result in unreachable code, and therefore, some standards require that system developers demonstrate that the risk of leaving unreachable code is less than the risk of removing it [13]. In extreme cases where a reusable component was not developed to the safety integrity level of a new system, reverse engineering and the application of more rigorous techniques may be required (in addition to the generation of a new safety argument). In short, although most safety standards do not address product-lines explicitly, they take into account the need to employ previously developed artefacts. They often require additional activities to assess the impact of a reusable component or function on the safety of the overall system.

Research in the field of product-line safety has focused on adapting traditional safety analysis techniques, such as Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA), to suit product-line processes. The majority of this work has been produced within the Laboratory for Software Safety at Iowa State University. Most noticeable is the extension of Software FTA (SFTA) to address the impact of product-line variation on safety analysis [17] [18] [19] [20]. This approach is based on a technique for the development of a product-line SFTA in the domain engineering phase and a pruning (or trimming) technique for reusing this SFTA for the analysis of new product-line instances. The approach offers a systematic approach to treating SFTA results as a reusable product-line asset. In particular, the ability to partially automate the pruning of a product-line SFTA, supported by domain expert reviews, should improve confidence in the analysis outcomes. The approach was later extended to demonstrate how to integrate product-line SFTA with the product-line requirements [21]. Further, it was used to integrate the results of the product-line SFTA with state-based models [21]. Based on this integration, reusable test scenarios can be generated for examining the validity of design definitions.

Finally, in a case study examining the impact of product-line variation on the safety assessment data for a Full Authority Digital Engine Control (FADEC), obtained from Rolls-Royce Controls, Stephenson et al created a dependency matrix that traces “*areas of vulnerability*” in the safety assessment data as a consequence of changes “*when moving from one product to another*”. A key conclusion of this case study was that a dependency matrix has the potential to improve traceability between product-line design and safety assessment, particularly in “*enhancing the completeness and robustness of a product line’s safety-related requirements*” [23].

9 Summary and Conclusions

In this paper, we showed how a product-line safety case can be developed in such a way that it can be reused for the assurance of individual product-line instances. We showed that, in addition to the reuse of typical design assets, safety case structures can form a key part of a safety-critical product line’s core assets. We adopted a ‘neat’ way for developing product-line safety cases using the GSN patterns and modular extensions. In particular, we demonstrated how optionalities and multiplicities in GSN patterns can be restricted to address the extrinsic variation in a product-line safety case in a traceable way. This traceability was realised using explicit obligations linking extrinsic safety case variation to their source in the context, feature and architectural models. Further, this paper did not introduce any new symbols which would unnecessarily complicate the development or comprehension of GSN-based safety arguments. This paper only introduced guidance on how the existing GSN (core, patterns and modular GSN) can be used to develop product-line safety cases.

Finally, it is important to note that the approach presented in this paper is suitable for safety-critical product-lines where the impact of variation can be traceably identified, examined and justified. For example, aerospace and automotive applications often satisfy this criterion by being driven by the need to reduce unnecessary complexity in order to simplify the design, and therefore the assessment, of safety-critical functions. To this end, this approach is not suitable for novel and complex applications in poorly understood

domains, as variation would aggravate an already complex problem, an aspect that should be avoided for safety applications. It is also noteworthy that, like in any intellectual design activity, the usage of a particular notation does not mechanically guarantee the production of the intended artefact. Therefore, although the modular and patterns extensions of GSN can help in defining a clear, structured and configurable product-line safety case, engineers need first and foremost to demonstrate a good understanding of design concepts related to abstraction, information hiding and separation of concerns. The product-line safety case approach presented in this paper can only be of value to competent safety case developers with adequate understanding of the product-line scope and domain.

References

1. Clements, P., Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley, Reading (2001)
2. Weiss, D.M., Robert, C.T.: *Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley Professional, Reading (1999)
3. Kelly, T.P.: *Arguing Safety – A Systematic Approach to Safety Case Management*. DPhil Thesis, Department of Computer Science, University of York, UK (1998)
4. Bate, I.J., Kelly, T.P.: *Architectural Considerations in the Certification of Modular Systems*. In: Anderson, S., Bologna, S., Felici, M. (eds.) *SAFECOMP 2002*. LNCS, vol. 2434, p. 321. Springer, Heidelberg (2002)
5. Industrial Avionics Working Group (IAWG): *Modular Software Safety Case Process – Part A: Process Definition (October 2007)*, <http://www.assconline.co.uk/>
6. Attwood, K., Kelly, T.P., McDermid, J.A.: *The Use of Satisfaction Arguments for Traceability in Requirements Reuse for System Families*. In: *International Workshop on Requirements Reuse in System Family Engineering (2004)*
7. Fenn, J., Hawkins, R., Kelly, T.P., Williams, P.: *Safety Case Composition Using Contracts – Refinements Based on Feedback from an Industrial Case Study*. In: *15th Safety Critical Systems Symposium (2007)*
8. Dowding, M.: *Maintenance of the Certification Basis for a Distributed Control System – Developing a Safety Case Architecture*. MSc Report, Department of Computer Science, University of York, UK (2002)
9. Alexander, C.: *A Pattern Language: Towns, Buildings, Construction*. OUP, USA (1978)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading (1995)
11. Object Management Group (OMG): *Systems Modelling Language*. vol. 1.1, OMG (2008)
12. EUROCAE/RTCA: *ED-12B/DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. EUROCAE/RTCA (1994)
13. UK Ministry of Defence (MoD): *00-55 Requirements of Safety Related Software in Defence Equipment. Part 2: Guidance, Issue 2, Defence Standard*, UK Ministry of Defence (1997)
14. International Organization for Standardization (ISO): *ISO26262 Road vehicles – Functional safety*. Draft, Baseline 15 (2009)
15. Society of Automotive Engineers (SAE): *Aerospace Recommended Practice 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*. SAE (November 1996)

16. Federal Aviation Administration (FAA): AC 20-148: Reusable Software Components (December 2004)
17. Dehlinger, J., Lutz, R.: PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool. *Automated Software Engineering* 13(1), 169–193 (2006)
18. Feng, Q., Lutz, R.: Bi-Directional Safety Analysis of Product Lines. *Journal of Systems and Software* 78(2), 111–127 (2005)
19. Dehlinger, J., Lutz, R.: Software Fault Tree Analysis for Product Lines. In: 8th IEEE International Symposium on High Assurance Systems Engineering (HASE 2004), Florida, USA (2004)
20. Dehlinger, J., Lutz, R.: Fault Contribution Trees for Product Families. In: 13th International Symposium on Software Reliability Engineering (2002)
21. Dehlinger, J., Humphrey, M., Suvorov, L., Padmanabahn, P., Lutz, R.: Decimal and PLFaultCAT: From Product-Line Requirements to Product-Line Member Software Fault Trees, Research Demonstration. In: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis (2007)
22. Liu, J., Dehlinger, J., Lutz, R.: Safety Analysis of Software Product Lines Using State-Based Modeling. *Journal of Systems and Software* 80(11), 1879–1892 (2007)
23. Stephenson, Z.R., de Souza, S., McDermid, J.A.: Product Line Analysis and the System Safety Process. In: 22nd International System Safety Conference (2004)