A model-based approach to support the systematic reuse and generation of safety artefacts in safety-critical software product line engineering

André Luiz de Oliveira

André Luiz de Oliveira

A model-based approach to support the systematic reuse and generation of safety artefacts in safety-critical software product line engineering

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION.*

Concentration Area: Computer Science and Computational Mathematics.

Advisor: Prof. Dr. Rosana Teresinha Vaccare Braga (ICMC/USP)

Co-Advisor: Prof. Dr. Timothy Patrick Kelly (CS Dept./The University of York)

USP – São Carlos
July, 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo autor

```
      Oliveira, André Luiz de
O48m      A model-based approach to support the systematic
       reuse and generation of safety artefacts in safety-
       critical software product line engineering / André
       Luiz de Oliveira; orientadora Rosana Teresinha
       Vaccare Braga; co-orientador Timothy Patrick Kelly.
       -- São Carlos, 2016.
          379p.

          Tese (Doutorado - Programa de Pós-Graduação em
      Ciências de Computação e Matemática Computacional)  --
       Instituto de Ciências Matemáticas e de Computação,
      Universidade de São Paulo, 2016.

         1. Software product lines. 2. Compositional
      safety analysis. 3. Variability management. 4.
      Model-driven development 5. Reuse. I. Braga, Rosana
      Teresinha Vaccare, orient. II. Kelly, Timothy
      Patrick, co-orient. III. Título.
```

André Luiz de Oliveira

Uma abordagem dirigida a modelos para apoiar o reuso sistemático e geração de artefatos de *safety* em engenharia de linhas de produtos de sistemas embarcados críticos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação (ICMC-USP) como parte dos requisitos para a obtenção do título de Doutor em Ciências – Ciências da Computação e Matemática Computacional. *VERSÃO REVISADA.*

Área de Concentração: Ciência da Computação e Matemática Computacional.

Orientadora: Profa. Dra. Rosana Teresinha Vaccare Braga (ICMC/USP)

Co-orientador: Prof. Dr. Timothy Patrick Kelly (CS Dept./The University of York)

USP – São Carlos
Julho, 2016

For my mother and father

ACKNOWLEDGMENTS

I would like to thank my family for their support to the completion of this thesis. I thank my wife Francine for her love, attention, and patience. I thank my mother Divina and father Luiz Carlos for their love, dedication and constant support in all moments of my life. I would to thank my aunt Vanda for her affection. I would to thank my sister Thiellen for her friendship and love.

I would to thank my advisor Rosana Teresinha Vaccare Braga for trust me the opportunity to work with you and for her friendship, dedication, valuable advices and ideas, and confidence in the supervision of this thesis.

I would like to thank my advisor Paulo Cesar Masiero for trust me the opportunity to work with you, for his friendship, valuable advices, and support in the supervision of this thesis.

I would like to thank my co-advisor Tim Kelly for trust me the opportunity to work with you for his friendship, receptivity when I arrived in the University of York, dedication, supervision, valuable ideas and important contributions to this thesis.

I would like to thank Ibrahim Habli for his friendship, constant support and valuable insights that contributed to this thesis.

I would like to thank Yiannis Papadopoulos for his friendship, for giving me access to the HiP-HOPS tool that helped me to put in practice my research, for his valuable advices and contributions for this thesis.

I would like to thank Richard Hawkins for his insights, support and provision of the model-based assurance cases tooling used during the course of this thesis.

I would like to thank Luiz Azevedo, David Parker, and Ioannis Sorokos researchers from the Dependable Systems research group of the University of Hull for their support with the HiP-HOPS and Tabu Search optimization algorithms.

I would like to thank the post-graduate students from the Software Engineering and Hypermedia Labs of the ICMC-USP and from the Computing Department of the Federal University of São Carlos for their dedication during the participation of the experimental study carried out during the course of this thesis.

I would to thank all members from LaBES/ICMC research group for their friendship, moments of relaxation and happiness proportioned through these five years.

I would like to thank my friends from São Carlos and ICMC, specially Maykon, Elias Adriano, Rafa Durelli, Edison, João Paulo, Kleberson, Johana, Raíza, Vânia, Milena, Maurício, Faimison, Silvana Morita, Thiago Gottardi, Victor, Lucas, Rafael Oliveira, Brauner, Diógenes, Iohan, Stevão, and Sidgley for their friendship, companionship, conversations, and good moments through these years.

I would like to thank my friends Pablo, Claúdio Rodolfo, and Matheus for the good moments during the period of my master degree at Federal University of São Carlos.

I would like to thank my undergraduate friends Cesar Augusto and Wellington Della Mura for their friendship and good moments through the undergrad period.

I would to thank my colleagues from HISE group: Richard Hawkins, Katrina Attwood, and Xiacheng Ge for their friendship, support and contributions through the period of my stay at the Department of Computer Science in the University of York.

I would like to thank Pedro, Tony, Andrey, Elena, Joan, Billy, Marc, Mélanie, Virginie, Ana, Nathan, and Imran for their friendship and good moments that have brought more happiness for my first months in York.

I would to thank Victor Bandur, Cristina Sechel, Frank Zeyda, Gustavo, Aninha, Gabriel, Abdul-Aziz for their friendship, good and happiness moments during my stay in York. Special thanks to Victor Bandur, Cristina Sechel, and Frank Zeyda for their support during my stay in York.

I would like to thank Valter Camargo and Rosângela Penteado for their friendship, support and valuable advices through my first steps with research at the Federal University of São Carlos.

I would like to thank the teachers from the "Universidade Estadual do Norte do Paraná" for their support through all my undergraduate period.

I would like to thank all teachers and staff from the ICMC/USP São Carlos and The University of York.

I would like to thank CNPq, INCT-SEC, and ICMC Pro-EX for the financial support.

“The mind that opens itself to a new

idea will never go back to its original size.”

(Albert Einstein)

ABSTRACT

Software Product Line Engineering (SPLE) has been proven to reduce development and maintenance costs, improving the time-to-market, and increasing the quality of product variants developed from a product family via systematic reuse of its core assets. SPLE has been successfully used in the development of safety-critical systems, especially in automotive and aerospace domains. Safety-critical systems have to be developed according to safety standards, which demands safety analysis, Fault Tree Analysis (FTA), and assurance cases safety engineering artefacts. However, performing safety analysis, FTA, and assurance case construction activities from scratch and manually for each product variant is time-consuming and error-prone, whereas variability in safety engineering artefacts can be automatically managed with the support of variant management techniques. As safety is context-dependent, context and design variation directly impact in the safety properties changing hazards, their causes, the risks posed by these hazards to system safety, risk mitigation measures, and FTA results. Therefore, managing variability in safety artefacts from different levels of abstraction increases the complexity of the variability model, even with the support of variant management techniques. To achieve an effective balance between benefits and complexity in adopting an SPLE approach for safety-critical systems it is necessary to distinguish between reusable safety artefacts, whose variability should be managed, and those that should be generated from the reused safety artefacts. On the other hand, both industry and safety standards have recognized the use of model-based techniques to support safety analysis and assurance cases. Compositional safety analysis, design optimization, and model-based assurance cases are examples of techniques that have been used to support the generation of safety artefacts required to achieve safety certification. This thesis aims to propose a model-based approach that integrates model-based development, compositional safety analysis, and variant management techniques to support the systematic reuse and generation of safety artefacts in safety-critical software product line engineering. The approach contributes to reduce the effort and costs of performing safety analysis and assessment for a particular product variant, since such analysis is performed from the reused safety artefacts. Thus, variant-specific fault trees, Failure Modes and Effects Analysis (FMEA), and assurance case artefacts required to achieve safety certification can be automatically generated with the support the model-based safety analysis and assurance case construction techniques.

Key-words: Software product lines, Compositional safety analysis, Variability management, Model-based development, Reuse.

RESUMO

Engenharia de Linha de Produtos de Software (ELPS) contribui para a redução dos custos de desenvolvimento e de manutenção, a melhoria do "*time-to-market*", e o aumento da qualidade de produtos desenvolvidos a partir de uma família de produtos por meio do reuso sistemático dos ativos principais da linha de produtos. A ELPS vem sendo utilizada com sucesso no desenvolvimento de sistemas embarcados críticos, especificamente nos domínios de sistemas automotivos e aeroespaciais. Sistemas embarcados críticos devem ser desenvolvidos de acordo com os requisitos definidos em padrões de segurança, que demandam a produção de artefatos de análise de segurança, árvores de falhas e casos de segurança. Entretanto, a realização de atividades de análise de segurança, análise de árvores de falhas e construção de casos de segurança de forma manual para cada produto de uma linha de produtos é uma tarefa demorada e propensa a erros. O gerenciamento de variabilidade em artefatos de análise de segurança pode ser automatizado com o apoio de técnicas de gerenciamento de variabilidades. Em virtude de "*safety"* ser uma propriedade dependente de contexto, a variabilidade no projeto e contexto inerente uma linha de produtos software impacta na definição de propriedades de segurança do sistema, modificando as ameaças à segurança do sistema, suas causas e riscos, medidas de mitigação aplicáveis, e resultados de análise de árvore de falhas. Dessa forma, gerenciar variabilidades em artefatos relacionados à "*safety"* em diferentes níveis de abstração aumenta a complexidade do modelo de variabilidade mesmo com o apoio de técnicas de gerenciamento de variabilidades. Para alcançar o equilíbrio eficaz entre os benefícios e a complexidade da adoção de uma abordagem de ELPS para o desenvolvimento de sistemas embarcados críticos é necessário fazer a distinção entre artefatos de "*safety*" reusáveis, em que a variabilidade deve ser gerenciada, e artefatos de "*safety*" que devem ser gerados a partir de artefatos reusáveis. Por outro lado, tanto a indústria quanto os padrões de segurança têm reconhecido o uso de técnicas dirigidas a modelos para apoiar a análise segurança e a construção de casos de segurança. Técnicas de análise de segurança composicional e otimização de projeto, e de construção de casos de segurança dirigido a modelos vêm sendo utilizadas para apoiar a geração de artefatos de "*safety*" requeridos para certificação. O objetivo desta tese é a proposta de uma abordagem dirigida a modelos que integra técnicas de desenvolvimento dirigido a modelos, análise de segurança composicional e otimização de projeto, e construção de casos de segurança dirigido a modelos para apoiar o reuso sistemático e a geração de artefatos de "*safety*" em engenharia de linhas de produtos de sistemas embarcados críticos. A abordagem proposta reduz o esforço e os custos de análise e avaliação de segurança para produtos de uma linha de produtos, uma vez que tal análise é realizada a partir de artefatos de "*safety*" reusados. Assim, artefatos como análises de árvores de falhas e de modos de falha e efeitos, e casos de segurança requeridos para certificação podem ser gerados automaticamente com o apoio de técnicas dirigidas a modelos.

Palavras-chave: Linha de produtos de software, Análise de segurança composicional, Gerenciamento de variabilidades, Desenvolvimento baseado em modelos, Reuso.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

AADL – Architectural Analysis and Description Language

AFM – Architectural Failure Model

ARP - Aerospace Recommended Practice

ASIL – Automotive Safety Integrity Level

BVR – Base Variability Resolution

CAA- Civil Aviation Authority

CAE – Claim-Argument-Evidence

CFM – Component Failure Model

CFTs - Component Fault Trees

CVL – Common Variability Language

DAL – Development Assurance Level

EMF – Eclipse Modeling Framework

EGL - Epsilon Generation Language

EOL – Epsilon Object Language

Epsilon - Extensible Platform of Integrated Languages for mOdel maNagement

ETA - Event Tree Analysis

ETL - Epsilon Transformation Language

EUROCAE - European Organization for Civil Aviation Equipment

FAA - Federal Aviation Administration

FDA – Food and Drug Administration

FFA – Functional Failure Assessment

FFM – Functional Failure Model

FHA – Functional Hazard Assessment

FLM – Failure Logic Model

FMEA – Failure Modes and Effects Analysis

FMECA – Failure Modes and Effects and Criticality Analysis

FODA - Feature-Oriented Analysis Domain

FPTN – Failure Propagation Transformation Notation

FPTC – Failure Propagation Transformation Calculus

FTA – Fault Tree Analysis

GSN - Goal Structuring Notation

HAZOP - HAZard and OPerability Studies

HiP-HOPS – Hierarchically Performed Hazard Origin and Propagation Studies

IEC – International Electro-technical Commission

ISO – International Standardization Organization

MBAC – Model-Based Assurance Cases

OSATE – Open Source AADL Tooling Environment

OMG - Object Management Group

PSSA - Preliminary System Safety Assessment

RTCA - Radio Technical Commission for Aeronautics

SAE – Society of Automotive Engineers

SACM - Structured Assurance Case Metamodel

SC-PLE – Safety-Critical Product Line Engineering

SEFTs - State-Event Fault Trees

SIL – Safety Integrity Level

SPMS – Structured Patterns Metamodel Standard

SPL – Software Product Lines

SPLE - Software Product Line Engineering

SSA – System Safety Assessment

SysML – System Modeling Language

UML - Unified Modeling Language

UAV - Unmanned Aircraft Vehicle

CONTENTS

Chapter 1

INTRODUCTION

1.1 Context

Software Product Lines (SPL) is as a software development approach where a set of software-intensive systems that share a common and 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 (CLEMENTS and NORTHROP, 2001). Software Product Line Engineering (SPLE) has been proven to reduce development and maintenance costs, improve the time-to-market, and increase the quality of the development of product variants from a product family via systematic reuse of its core assets (CMU, 2015; ISO, 2011; CLEMENTS and NORTHROP, 2001). It has been reported that SPLE has been successfully used in the industry (CMU, 2013; LINDEN *et al.* 2007). SPLE lifecycle comprises domain engineering and application engineering processes (POHL *et al.* 2005).

In domain engineering, the scope of the product line is defined comprising common and variable requirements specified in a feature model (KANG *et al.* 1990), and their supporting core assets, e.g., component models, class diagram, or test cases, which "*realize*" the product line features in the solution space. In application engineering, concrete product variants are defined via systematic reuse of product line core assets. When variant-specific requirements are not fully addressed by the product line, after product derivation, variant-specific assets are added to the generated variant. Product line variant management techniques, e.g., BVR toolset (VASILEVSKIY *et al.* 2015), pure::variants (PURE::SYSTEMS, 2016), GEARS (BIG LEVER, 2016), and Hephaestus (BONIFÁCIO *et al.* 2009), use feature models and product variants to define the scope of the product line, and map features to variation points in the core assets, which need to be resolved in application engineering. During the application engineering, a product variant is defined by selecting variant-specific features from the feature model, and variant management techniques automatically resolve the variation points defined in the core assets, to generate the core assets for a concrete product variant (DOMIS *et al.* 2015).

SPLE and component-based approaches have been considered by the industry in the development of safety-critical systems, especially in automotive (SCHULZE *et al.* 2013; WEILAND, 2006) and aerospace domains (DORDOWSKY *et al.* 2011; HABLI *et al.* 2007). However, safety-critical systems have to be developed according to guidance prescribed by safety standards, e.g., IEC 61508 (IEC, 2010) for the functional safety of electrical/electronic/programmable electronic safety-related systems, ISO 26262 (ISO, 2011) for automotive systems, DO-178C (RTCA, 2012) and SAE ARP 4754A (EUROCAE, 2010) for aerospace systems, and EN 50159:2001 (CENELEC, 2001) for railway systems. Safety standards establish that safety-related properties of a product should be analyzed and demonstrated at different levels of abstraction before releasing the system for operation. At functional level, specifically on requirements and earlier stages of the design, *Hazard Identification* and *Risk Assessment* are performed to identify the potential threats to the safety of a product and their risks, e.g., in terms of probability and severity, safety requirements are allocated to mitigate hazard effects, and *Failure Analysis* identifies the hazard causes (i.e., safety analysis). At detailed design, *Fault Tree Analysis* (FTA) identifies the causal chains that can lead the system to an unsafe state. At system level, *Failure Modes and Effects Analysis* (FMEA) identify the impact of component failures on the overall safety of the system. In addition, safety standards from different domains have recommended or mandate the development of an assurance case as a pre-requisite to achieve safety certification (ISO, 2011; EUROCAE, 2010). Assurance case is a clear, comprehensive and defensible argument, supported by a body of evidence, which demonstrates that a system is acceptably safe to operate in a particular context (KELLY, 2003). Therefore, to take the benefits of the SPLE approach in safety-critical systems development, the systematic reuse of safety properties (safety analysis) and generation of safety artefacts should be taken into account.

1.2 Motivation/Problem

Performing system safety analysis, FTA, and FMEA from scratch or in an ad-hoc manner, manually and individually for each variant of a product line is a time-consuming and error-prone task, whereas variability in safety engineering core assets can be systematically and automatically managed with the support of variant management techniques. The ability to reuse safety analysis, and not only requirements, design, and implementation assets, is important for safety-critical software product line engineering by reducing the effort in performing safety analysis for product variants (DOMIS *et al.* 2015; OLIVEIRA *et al.* 2014; BAUMGART *et al.* 2014; HABLI *et al.* 2009). Otherwise, the value of a safety-critical SPL can be undermined if there is a need to derive fault trees, FMEA results, and assurance cases by performing safety analysis from scratch for each product variant whereas these artefacts are potentially expensive. So, SPLE can contribute to reduce the costs of product-specific safety analysis through structured and managed reuse. However, in safety-critical SPLE, simple asset reuse is not a technically sufficient approach to establish assurance artefacts, i.e., safety assessment and assurance cases, for a given product variant.

As safety is context-dependent, product line context and design variation directly impact the safety properties changing hazards, their causes, and the risks posed by these hazards to system safety, and risk mitigation measures. Safety properties may change according to the selection of product variants, thereby lower-level safety artefacts such as FTAs cannot be straightly reused since they may also change according to the selection of product variants. In order to achieve the reuse of safety assets, product line safety analysis should be performed aware of context and design variation that need to be further mapped to variation points on the safety analysis core assets in the product line variability model, with the support of a variant management technique. So, safety analysis can be only treated as a product line asset only if the variability model contains information about variation points and their realization in safety analysis assets. Therefore, safety analysis artefacts should be included in the product line core assets, to enable the systematic reuse of safety properties together with other product line assets, reducing the costs of generating safety artefacts for a larger number of variants built around the product line core assets.

Although the reuse of safety artefacts provided by an SPLE approach is an attractive idea, existing product line approaches for reuse of safety assets are focused on the reuse of safety assets that can be derived from product line safety analysis, e.g. fault trees, FMEA, and assurance cases. For example, the extensions of Software Fault Tree Analysis (SFTA) for product lines (LIU *et al.* 2007; DEHLINGER and LUTZ, 2006; FENG and LUTZ, 2005; DEHLINGER and LUTZ, 2004) consider the product line FTA and FMEA as a reusable asset. These techniques are limited to manage the variability in the fault tree leaf nodes, not covering variability in fault tree gates (DEHLINGER and LUTZ, 2007). Other approaches are also focused on the reuse of fault trees and FMEA results (SCHULZE *et al.* 2013; GOMEZ *et al.* 2010; HABLI, 2009). Although Gomez *et al.* (2010) present an approach to support the reuse of component fault trees of similar systems as input for the construction of variant-specific fault trees, their reuse strategy is based on "*clone and own*", which is time-consuming and error-prone. Habli and Kelly (2010) provides a systematic approach to support the reuse of product line assurance cases, by linking context and design variation to the assurance case. This approach has been built upon a catalogue of conceptual safety assessment metamodels named "*Product Line Safety Metamodel*" (HABLI, 2009; HABLI *et al.* 2009). These metamodels define how contextual and design variation are traced to variation on safety assessment artefacts, produced at the functional (hazard and risk analysis), architecture (fault trees), and component (FMEA) abstraction levels, and the assurance case. These traceability links should be addressed by product line assets to support the systematic reuse of safety-critical components. So, achieving the reuse of safety artefacts at different levels of abstraction requires traceability of context and design variations throughout architecture, safety analysis, fault trees and FMEA, and assurance case artefacts. However, managing variability at different levels of abstraction increases the complexity of the product line variability model, even with the support of variant management techniques. Thus, changes in a product line core asset are propagated throughout the variability model, whose composition rules associated with each affected core asset should be added, modified, or removed.

In order to achieve an effective balance between benefits and complexity in adopting an SPLE approach for safety-critical systems it is necessary to distinguish between reusable safety artefacts, in which variability should managed, and safety artefacts that can be generated from the reused safety artefacts. Hazard and risk analysis, safety requirements, and component failure analysis, which are primary safety artefacts produced early on the design during the product line domain engineering, can be considered reusable artefacts in which variability should be managed. On the other hand, at detailed design, since the structure of safety assessment assets, e.g., fault trees and FMEA, and assurance cases are dependent upon variant-specific architecture and safety analysis, they cannot be reused, but they must be generated. However, the manual generation of these artefacts for each individual variant is time-consuming and error-prone. In addition, changes in a given product variant demand the regeneration of these artefacts, increasing the costs to achieve safety certification. Fortunately, these safety artefacts do not need to be manually created. There is a growing field of interest in safety-critical systems engineering that has been exploring, for more than twenty years, the potential use of model-based techniques to support automated safety assessment (DELANGE and FEILER, 2014; BATTEUX *et al.* 2013; PAPADOPOULOS *et al.* 2011; JOSHI *et al.* 2005; WALACE, 2005; FENELON and McDERMID, 1993), and more recently, assurance case construction (HAWKINS *et al.* 2015; DENNEY *et al.* 2015; DENNEY *et al.* 2014). With regard to assurance cases, Model-Based Assurance Cases (MBAC) (HAWKINS *et al.* 2015) supports the automatic generation of assurance cases for a given system from a diverse set of safety assessment and system models. Additionally, both industry and safety standards (RTCA, 2012; ISO, 2011; EUROCAE, 2010), especially in aerospace and automotive domains, have recognized the maturity of model-based development techniques, and model-based engineering became a reality in safety-critical systems development (FEILER and NIZ, 2008; LISAGOR *et al.* 2006).

The use of models provides benefits related to unambiguous expression of requirements and architecture, and the provision of automated support for development and safety assessment (RTCA, 2011). The SAE ARP 4754A has considered the use of model-based techniques to support the safety assessment process in aerospace systems. Thus, safety standards have been recognized the possibility of using model-based safety assessment techniques. So, existing model-based safety assessment techniques have been largely adopted by the industry. These techniques can be classified in one the following categories: compositional safety analysis, e.g., AADL Error Annex (DELANGE and FEILER, 2014), HiP-HOPS (PAPADOPOULOS et al. 2011), Failure Propagation and Transformation Calculus (FPTC) (WALLACE, 2005), and Failure Propagation and Transformation Notation (FPTN) (FENELON and McDERMID, 1993); or extensions of formal verification techniques, e.g., AltaRica (BATTEUX et al. 2013), or both, which is the case of AltaRica. In this thesis, we are interested on compositional safety analysis techniques. These techniques provide formal or semi-formal languages to enable the specification of the system failure behavior by characterizing the failure behavior of each individual system component, i.e., failure logic modeling.

Compositional safety analysis integrate architecture and failure modeling in a single model, and automate parts of safety analysis, enabling the automatic synthesis of FTA and FMEA from a system model enhanced with failure behavior information. By integrating failure analysis into the system model, compositional techniques support the efficient and consistent evolution of design and failure models, reducing the costs and improving the quality of system safety analysis (DOMIS and TRAPP, 2008; LISAGOR *et al.* 2006; JOSHI *et al.* 2005). Compositional safety analysis techniques are effective in gathering safety and reliability information of a system, which are important factors to be considered to take architectural decisions. Since the manual evaluation of multiple choices in the design space against optimization objectives such as reliability and cost is time consuming, design optimization techniques provide automated analysis of candidate design solutions against optimization objectives, returning near optimal solutions. In the context of safety-critical systems development, design optimization can be used to support the decomposition of safety integrity requirements throughout contributing component failures in order to achieve compliance with safety standards without being unnecessarily stringent or expensive. There exists a set of design optimization techniques that support the allocation of safety integrity requirements in automotive and aerospace systems (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; PAPADOPOULOS *et al.* 2011; BIEBER *et al.* 2011). This thesis shows how these techniques can be used to support the analysis and allocation of safety integrity requirements in safety-critical product lines and their instances.

This thesis demonstrates how adopting a software product line engineering approach that integrates compositional safety analysis, variant management, and model-based development, supports the systematic reuse of safety analysis and automatic generation of fault trees, FMEA, safety integrity requirements, and assurance case safety artefacts. The distinction between reusable safety artefacts and safety artefacts that must be generated enables the automated traceability of product line variation throughout architecture and safety artefacts as defined in "*Product Line Safety Metamodel*" (HABLI, 2009), whilst it reduces the complexity of the variant management on product line core assets related to the variability model.

1.3 Research Challenges

Integrating system safety engineering, variability management and model-based development disciplines in a Safety-Critical Software Product Line Engineering approach poses the following challenges addressed in this thesis:

- To define means for reusing safety-related domain artefacts in such way that allows variant-specific safety analysis to be generated;
- To link safety analysis artefacts and system models, so that assurance cases can be generated for specific product variants; and
- To establish means of analyzing and optimizing how safety integrity requirements should alternatively, be allocated to different product variants in order to achieve process-based safety certification, i.e., compliance with safety standards.

From the analysis of the above challenges, the following general research questions have been derived:

- ***RQ1:*** *How safety analysis reuse and automated generation of safety assessment and assurance case artefacts can be conciliated in safety-critical product line engineering processes?*
- ***RQ2:*** *How to integrate system safety engineering, variant management, and model-driven development to achieve traceability, reuse, and automation in safety-critical product line engineering?*

The following specific research questions have been derived from these two general research questions:

- ***RQ3***: *How is context and design variation traced to and managed in product line safety analysis (hazard and risk analysis, and functional failure assessment) and assurance (safety assessment and assurance cases)?*
- ***RQ4:*** *How linking assurance case patterns, safety assessment and system models to generate assurance cases for a specific product variant?*
- ***RQ5:*** *How to perform the allocation of safety integrity requirements for product line components in order to achieve compliance with safety standards without being expensive?*

The research questions are addressed by the thesis objectives presented in the following section.

1.4 Thesis Objectives and Hypothesis

The objective of this thesis is to establish a model-based approach that integrates system safety engineering, variability management, and model-based development, to support the systematic reuse of safety analysis artefacts and automated generation of assurance artefacts in software product line engineering for safety-critical systems.

This thesis provides contributions for each one of the three areas mentioned in the research challenges, and it addresses the following hypothesis:

*Through adopting a model-based approach for managing and **tracing** variability in architectural and safety models it is **feasible** to support the **traceable** and **systematic** construction of safety assessment and assurance case artefacts within a product line engineering approach for safety-critical systems.*

In the following, the key terms highlighted in the thesis hypothesis are explained:

- ***Tracing/Traceable:*** it is related to the ability of the approach in creating traceability links between product line variation, reused safety analysis assets, and generated safety assessment and assurance case artefacts;
- ***Feasible:*** regarded the practicability of the approach in supporting reuse and automation in product line safety assessment and assurance case construction, especially in automotive and aerospace domains;
- ***Systematic:*** it means that the approach is repeatable, i.e., its application in different domains, safety standards, and chosen tools and techniques follows the same steps and it generates the same output artefacts.

The hypothesis terms are the criteria adopted to evaluate the thesis contributions.

1.5 Thesis Structure

This thesis is organized into nine chapters:

Chapter 2 presents the literature review on system safety assessment, traditional and modern safety assessment techniques, safety certification, assurance cases, software product lines, and the state of the art of research on product line safety assessment and assurance cases.

Chapter 3 presents an overview of the proposed systematic and holistic approach to integrate compositional safety analysis, variant management, and model-based development techniques to support safety-critical software product line engineering processes. This approach extends traditional Software Product Line Engineering methods with the incorporation system safety engineering into product line processes. The parts of the approach are detailed in Chapters 4 to 7.

Chapter 4 presents a systematic approach to support variability management in safety models, guidelines to adapt existing variant management tools to support variability management in safety models, and the development of tooling support.

Chapter 5 shows the contributions in product line compositional safety assessment: a systematic approach to integrate compositional safety analysis into product line processes, and a method and tool built upon the HiP-HOPS Tabu Search design optimization tool for automated analysis and allocation of safety integrity requirements to product line components to support process-based certification.

Chapter 6 presents a mode-based approach to support the automated generation of assurance cases for product line instances, and the results of an experimental study that assessed the feasibility of the MBAC approach and tooling (HAWKINS *et al.* 2015), in terms of effort and quality of the generated artefact, in software product line engineering processes.

Chapter 7 presents the proposed extensions to the MBAC approach to support the specification of pattern instantiation constraints in assurance case patterns, and the concept of artefact pattern and its integration with assurance case patterns.

Chapter 8 presents the evaluation of the thesis contributions against the criteria defined in the thesis hypothesis (Section 1.4).

Finally, **Chapter 9** presents a summary of the thesis contributions, their limitations, and future research directions. The list of publications resultant from this thesis is presented in Appendix E.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

This chapter presents the concepts and a review of the literature on system safety engineering, software product line engineering, and supporting model-based techniques used through this thesis. Firstly, an overview of system safety assessment, which comprises the definition of the safety terminology used through this thesis, hazard identification, risk assessment, and allocation of safety requirements processes, is presented (Section 2.2). In the following, a review on traditional and compositional safety assessment techniques is presented, followed by a review on goal-based and process-based safety certification approaches (Section 2.3). Section 2.4 presents how structuring and representing assurance arguments and evidence through safety-critical system development lifecycle, and it shows existing model-based techniques to support the automated generation of assurance cases. Software product line engineering is introduced, and a literature review on variability management, product line processes, and existing model-based techniques for product line variability management are presented in Section 2.5. Finally, a survey on existing research in product line safety assessment, variability management on safety models, and assurance cases is presented in Section 2.6. Section 2.7 presents a summary of this chapter.

2.2 Safety Assessment

This section presents the safety terminology used through this thesis, an overview functional hazard assessment process: hazard identification, risk assessment, and allocation of safety requirements. Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) techniques and their relationships with system safety assessment process is also presented, followed by an overview of existing model-based safety assessment techniques.

2.2.1 Safety Terminology

Although the key safety terms used through this thesis may change depending on the domain, standard, and geographic region, in order to avoid ambiguity, the definition of these terms are highlighted in the following. ***Safety*** in the context of risk assessment is defined as "*freedom from unacceptable risk*" (PAPADOULOS and McDERMID, 1999). Safety can be confused with reliability. However, they have different meanings. ***Reliability*** relates to "*the probability that a piece of equipment or component will perform its intended function satisfactorily for a prescribed time and under stipulated environmental conditions"* (LEVESON, 1995). Therefore, ***reliability*** relates to all potential failures, whilst ***safety*** is only associated with hazardous failures (LEVESON, 1996). A ***hazard*** is defined as "*the potential source of harm caused by malfunctioning behavior of the item*" (ISO, 2011). An item is defined as *"a system or an array of systems that implement a function"* (ISO, 2011). Whereas safety should be considered in the context of a particular system in a given operating environment, a ***system*** is defined as *"a combination, with defined boundaries, of elements that are used together in a defined operating environment to perform a given task or achieve a specific purpose. Elements may include personnel, procedures, materials, tools, equipment, facilities, services and/or software as appropriate"* (MoD, 2007).

In context of hazard analysis and risk assessment processes from system engineering for safety-critical systems, a ***safety risk*** is defined as the *"combination of the likelihood of harm and the severity of that harm"* (MoD, 2007). ***Harm*** can be defined as "*death, physical injury, damage to the health of people, or damage to property or the environment*" (MoD 2007). A ***failure*** is *"the inability of a system or system component to perform a required function within specified limits". "A failure may be produced when a fault is encountered"* (RTCA, 2012). A ***fault*** is a *"manifestation of an error if it occurs, may cause a failure"*

(RTCA, 2012). Safety requirements can be allocated to mitigate the effects of a *fault*. ***Safety requirement*** is defined as the required risk reduction measures associated with a given *hazard*, or *component failure* (MoD, 2007). There are three types of safety requirements. ***System safety requirements*** are allocated to system-level *hazards.* Design and implementation decisions, e.g., system functions, intended to eliminate or minimize the effects of failures in the system safety are called ***functional safety requirements*** (ISO, 2011) or ***derived safety requirements*** (MoD, 2007). Finally, ***safety integrity requirement*** specifies the reliability (risk), e.g., in terms of probability and severity, associated with a given *component failure*, *hazard*, or *functional safety requirement*. In safety management, ***assurance*** is defined as "*planned and systematic actions necessary to provide adequate confidence and evidence that a product or process satisfies the given safety requirements*" in order to achieve safety certification (RTCA, 2012). From the analysis of existing definitions for evidence, in this thesis, ***evidence*** is considered as the "*information that serves as the grounds and starting-point of (safety) arguments, based on which the degree of truth of the claims in arguments can be established, challenged and contextualized"* (SUN, 2013).

2.2.2 Hazard and Risk Analysis

Hazard identification and risk assessment are preliminary activities in the system safety assessment process (RTCA, 2012; EUROCAE, 2010). Hazard identification is intended to identify hazardous functional failure conditions, i.e., failure conditions leading the system to an unsafe state. At the risk assessment, the risk factors associated with each identified hazard is determined, e.g., in terms of severity and probability. However, with regard to guidelines for development and safety assessment of safety-critical systems, there is not a consensus in a universal approach since safety standards may vary from one industry, domain, or region to another (MoD, 2007). For example, military standards (MoD, 2007; MIL-STD-882E, 2000) tend to accept the consideration of novel techniques and technologies for developing safety-critical systems. On the other hand, civil standards, e.g., DO-178C (RTCA, 2012) and ISO 26262 (ISO, 2011), are highly prescriptive. Although such difference, it is common for these standards to cover basic safety engineering activities such as hazard and risk analysis, allocation of safety requirements, and provision of safety evidence. Interactions between "*Hazard Identification"*, "*Risk Assessment"*, and "*Allocation of Safety Requirements"* activities, detailed in the following, represent the core of a safety lifecycle. These activities typically occur prior the allocation of the safety requirements to software and hardware items. Figure 2.1 shows the relationships between these safety activities in IEC 61508 safety standard.

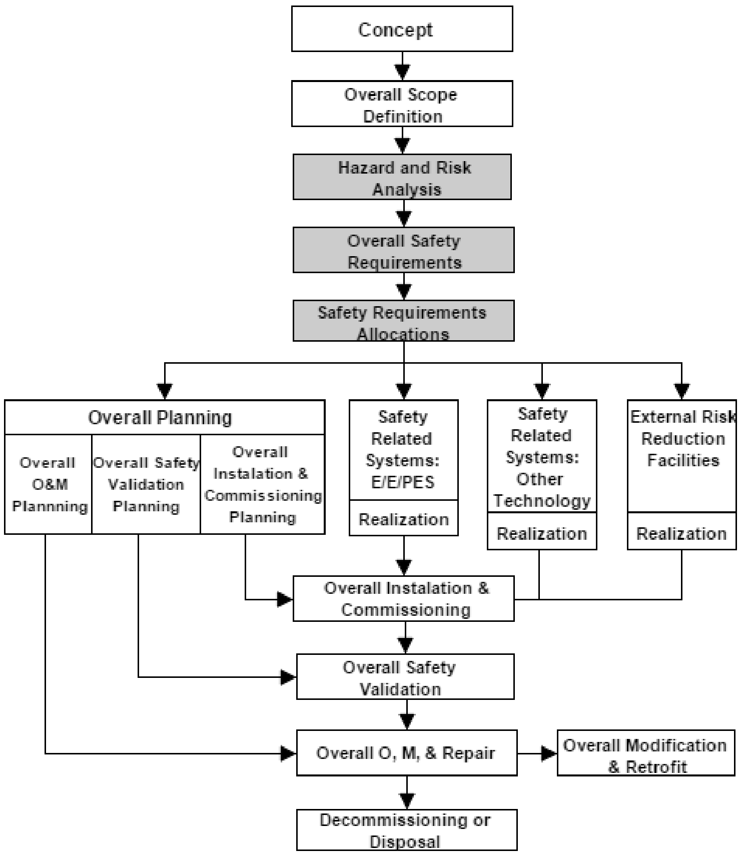

Concept
Overall Scope Definition
Hazard and Risk Analysis
Overall Safety Requirements
Safety Requirements Allocations
Overall Planning
Overall O&M Plannning
Overall Safety Validation Planning
Overall Instalation & Commissioning Planning
Safety Related Systems: E/E/PES
Realization
Safety Related Systems: Other Technology
Realization
External Risk Reduction Facilities
Realization
Overall Instalation & Commissioning
Overall Safety Validation
Overall O, M, & Repair
Overall Modification & Retrofit
Decommissioning or Disposal

Figure 2.1. IEC 61508 safety lifecycle (IEC, 2010).

Hazard Identification can be performed after the system and its target operating environment is specified and understood. The system should be specified in terms of the physical and operational environment, system boundaries, functions and inter-functional dependencies. Hazard identification establishes hazards and sequence of events that could cause system failures (MoD, 2007). The identified hazards are usually stored into a hazard log, which is updated throughout the system development life-cycle. The hazard identification can be performed using checklists, "*what IF*" analysis, Functional Failure Assessment (SAE, 1994), or HAZard and OPerability Studies (HAZOP) (KLETZ, 1992). Hazard identification can be performed interactively as the system design evolves. Thus, the level understanding with regard to hazards and their risks evolves in parallel to the evolution of the system architecture and its deployment, later evolved based on the information gathered from system operation (HABLI, 2009).

Risk Assessment is performed after hazard identification with the aim to estimate, based on probabilistic criteria e.g., likelihood and severity, the risk posed by these hazards

(IEC, 2010). In many countries, e.g., United Kingdom and United States, it is necessary to establish risk tolerability criteria as well as demonstrating that the risks posed by hazards are "*As Low As Reasonably Practicable*" (ALARP). It means that for a risk to be considered ALARP, the cost of additional risk reduction should be "*grossly disproportionate to the benefit obtained from that Risk Reduction"* (HSE, 2016). Determining whether a risk is ALARP requires informed judgment and in most cases the adoption of formal decision making techniques. According to the ALARP principle illustrated in Figure 2.2, a risk can be classified as:

- **Intolerable**: this risk does not satisfies ALARP and hence should be reduced;
- **Tolerable**: the risk satisfies the ALARP principle;
- **Broadly acceptable:** the risk is acceptable as long as the system addresses relevant good practices, e.g., formal verification is performed to generate the evidence, which demonstrates that the risk posed by a given hazard is acceptable.

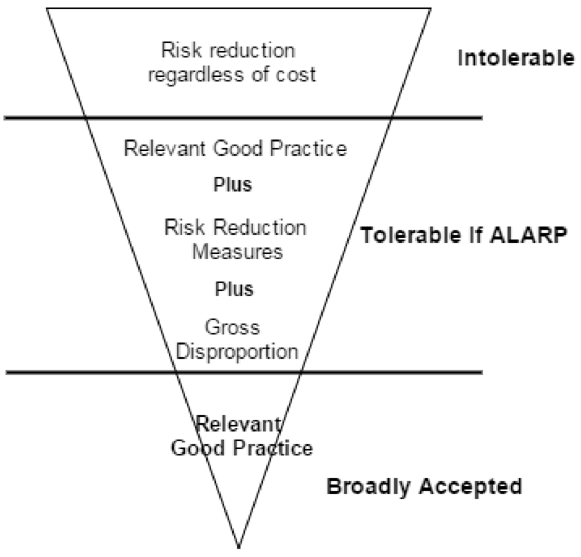


Figure 2.2. ALARP principle (HSE, 2016).

Allocation of Safety Requirements is performed from the analysis of the outputs provided by hazard identification and risk assessment. Therefore, risk reduction measures are allocated to each identified system hazard in the form functional safety requirements, or safety integrity requirements. Safety integrity requirements are defined according to quantitative/probabilistic criteria established in the target safety standard (MoD, 2007). Most safety standards adopt such criteria to establish safety integrity requirements. In these standards, the higher is the importance of the safety requirements to address system safety, more stringent the safety integrity requirements are. These standards also establish different quantitative criteria to specify safety integrity requirements. For example, the safety integrity can be defined in terms of mean-time-to-failure, probability of fault-free operation, or unavailability (LITTLEWOOD and STRIGINI, 1993). IEC 61508 establishes safety integrity requirements in terms of two categories of failure rates: probability of failure in performing high demand/continuous functions, and probability of failure to perform on demand functions. Therefore, each one of the Safety Integrity Levels (SILs), shown in Table 2.1, is associated with ranges of failure rates in both categories. The SIL 4 is the most stringent and SIL 1 the less stringent.

Table 2.1 – Probabilistic criteria and safety integrity levels in IEC 61508 (IEC, 2010).

Safety Integrity Level	**Continuous** **Probability of dangerous failure per Hour**	**On Demand** **Probability of failure to perform the Function**
4	$10^{-9} < P \leq 10^{-8}$	$10^{-4} < P \leq 10^{-5}$
3	$10^{-8} < P \leq 10^{-7}$	$10^{-3} < P \leq 10^{-4}$
2	$10^{-7} < P \leq 10^{-6}$	$10^{-2} < P \leq 10^{-3}$
1	$10^{-6} < P \leq 10^{-5}$	$10^{-1} < P \leq 10^{-2}$

Safety integrity requirements are allocated to system components within the system architecture. If a system component comprises hardware and software components, safety integrity requirements are refined and allocated to these components. If multiple safety integrity requirements from different stringencies are allocated to components, each one of these components should be developed to address the higher stringent SIL (HABLI, 2009). Additionally, existing design optimization techniques to support the automatic allocation and decomposition of safety integrity requirements (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; PARKER *et al.* 2013; PAPADOPOULOS *et al.* 2011; BIEBER *et al.* 2011) can be used to provide sufficient separation and partitioning without being stringent or expensive. Design optimization techniques (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; WALKER *et al.* 2013) implement SIL allocation and decomposition rules defined in existing safety standards, e.g., ISO 26262 ASIL (Automotive Safety Integrity Level) allocation rules in automotive and ARP 4754A DAL (Development Assurance Level) allocation rules in aerospace domains.

Hazard identification, risk assessment, and allocation of safety requirements activities are the starting point of a safety lifecycle. The following sections present a review on existing traditional and modern safety analysis techniques commonly used to support identification of hazards and contributing failures modes of components, and risk assessment.

2.2.3 Traditional Safety Analysis Techniques

Existing safety analysis techniques are usually categorized as inductive and deductive (ISO, 2011; EUROCAE, 2010). Inductive analysis techniques are known as bottom-up, which the analysis starts from component failure events and it identifies their potential effects on the system, i.e., it identifies the hazards in which failure events might directly or indirectly contribute. On the other hand, deductive analysis is a top-down technique starting from an undesirable top-event (i.e., a hazard) and it searches for failure events which are possibly causes for the top-event. Functional Failure Analysis (FFA) (SAE, 1996), Failure Modes and Effects Analysis (US MILITARY, 1977), and Event Tree Analysis (ETA) are examples of traditional inductive safety analysis techniques, whilst Fault Tree Analysis (NASA, 2002) is an example of deductive analysis technique. Although such classification, there no restriction in using an inductive technique in a deductive fashion and vice-versa. For example, fault trees are largely used in aerospace domain to support Preliminary System Safety Assessment (PSSA) (EUROCAE, 2010) to examine whether the system architecture satisfies the safety requirements established at the Functional Hazard Assessment (FHA) phase. Fault tree analysis is also commonly used to investigate possible causes of an accident (HABLI, 2009). The following sections present an overview of FFA, FTA, and FMEA traditional safety analysis techniques.

2.2.3.1 Functional Failure Analysis (FFA)

Functional failure analysis aims to examine how system functions can impact on, or contribute to system safety (SAE, 1996). FFA identifies *failure conditions* associated with system functions and their *effects* on the overall safety. Failure conditions can be identified based on the following deviations: function not provided, function provided when not required, or function provided incorrectly. Later, the *effects* of each failure condition are, then identified by defining how failure conditions affect the intended behavior of the system, its environment, and users. Effects provide assumptions about the system environment, e.g., outside temperature and flight phase in aerospace systems. FFA classifies failure conditions based on the severity of their effects where failure conditions leading to death or injuries are classified as "*Catastrophic*" or "*Hazardous*". Conversely, less severe failure condition effects are typically classified as "*Major*" or "*Minor*". The criteria to classify failure condition effects vary according to the target safety standard. Finally, *safety integrity requirements* are allocated to failure condition effects according to their severity. Table 2.2 shows an excerpt of FFA for "*Decelerate Aircraft on the Ground*" aircraft function (SAE, 1996).

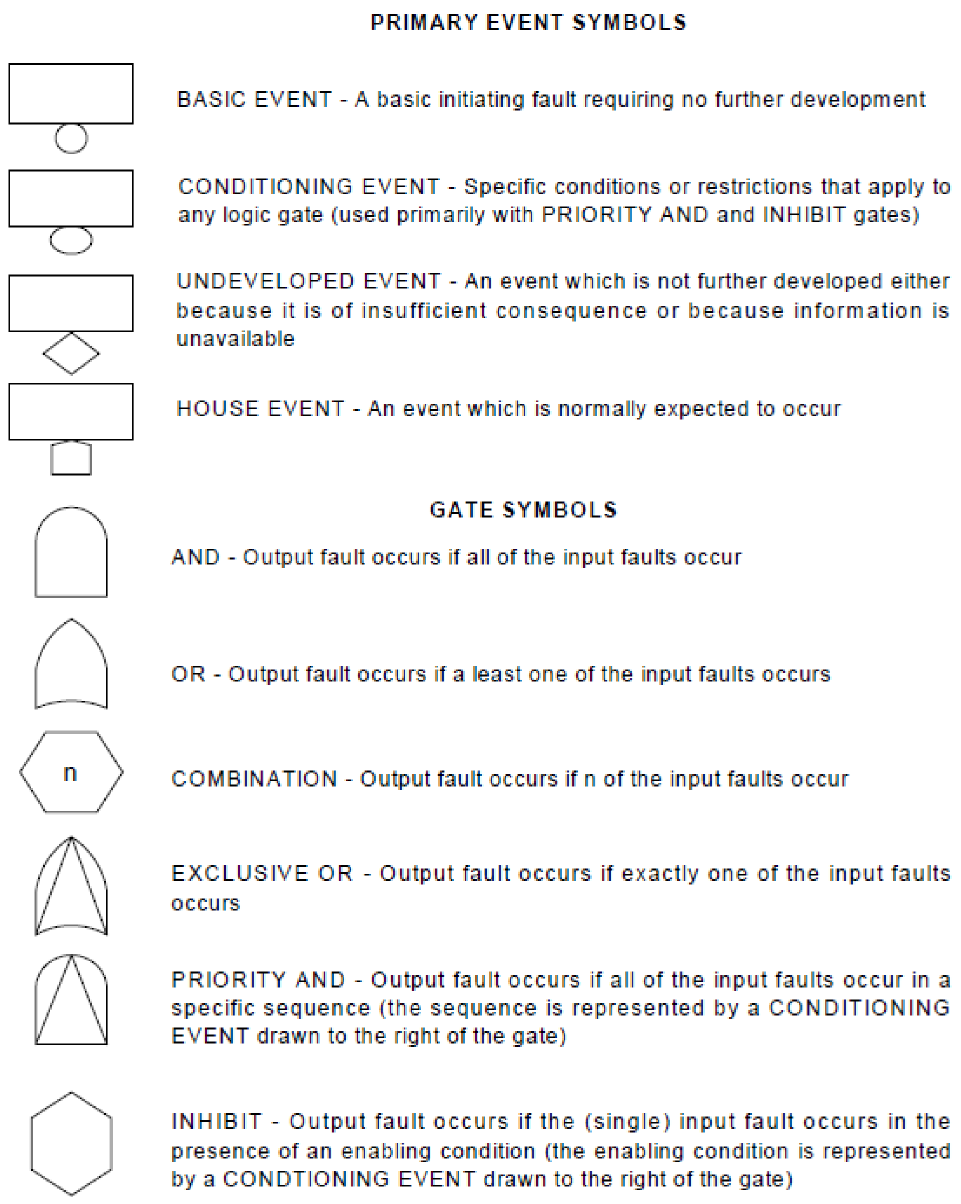
Table 2.2 – FFA for '*decelerate aircraft on the ground*' system function (SAE, 1996).

Function	Failure Condition	Phase	Effects	Classification (severity)	DAL
	Un-annunciated loss of Deceleration Capability	Landing	Crew is unable to decelerate the aircraft, resulting in a high speed overrun.	Catastrophic	A
Decelerate Aircraft on the Ground	Annunciated loss of Deceleration Capability	Landing	Crew selects a more suitable airport, notifies emergency ground support, and prepares the occupants for landing overrun.	Hazardous	B
	Un-annunciated loss of Deceleration Capability	Taxi	Crew is unable to stop the aircraft on the taxi way or gate, resulting in low speed contact with terminal, aircraft or vehicles.	Major	C

Considering the safety lifecycle, such as defined in ISO 26262, ARP 4754A, and IEC 61508 standards, and compositional safety analysis, at the hazard identification, FFA identifies combinations between functional failure conditions, named *hazards*, which lead to the occurrence of system-level failures. The risks posed by these hazards are further assessed based on probabilistic criteria, e.g., likelihood and severity, and safety integrity requirements are allocated to each identified hazard. Later on the analysis, FFA identifies failure conditions associated with each system function that directly or indirectly contribute to the occurrence of hazards (WINKINSON and KELLY, 1998).

2.2.3.2 Fault Tree Analysis and FMEA

Fault Tree Analysis (NASA 2002) and FMEA (US MILITARY, 1977) are well-established safety analysis methods largely used in safety-critical systems engineering in aerospace, automotive, rail, and nuclear power domains. These methods are intended to support safety analysts in identifying potential faults on the system early on the design, in order to use such information to prevent these faults. FTA is a deductive technique by considering the analysis of a top-event, typically a system failure, and then deducing its causes. At the PSSA phase from ARP 4754A safety assessment process, a top-event of a fault tree can be a hazard identified in FFA. Fault tree analysis is performed based on the preliminary design of the system architecture in order to identify failure modes of components, called basic events, leading to the top-level event. Fault trees are graphical representations of logical combinations of component failures that consist in a top-event connected to one or more basic events via logical gates such as AND, OR, or NOT, as illustrated in Figure 2.3. Fault tree analysis can be performed qualitatively, by means of logical analysis, or quantitatively, via probabilistic analysis. Qualitative analysis is performed by considering basic events, their relationships, and their contribution to the top-event. Quantitative analysis calculates the probability of occurrence of a given top-event based on the analysis of the probability of occurrence of each basic event.



PRIMARY EVENT SYMBOLS

BASIC EVENT - A basic initiating fault requiring no further development

CONDITIONING EVENT - Specific conditions or restrictions that apply to any logic gate (used primarily with PRIORITY AND and INHIBIT gates)

UNDEVELOPED EVENT - An event which is not further developed either because it is of insufficient consequence or because information is unavailable

HOUSE EVENT - An event which is normally expected to occur

GATE SYMBOLS

AND - Output fault occurs if all of the input faults occur

OR - Output fault occurs if a least one of the input faults occurs

COMBINATION - Output fault occurs if n of the input faults occur

EXCLUSIVE OR - Output fault occurs if exactly one of the input faults occurs

PRIORITY AND - Output fault occurs if all of the input faults occur in a specific sequence (the sequence is represented by a CONDITIONING EVENT drawn to the right of the gate)

INHIBIT - Output fault occurs if the (single) input fault occurs in the presence of an enabling condition (the enabling condition is represented by a CONDTIONING EVENT drawn to the right of the gate)

Figure 2.3. Fault tree graphical notation symbols (NASA, 2002).

Figure 2.4 shows an example of a fault tree for "*rupture of the pressure tank*" top-event within pressure tank control system extracted from the Fault Tree Handbook (NASA, 2002). The top-event can be caused by one or more of the following events: 1) tank rupture due to improper installation (undeveloped event); 2) tank rupture primary failure (basic event); or 3) tank rupture secondary failure (intermediary event). The intermediary event can be caused by two events (represented as undeveloped events), which are further investigated.

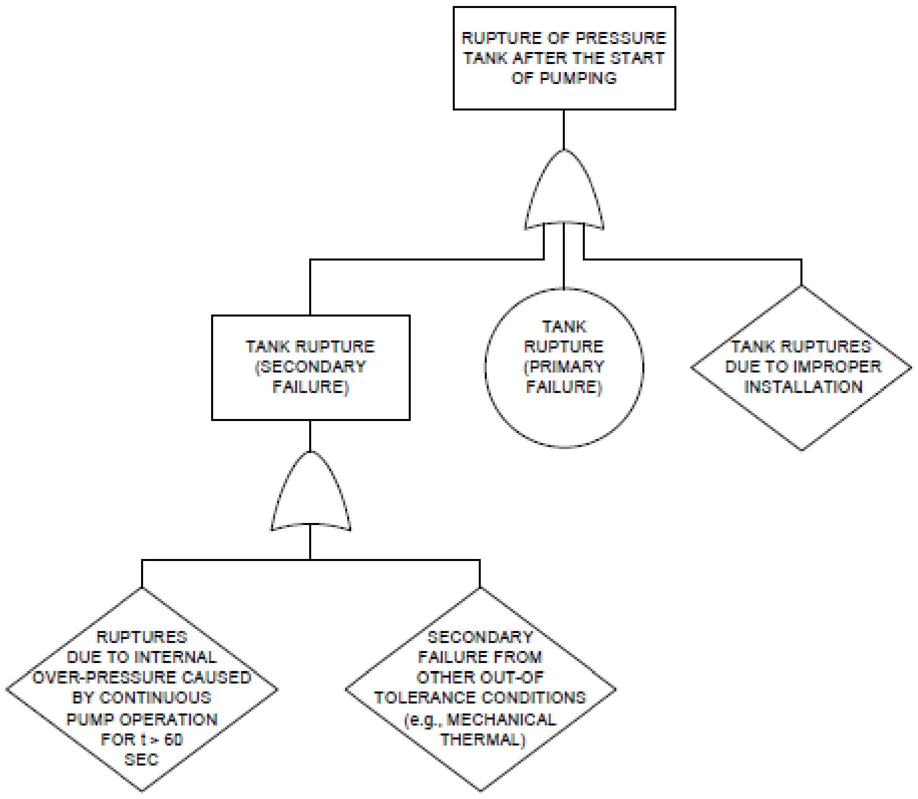

RUPTURE OF PRESSURE TANK AFTER THE START OF PUMPING
TANK RUPTURE (SECONDARY FAILURE)
TANK RUPTURE (PRIMARY FAILURE)
TANK RUPTURES DUE TO IMPROPER INSTALLATION
RUPTURES DUE TO INTERNAL OVER-PRESSURE CAUSED BY CONTINUOUS PUMP OPERATION FOR t > 60 SEC
SECONDARY FAILURE FROM OTHER OUT-OF TOLERANCE CONDITIONS (e.g., MECHANICAL THERMAL)

Figure 2.4. Rupture of the pressure tank fault tree (NASA, 2002).

Fault tree analysis in PSSA phase aims at refining high-level safety requirements allocated to system-level hazards during the FFA, to derive safety requirements to be allocated to components of the system architecture. Therefore, component developers should produce evidence that demonstrates the satisfaction of the allocated safety requirements, which can be functional or safety integrity requirements (HABLI, 2009). Unlike FTA, FMEA is an inductive technique where the analysis starts from *known* failure modes of components by inferring the effects of these failures on the overall safety of the system (PAPADOPOULOS *et al.* 2011; PUMFREY, 1999). Thus, FMEA differs from FFA where the analysis is based on hypothetical failure modes. An effect in an FMEA corresponds to a top-event of a fault tree. Effects can be evaluated based on criteria such as severity, probability, and detectability that can be combined to estimate the overall risk. The results of an FMEA are usually presented in a tabular form. Independently from the domain, FMEA tables basically include the identification of (PUMFREY, 1999):

- The component or subsystem under consideration;
- The *known* component/subsystem failure modes; and
- The effects of each failure mode.

If FMECA (Failure Modes, Effects and Criticality Analysis) (US MILITARY, 1977), a variation of FMEA, which classifies the risk posed by each failure mode based on its contributions to system-level hazards according to criteria such as severity/probability/risk classification, is carried out, columns for criticality information should be included (NASA, 2002). Although the similarities in the structure and analysis process of component FMEA/FMECA and FFA inductive techniques, FFA is a predictive technique commonly used early on the safety lifecycle, to identify potential threats to the system safety and allocating safety requirements, and FMEA is used late in safety lifecycle to demonstrate that the system has addressed the allocated safety requirements (PUMFREY, 1999).

Fault trees and FMEA/FMECA provides valuable information about the system failure behavior, and can be used complementary to each other. FTAs provide the causal information for the system-level failures, and FMEA/FMECA provides a view of the impact of component failures on the overall system safety. However, due FTA and FMEA being manual methods, their application in complex systems is error prone and time-consuming. Since the system architectural design impact the overall safety, changes in the design demand costly reviewing of FTAs and FMEA artefacts (PAPADOPOULOS *et al.* 2011). This occurs because FTA and FMEA dot not yield reusable models. In addition, performing FTA and FMEA require the knowledge about the whole system, which is not easily done in the context of complex systems that combine traditional technologies with computer-based controllers (LISAGOR *et al.* 2006). The deficiencies of traditional FTA and FMEA techniques have been recognized not only by researchers but also by aircraft manufacturers with the revision of the SAE ARP 4761 (SAE, 1996) document that includes a specific section dedicated to model-based automated safety assessment. Therefore, a number of tools and techniques have been developed to automate the system safety analysis process. These tools and techniques are discussed in the following section.

2.2.4 Model-Based Safety Assessment

Although existing tools such as Fault Tree+ (ISOGRAPH SOFTWARE, 2016) and SETS (WORRELL and STACK, 1978) provide graphical-user interface support for creating fault trees and FMEA, they are purely manual. Modern tools and techniques to support safety analysis have shifted towards integrating safety analysis with the system design process, enabling automatic synthesis of FTA and FMEA from a system design model with failure behavior information (DELANGE and FEILER, 2014; PAPADOPOULOS *et al.* 2011; JOSHI *et al.* 2005). It enables the reuse of safety information in an iterative design process. These safety analysis techniques and tools can be classified as compositional or extensions of formal verification techniques (LISAGOR *et al.* 2006).

Compositional safety analysis techniques provide formal and semi-formal languages to support the specification, composition, and analysis of the system failure behavior based on safety information about the system components. Based on a specific language, a failure logic model of the system is created by characterizing the failure behavior of each individual system component with its output failure modes, input failure modes and internal failures that contribute to the occurrence of output failures. The failure logic model connects output failure modes of a component with input failure modes of another component, creating flows of failure modes. UML/SysML-based failure modeling techniques (KÄβMEYER *et al.* 2015; DAVID *et al.* 2010), AADL Error Annex (DELANGE and FEILER, 2014), and HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) (PAPADOPOULOS *et al.* 2011) are examples of compositional safety analysis techniques. Despite compositional techniques are not fully automated, requiring manual input of component failure data, they are applicable in the analysis of both hardware and software architectures in the context of model-based design. Design optimization techniques based on meta-heuristics, such as genetic algorithms (AZEVEDO *et al.* 2014; PARKER *et al.* 2013; DEB *et al.* 2002) can be used to support compositional safety analysis. Optimization techniques provide the design assessment based on optimization objectives such as cost and reliability (PAPADOPOULOS *et al.* 2011). These algorithms automatically generate near optimal candidate solutions that satisfy pre-defined optimization objectives, supporting safety analysts in taking design decisions to meet the safety requirements. Optimization techniques also provide candidate SIL allocation solutions based on optimization objectives such as cost and reliability (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; PARKER *et al.* 2013a).

The second category of model-based safety assessment approaches consists in extensions of formal verification techniques to support safety analysis named fault injection approaches. It involves the use of formal modeling languages and model-checking techniques to automatically deduce abnormal conditions (failure modes) that may lead the system to an unsafe state (LISAGOR *et al.* 2006). These techniques simulate the normal functioning of the system design and then inject faults to determine their effects on the system safety (PAPADOPOULOS *et al.* 2011). Fault injection approaches do not require the specification of component failure logic to describe the failure propagation as all required information is extracted from the model, based on a library of pre-defined failure types. Examples of techniques that fall into this category are AltaRica 3.0 (BATTEUX *et al.* 2013), Safety Analysis Modeling Language (SAML) (GÜDEMANN and ORTMEIER, 2010), and Formal Safety Analysis Platform-NuSMV (BOZZANO and VILLAFIORITA, 2007). Specifically, AltaRica provides a formal modeling language to specify the system model, and a model checker to simulate the system behavior in the presence of injected faults.

Compositional and extensions of formal verification safety analysis approaches provide clear benefits for the system safety assessment process and have reached levels of maturity required to be included in industrial processes (LISAGOR *et al.* 2006). Both approaches can be applied complementary to each other at different stages of the system development lifecycle. Compositional safety analysis is applicable from early stages of system design to provide the basis for incremental safety assessment in order to identify safety problems early on the design, driving re-design and allocation of safety requirements. Extensions of formal verification techniques are applicable only at later stages of the design, e.g., towards the end of PSSA or in SSA phases from SAE ARP 4754A safety assessment process, when the design is relatively mature and detailed. These techniques provide automated analysis of the detailed system design to verify whether or not the system design meets the allocated safety requirements (LISAGOR *et al.* 2006). In this thesis we are interested in compositional model-based safety analysis and design optimization techniques to support product line safety assessment. Compositional safety analysis and design optimization techniques are detailed in the following two subsections.

2.2.4.1 Compositional Safety Analysis

Whereas compositional techniques are based on failure logic modeling approach, it allows safety analysts to model the failure behavior of the system incrementally as the design progresses from the system architecture to a detailed level. It facilitates the adoption of a “divide-and-conquer” approach for safety assessment, where the assessment of a complex system is broken down into more manageable tasks of characterization of the failure behavior of individual components (LISAGOR *et al.* 2006). Compositional techniques enable the reuse of safety information, overcoming the limitations related the need to review FTA and FMEA artefacts when the system design is changed. Failure Propagation and Transformation Notation (FPTN) (FENELON and McDERMID, 1993) and its extension Failure Propagation and Transformation Calculus (FPTC) (WALLECE, 2005) are the earliest compositional safety analysis techniques. FPTN and FPTC have influenced other techniques such as AADL Error Annex, HiP-HOPS, State-Event Fault Trees (SEFTs) (KAISER *et al.* 2007), and Component Fault Trees (CFTs) (GRUNSKE and KAISER, 2005).

FPTN is a graphical notation to represent the failure behavior of the system based on a component module approach to describe the propagation of component failures throughout the system architecture. In FPTN, component modules are connected via input and output ports to other modules, allowing combination and propagation of failures from one module to another. FPTC was developed to overcome the limitations of FPTN related to the construction of a failure model separated from the system design model. FPTC links the failure model to the architectural model so that dependencies are identified and managed. FPTC defines a set of failure classes (omission, commission, value, early/late failure types) that are specified by means of annotations directly in the components of the system model. Failure types are used to characterize how components, their inputs and outputs can fail. FTPC has also been extended to support probabilistic analysis of component failure expressions (GE *et al.* 2009). SEFTs and CFTs were built upon the FPTN. An SEFT makes the distinction between a system in a certain state and an event that triggers a state transition. In a CFT, the component failure logic is defined as a graph of interconnected fault trees using a specification similar to FPTN and HiP-HOPS (PAPADOPOULOS *et al.* 2011).

HiP-HOPS and ADDL Error Annex are compositional safety analysis techniques/tools built upon FPTN and FTPC concepts. AADL Error Annex adds failure modeling capabilities to AADL models that can be developed with the support of Eclipse-based Open Source AADL Tool Environment (OSATE) (DELANGE and FEILER, 2014). The HiP-HOPS compositional safety analysis tool enables failure modeling in system models developed in MATLAB/Simulink (MATHWORKS, 2016), SimulationX (ITI GMBH, 2010), and Eclipse-based UML tools (i.e. EAST-ADL) (CUENOT *et al.* 2010). Both AADL Error Annex and HiP-HOPS techniques/tools take a set of local component failure data (failure logic model) describing how output failures of components are generated from combinations of internal failure modes and deviations in component's inputs, and synthesizes fault trees and FMEA artefacts that reflect the propagation of failures throughout the system architecture. HiP-

HOPS and AADL Error Annex also allow the specification of probabilistic data, e.g., failure and repair rates, in the failure logic model. Component failure logic and probabilistic data can be stored in a library to allow it to be used in other systems models.

The component failure logic relates to design components in an intuitive and clear fashion. The clear relationship between the system design and the inherent modularity of failure logic models allow the systematic reuse of component failure logic whenever design components are reused. Therefore, when the system design is changed, the identification of the effects of these changes on the failure logic model (FLM) is simplified, and the effort to adapt the FLM is proportional to the size of the change. Figure 2.5 illustrates the failure logic modeling approach to characterize the failure behavior associated with a hydraulic pump component using HiP-HOPS and FPTN techniques. The *"pump"* component is intended to provide a constant level of hydraulic pressure on its output (*"o"*). The pump has two inputs: hydraulic (*"hi"*) and electrical (*"ei"*). *"hi"* supplies the *pump* with non-pressurized hydraulic fluid; and *"ei"* feeds the motor with electrical power. With regard to the *"pump"* failure behavior, the omission of either hydraulic or electrical input causes an omission of output pressure. In this example it is assumed that the *pump* is not sensitive to any other deviations in its hydraulic inputs. At the same time, it is assumed that all relevant failure modes of the electrical current (i.e., omission, late provision and wrong - too large or too small, value) result in similar failure modes on pump's output, e.g., low current gives rise to low flow. The *pump* can also fail itself, and it can either become fully non-functional (*"broken"*) or less res-

Internal Failure	**Failure Rate**	**Repair Rate**
pump_stuck	5e-5	1e-4
pump_struggling	1.5e-5	1e-4
Output Deviation	**Description**	**Causes**
Omission-o	No pressure (i.e. omission) on pump's output. Caused by omission of fluid on the hydraulic input ("hi") or omission on the electrical input ("ei") or by the internal failure of the pump ("stuck").	Omission-hi OR Omission-ei OR pump_stuck
Value-o	Insufficient pressure on pump's output (value FM). Caused by insufficient voltage on electrical input or by the internal failure of the pump ("struggling").	Value-ei OR pump_struggling
Value-o	Excessive pressure on pump's output (value FM). Caused by excessive voltage on electrical input.	Value-ei
Late-o	Late reaching required pressure on start Caused by late provision of electrical current or by internal failure of a pump ("struggling").	Late-ei OR pump_struggling

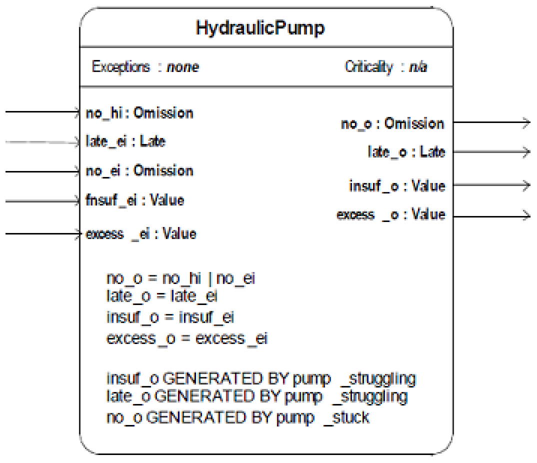


Figure 2.5. Failure logic of a hydraulic pump component: HiP-HOPS (left) and FPTN (right) (adapted from LISAGOR *et al.* 2006).

ponsive to incoming power ("*struggling*"), e.g., due to increased impedance in the windings. These internal failures may lead to the omission of output pressure or low (*insufficient*) pressure respectively. Finally, a *pump* that is "*struggling*" during start-up may provide the required hydraulic pressure later than expected (LISAGOR *et al.* 2006). Additionally, the specification the "*pump_stuck"* and "*pump_struggling"* internal failure modes contain information about their respective failure and repair rates (Figure 2.5 left).

From the perspective of traditional safety analysis techniques, the failure logic of an individual component can be seen as a complete local FMEA or a small fault tree. In terms of FTA, output failure modes represent top events, input failure modes are undeveloped events, and internal failures are basic events. The failure logic model allows safety analysts to control the level of detail of the FTA, which depending on the purpose, more detailed or less detailed fault trees can be generated at low cost due the systematic reuse of component failure logic. These different FTA views when generated from a failure logic model are consistent by construction.

2.2.4.2 Design Optimization

Compositional safety analysis techniques are effective in gathering safety and reliability information of a system architecture. Safety and reliability are important factors to be considered to take design decisions. However, the manual evaluation of multiple choices in the design space against multiple and often contradictory optimization objectives such as safety, reliability and cost is time-consuming and it can restrict the design candidates to be investigated. On the other hand, design optimization techniques support the analysis of architectures with higher degree of variability, with hundreds and thousands of candidate solutions in the design space, to obtain near optimal design solutions that meet multiple and contradictory objectives (PAPADOPOULOS *et al.* 2011). Different genetic algorithms (KULTUREL *et al.* 2006; DEB *et al.* 2002; COIT and SMITH, 1996) have been used to support the analysis of safety-critical system design space against optimization objectives such as cost and reliability. Genetic algorithms mimic the evolution of the biological life in nature in which a population of different candidates is evaluated according to their fitness and the best are chosen to reproduce and form the basis for the next generation of candidates (PAPADOPOULOS *et al.* 2011).

Tabu Search (KULTUREL *et al.* 2006) is a multi-objective and genetic algorithm. It explores the design space on the basis of evaluation functions, e.g., if a solution has better characteristics such solution is the basis for the following iteration. In Tabu Search, multiple objectives are evaluated by multiple evaluation functions in multiple iterations. Thus, the objective *"1"* is evaluated in the first iteration, objective "*2"* in the second, and so forth. A characteristic of tabu search is that it stores the solutions that have been explored, preventing looping or getting stuck in a local optimum, forcing the algorithm to focus on unexplored areas of the search space. After a number of iterations, tabu search provides the best near optimal solution(s) found.

Penalty-based (COIT and SMITH, 1996) is other genetic algorithm that combines multiple objectives in a single evaluation function. In this algorithm, an objective is optimized, e.g., reliability, but constraints on the other objectives, e.g., cost, weight are imposed. It means that any infringement of these constraints incurs in a penalty that is subtracted from the fitness score of the candidate solution. Therefore, if two candidate solutions have the same reliability, but one violates the constraints, the solution that violated the constraint is ranked lower than the one that have not violated it. Non-Dominated Sorting (NSGA-II) (DEB *et al.* 2002) is another form of genetic algorithm. Similar to Tabu Search, NSGA-II is a multi-objective algorithm that explores the design space more widely in comparison with Penalty-based, whereas an evaluation function is provided peer optimization objective. In NSGA-II, the analysis starts from a given *"X"* solution that is compared with another solution *"Y"*. If the *"Y"* solution is better in at least one objective and no worse in any other, then the *"X"* solution is said to be dominated by *"Y"*. The set of all non-dominated solutions is known as the Pareto front, and it is the set of currently identified optimal solutions.

HiP-HOPS compositional safety analysis tool provides extensions, based on Tabu Search, NSGA-II and Penalty-Based genetic algorithms, to support automatic and optimal allocation of safety integrity requirements. These algorithms support the safety analysts in decision making processes early on the development life-cycle seeking to achieve a design solution that meets the safety requirements and minimizes the cost. HiP-HOPS design optimization algorithms also support designers to comply with safety standards by automating the process of decomposing *safety integrity requirements* allocated to system-level hazards throughout contributing failures modes in the system architecture, and evaluating possible allocations, and it determines better allocation solutions based on a given cost heuristic. The HiP-HOPS penalty-based algorithm allows to efficiently explore the search space to find optimum ASIL allocation solutions for automotive system architectures (PARKER *et al.*

2013). HiP-HOPS Tabu Search optimization algorithms support the automatic decomposition of *safety integrity requirements* in automotive (AZEVEDO *et al.* 2013) and aerospace (SOROKOS *et al.* 2015) systems. These algorithms allow finding near optimal ASIL/DAL allocation solutions based on a given cost heuristic. HiP-HOPS Tabu Search optimization algorithms explore the solution space more efficiently, providing near optimal allocation solutions within acceptable time spans in comparison with Penalty-Based (PARKER et al. 2013) and NSGA-II (PAPADOPOULOS *et al.* 2011) algorithms (AZEVEDO *et al.* 2013). Optimal allocation of safety integrity requirements contributes to achieve process-based certification without being stringent or expensive. In this thesis, HiP-HOPS Tabu Search optimization algorithms have been used to support process-based certification of automotive and aerospace product line architectures and their instances. These implementations of Tabu Search support the decomposition of safety integrity requirements in compliance with ISO 26262 and SAE ARP 4754A safety standards.

The Tabu Search extensions of HiP-HOPS (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2013) are based on the Steepest Descent Mildest Ascent (SDMA) reliability optimization method developed by Hansen and Lih (1996). The method consists of iteratively finding the ASIL/DAL that, by being decremented, reduces the cost of a solution (the steepest descent direction). The ASILs/DALs allocated to system-level hazards during the Functional Failure Assessment, the information about failure propagation in the form of Minimal Cut Sets (MCT[1]) generated from fault tree synthesis, and a cost heuristic defined by the analyst are inputs to HiP-HOPS Tabu Search performing the analysis to find suitable ASIL/DAL allocation solutions for a given system architecture. A cost heuristic is an evaluation function that expresses the relative cost jumps of developing a component according to the different safety integrity requirements. For example, considering the ISO 26262 standard, the following linear cost heuristic establishes that the costs of developing components according to different ASILs are: 0 (ASIL QM), 10 (ASIL A), 20 (ASIL B), 40 (ASIL C), and 50 (ASIL D). At the end the analysis, HiP-HOPS Tabu Search algorithm outputs near optimal ASIL/DAL allocation solutions for a given system architecture in XML format. Multiple allocation results are inputs artefacts for the proposed method and tool for automated allocation of safety integrity requirements to product line components presented in Chapter 5. The tool is built as an extension of HiP-HOPS to support the product line process-based certification. The

[1]MCS are combinations of basic events that result in system-level hazards (NASA, 2002).

following section introduces the safety certification concepts, and reviews existing safety certification approaches.

2.3 Safety Certification

The safety certification of safety-critical systems can be achieved by means of process-based and goal-based approaches. In domain-specific process-based safety standards such as SAE ARP 4754A (EUROCAE, 2010) and DO-178C (RTCA, 2012) (aerospace), ISO 26262 (ISO, 2011) (automotive), EN 50159 (CENELEC, 2001) (rail), UK Defense Standard 00-56 (MoD, 2007) (defense), FDA Infusion Pumps (FDA, 2014) (medical devices), IEC 61513 (nuclear) (IEC, 2001), and more generic standards, e.g., IEC 61508 (IEC, 2010), system safety is demonstrated by addressing safety objectives. Addressing these safety objectives demand applying a set of development techniques and methods prescribed by the standards according to an specific Safety Integrity Level (SIL), Development Assurance Level (DAL) or risk classification, as shown in Table 2.3. ISO 26262 prescribes different guidance and techniques to develop a system function according to the assigned automotive safety integrity level (ASIL). In ISO 26262, the ASIL D is the most stringent.

Table 2.3 – ASILs and software architecture verification techniques in ISO 26262 (ISO, 2011).

Techniques	ASIL			
	A	B	C	D
Walk-through of the design	++[2]	+[3]	o[4]	o
Inspection of the design	+	++	++	++
Simulation of dynamic parts of the design	+	+	+	++
Prototype generation	O	o	+	++
Formal verification	O	o	+	+
Control flow analysis	+	+	++	++
Data flow analysis	+	+	++	++

The definition of SILs or risk classifications may vary between different safety standards. For example, in DO-178C, ARP 4754A, and IEC 61508 safety standards, the allocation of safety integrity levels to the system functions is defined based on the risk reduction measures associated with a given probabilistic criteria, e.g., severity and probability. ISO 26262 adopts a risk classification scheme based on Severity (S), Exposure

[2] '++': Highly Recommended

[3] '+': Recommended

[4] 'o': No recommendation for or against its usage for the identified ASIL.

(E), and Controllability (C) to allocate safety integrity levels to system functions. IEC 61513 allocates safety categories to system functions based on deterministic criteria and engineering judgement with regard to safety consequences of potential malfunctions (IEC, 2001). Process-based safety standards also establish rules to decompose SILs allocated to system-level failures throughout contributing subsystem and component-level failures. SIL decomposition allows a safety-critical system architecture to meet a particular targeted safety integrity level assigned to a system failure without all contributing components having to meet that SIL. Therefore, whether a system failure can occur only when two independent components fail together, these components share the responsibility of meeting the SIL allocated to the given system failure, rather than each one having to meet the original SIL. Table 2.4 shows the ASIL decomposition rules defined in ISO 26262. Therefore, if an ASIL D is allocated to a system failure caused by failures in two independent "*C1*" and *"C2"* components, ASIL C and ASIL A, or ASIL B and ASIL B, or ASIL D and Quality Management (QM) integrity requirements can be respectively allocated to "*C1*" and *"C2"* to address ASIL D.

SILs directly impact in the definition of the required safety objectives to achieve process-based certification, which could significantly affect both development and production costs. Therefore, higher SILs means higher costs, because meeting more stringent safety objectives typically require more safety measures, and development effort to deliver higher-quality components. SIL decomposition allows to efficiently allocating SILs so that process-based certification can be achieved without being unnecessarily stringent or expensive with regard to addressing safety objectives. Existing model-based safety assessment techniques

Table 2.4 – ISO 26262 ASIL decomposition rules (ISO, 2011).

ASIL Decomposition Rules
ASIL D requirement → ASIL C(D) requirement + ASIL A(D) requirement
ASIL D requirement → ASIL B(D) requirement + ASIL B(D) requirement
ASIL D requirement → ASIL D(D) requirement + QM(D) requirement
ASIL C requirement → ASIL B(C) requirement + ASIL A(C) requirement
ASIL C requirement → ASIL C(C) requirement + QM(C) requirement
ASIL B requirement → ASIL A(B) requirement + ASIL A(B) requirement
ASIL B requirement → ASIL B(B) requirement + QM(B) requirement
ASIL A requirement → ASIL A requirement + QM(A) requirement

provide frameworks to support SIL decomposition in different safety standards, e.g., ISO 26262 (PAPADOPOULOS *et al.* 2010; AZEVEDO *et al*. 2014; PARKER *et al.* 2013), IEC 61508 (DHOUIBI *et al.* 2014; MADER *et al.* 2012), DO-178C and ARP 4754A (SOROKOS *et al.* 2015; BIEBER *et al.* 2011). These techniques support the automated identification of: combinations between component failures leading to system failures, and near optimal SIL decomposition strategies by means of linear programming (DHOUIBI *et al.* 2014; MADER *et al.* 2012), exhaustive (PAPADOPOULOS *et al.* 2010) and genetic algorithms (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; PARKER *et al.* 2013, PAPADOPOULOS *et al.* 2011). Specifically, genetic algorithms are capable of finding near optimal SIL decomposition strategies within acceptable time spans (AZEVEDO *et al.* 2014).

Process-based safety standards offer useful guidance on "*good practices*" related to safety engineering methods and techniques and it describes how factors such as independence in the development process can improve the confidence (HABLI 2009). However, these standards are criticized by being highly prescriptive and restricting the adoption of novel methods and techniques to achieve safety certification (McDERMID and PUMFREY, 2001). The limitation of process-based certification lies in believing that qualified tools, techniques and method are sufficient to achieve a specific level of integrity, and the fact that the correlation between prescribed techniques and system failure rate is often infeasible to justify. Due the limitations of process-based certification, safety standards such as ARP 4754A, CAP 670 (CAA, 2014), DO-178C, ISO 26262, UK Defense Standard 00-56, and FDA Infusion Pumps, have been changed towards recommending or mandating the development and management of well-structured assurance cases as a mean to achieve safety certification. Assurance case justifies that the acceptability of system safety is based on product-specific and targeted evidence, referred as goal-based or product-based certification (HABLI, 2009). The UK Defense Standard 00-56 and FDA Infusion Pumps are examples of goal-based safety standards. These standards require the submission of an assurance case that comprises safety arguments that contain safety claims, which are supported by a body of evidence that justifies that the system is acceptably safe to operate in a given environment, to achieve safety certification. In a goal-based safety argument, safety claims address the sufficiency, traceability of system safety requirements, and the absence of faults leading the system to an unsafe state. Evidence "*is information, based on established fact or expert judgement, which is presented to show that the Safety Argument to which it relates is valid*" (EUROCONTROL, 2011). Evidence corresponds to the information that demonstrates the satisfaction of safety claims, which can be classified in the following types (SUN, 2013):

- *Analytical evidence*: it can be generated from hazard analysis, causal analysis, or behavior modeling, which can provide *direct* evidence that demonstrates the absence of dangerous faults, and satisfaction of system safety requirements;
- *Qualitative evidence for "good" design and process*: is an *indirect/backing* form of evidence that serves to increase the confidence in the *direct* evidence, e.g., compliance with standards, guidance, and design rules;
- *Empirical evidence*: typically quantitative, it can be generated from testing and/or operation;
- *Evidence based on engineering judgement*: can be generated from review, inspection, expert opinion based on personal knowledge, or engineering experience.

The all early mentioned evidence types can be used to substantiate assurance case claims. However, in goal-based certification, the quality and amount of required evidence vary accordingly the criticality and safety integrity of the system. Therefore, the degree of rigor, coverage, diversity, scrutiny and independence of the evidence, and the degree of confidence that can be placed in the evidence should be proportional to the required safety integrity. Safety integrity is typically specified quantitatively in terms of failure probability and severity. Opposed to process-based certification, in goal-based certification it is responsibility of engineers to show the sufficiency of evidence in supporting safety claims stated in the assurance case, and to demonstrate that the confidence put into the evidence should commensurate with safety integrity requirements. Confidence can be expressed quantitatively, via empirical evidence (HABLI, 2009), and qualitatively, via compliance with standards and engineering judgment. Quantitative confidence provides strength evidence, and qualitative confidence provides qualitative elements to judge the adequacy of the evidence in substantiating a given safety claim.

Although the limitations of process-based certification, studies have shown that if used correctly, prescriptive process-based certification approach can be adopted complementary to goal-based certification (HAWKINS *et al.* 2013; KELLY, 2008). Therefore, process-based certification approach prescribed by standards such as DO-178C provide clear guidance on the processes and techniques to be adopted in a particular domain, and best practices on evidence selection. Process-based certification also provides the backing support to justify the confidence in the generated evidence. Additionally, process-based certification provides an authoritative definition of "*good practices*" on selecting and producing the evidence referenced in an assurance case. On the other hand, assurance case explains the role of the evidence as part of the demonstration of the safety of a particular system under consideration (HAWKINS *et al.* 2013). In this thesis, we have focused on both process-based and goal-based certification. Model-based techniques for SIL allocation were used and extended to support process-based certification of product line components and product line instances (Chapter 5). Model-based assurance cases techniques have been used to support the generation of assurance cases to achieve goal-based certification of product line instances (Chapter 6). Assurance cases and model-based assurance cases are detailed in the following section.

2.4 Assurance cases

An *Assurance Case* or *Safety Case* is defined as *"a clear, comprehensive and defensible argument, supported by a body of evidence, which demonstrates that a system is acceptably safe to operate in a particular context"* (MoD, 2007, KELLY, 1998). An *Argument* is a body of information intended to establish the relationships between claims, evidence, and contextual information. *Claim* is a proposition being asserted that is a true or false statement. *Evidence* represents objective artefacts, offered in support of one or more claims. OMG defines an *Assurance Case* as a collection of auditable *claims*, *arguments*, and *evidence* created to support the statement that a given system/service is acceptably safe by addressing its associated assurance requirements (OMG, 2015a). The relationships between claims and evidence are explicitly represented in a structured argument. The submission of an assurance case has been required by certifying authorities (FAA, 2016, ANAC, 2016, CAA, 2016) and safety standards from automotive (ISO, 2011), military (MoD, 2007), aerospace (RTCA, 2012; EUROCAE, 2010), rail (CENELEC, 2001), air traffic service (CAA, 2014), medical (FDA, 2014), and nuclear (IEC, 2001) domains as a requirement to achieve safety certification. This section presents a review on assurance case development lifecycle and existing forms of representation, emphasizing Goal Structuring Notation (GSN), and its modular and patterns extensions that were used in this thesis, together with existing model-based assurance cases technique (HAWKINS *et al.* 2015), to support the automatic generation of assurance cases in safety-critical software product line engineering (Chapter 6).

2.4.1 Assurance Case Development Lifecycle

Assurance cases are recommended to be developed incrementally, and in parallel with the system design (KELLY, 2003). Thus, the assurance case evolves as more information about the development of the system architecture and safety life-cycle is obtained (KELLY, 1998). Existing safety standards (MoD, 2007) explicitly recommend the production, presentation, and issue of at least three versions of assurance cases during the development lifecycle that yields:

- **Preliminary Assurance Case:** it justifies how safety and integrity requirements defined in the safety plan[5] are addressed. It presents an anticipated outline assurance case showing the main safety objectives, the selected approach to argue safety, and forms of evidence;
- **Interim Assurance Case**: it provides the evidence that justifies the system design specification satisfy safety and integrity requirements.
- **Operational Assurance Case**: it provides the complete evidence that demonstrates the deployed system satisfies safety and integrity requirements.

Although the absence of design detail, the production of a preliminary assurance case is important to: defining the ***scope*** of the assurance case, the key ***safety issues and objectives*** associated with the system such as system hazards, safety requirements, and applicable safety standards, the assurance ***argumentation approach*** to be adopted to argue safety, provided by assurance case patterns, the ***supporting evidence***, e.g., verification, validation, and testing artefacts that substantiate assurance claims, and the definition of development ***procedures*** to be considered during the system development lifecycle, e.g., languages, methods and tools to be used to address each Safety Integrity Level. Therefore, the early identification of safety objectives at *Preliminary Assurance Case* enables the system design to be influenced as development progresses contributing to establish a more compelling assurance case. The adoption of such approach supports the confidence in the feasibility of establishing an assurance case that evolves through safety lifecycle. It contributes to reduce potential project

[5]It defines the key safety processes, roles and responsibilities to be enacted during the system development (KELLY, 2003).

risks associated with failure in achieving system certification/safety acceptance (KELLY, 2003).

2.4.2 Representation of Assurance Cases

Argument and evidence constitute the core of an assurance case since "*an Argument without supporting evidence is unfounded, and therefore unconvincing, and Evidence without argument is unexplained, i.e., it is unclear that (or how) safety objectives have been satisfied*" (KELLY, 1998). An *Argument* is defined as "*a group of propositions which one, i.e., the conclusion, is claimed to be asserted by the others, i.e., premises, that provide support or grounds for the truth of that one*" (COPI *et al.* 2010). A simple collection of propositions cannot be considered an argument. An argument must have a structure showing that some of these propositions are premises (claims) from which a conclusion is drawn (GULA, 2002).

In safety-critical systems domain, initially, safety arguments were represented in free text, in the same way as arguments are communicated by means of claims, inference, and conclusion in the law field. However, it has been noted that the use of free text in the specification of a complex arguments may lead to an ambiguous and unclear safety argument (KELLY, 2003), being problematic in ensuring that different stakeholders have a shared understanding of the safety argument. In order to address the limitations of free text, different approaches, mainly based on tabular structures, Bayesian Believe Networks (BBN) e.g., SERENE - *SafEty and Risk Evaluation using bayesian NEts* (MARSH, 1999), and graphical notations have been proposed along with the research progress in safety engineering field. Tabular structures represent safety argument using different columns (BISHOP and BLOOMFIELD, 1998), which each one covers an argument element such as an attribute, a claim, or an evidence item as shown in Table 2.5. Tabular structures have advantages of explicit representation of an argument structure in comparison with free text (HABLI 2009; KELLY, 1998). However, tabular structures have limitations in representing hierarchical safety arguments, i.e., an argument that contain assumptions and evidence, which is further supported by another argument (HABLI, 2009).

Graphical notations such as Goal Structuring Notation from the University of York (GSN STANDARD, 2011), and Claim-Argument-Evidence (CAE) (BLOOMFIELD and BISHOP, 2010), have been created to improve the expressiveness and management of the complex safety arguments. Due the provision of mature tool support (CERTWARE, 2016, THE UNIVERSITY OF YORK, 2011, ADELARD, 2016) both notations have been largely used by the industry in the specification of complex and modular argument structures. GSN and CAE notations are both based on Toulmin's (1958) philosophical argumentation approach. Thus, these notations share the claim, argument, and evidence concepts. GSN distinct from CAE by the inclusion of: the *strategy* concept used for defining how a claim is recursively decomposed into sub-claims; and *contextual* elements that could be attached to a claim, to provide background information to facilitate the understanding of a given claim.

Table 2.5 – Tabular safety argument (Adapted from Bishop and Bloomfield, 1998).

Attribute	*Design Features*	*Claim*	*Evidence*
Fail-safety	Fault-tolerant architectures	Claim that safety is maintained under stated failure conditions, assuming the subsystems are correctly implemented.	System Hazard Analysis Fault Tree Analysis
Reliability/ Availability	Fault tolerant architectures Design simplicity	Reliability claim based on experience with similar systems.	Prior field reliability in similar applications.

2.4.2.1 Goal Structuring Notation

In the GSN core notation shown in Figure 2.6, safety arguments are represented in terms of goals, strategies, solutions, context, justifications, and assumptions elements linked using "*supported by*" or "*in context of*" relationships, forming a goal structure. A goal structure shows how *goals* (claims) are recursively decomposed into sub-goals, using *strate-*

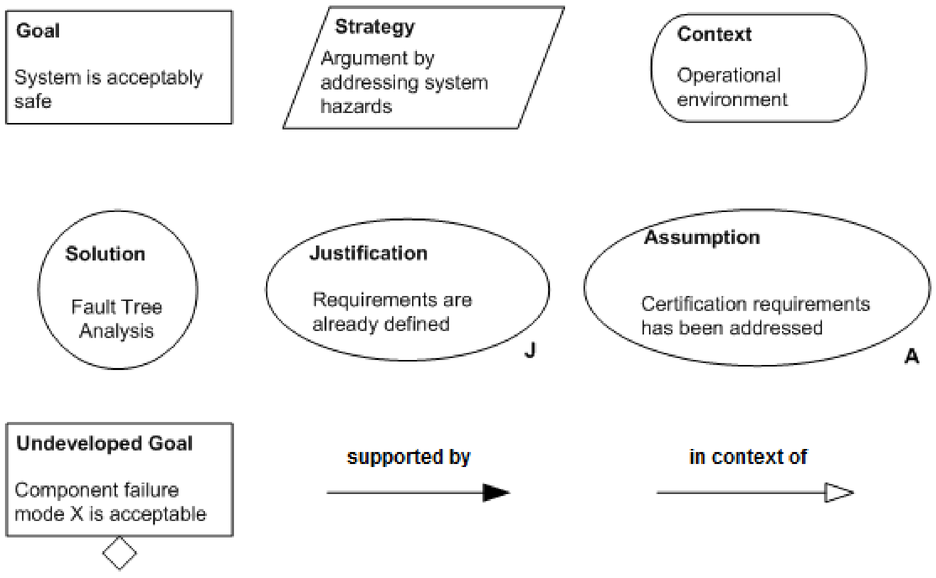

Goal
System is acceptably safe
Strategy
Argument by addressing system hazards
Context
Operational environment
Solution
Fault Tree Analysis
Justification
Requirements are already defined
J
Assumption
Certification requirements has been addressed
A
Undeveloped Goal
Component failure mode X is acceptable
supported by
in context of

Figure 2.6. GSN core (Adapted from Kelly, 2003).

gies, until reaching the point where goals can be supported by direct references to evidence items (solutions). As part of such decomposition, GSN provides elements to make clear the adopted argument decomposition approach (*strategy*), which could be qualitative, deterministic, or probabilistic (BLOOMFIELD and BISHOP, 1998), the rationale for the adopted argument decomposition approach, and the ***context*** in which goals are stated (KELLY, 2003). Finally, GSN also supports the specification of claims whose rationale is intentionally left in the argument.

Figure 2.7 shows an example of a GSN-based assurance case that justifies the safe operation of a sheet metal press (KELLY, 1998). The press is controlled by an operator via a control system based on a Programmable Logic Controller (PLC). In this argument structure, the top-level goal is *"C/S (Control System) logic is fault free" (G1)*. This goal is supported by "*Argument by satisfaction of all C/S safety requirements*" (S1) and "*Argument by omission of all identified software hazards*" *(S2)* argument decomposition strategies. "*S2"* strategy is stated in the context of the "*Identified Software Hazards*" *(C1),* which is the backing information required to understand the adopted argument decomposition strategy. Further, "*S2"* is supported by "*G8"* and *"G9"* sub-goals, which are finally supported by "*Sn3"* and "*Sn4"* solution elements, referencing "*Fault tree analysis cut-sets for event – Hand trapped in press due to command error"* and "*Hazard Directed Test results*" evidence items, no requiring further support. "*S1"* is supported by "*G2"*, "*G3"*, and *"G4"* requirements satisfac-

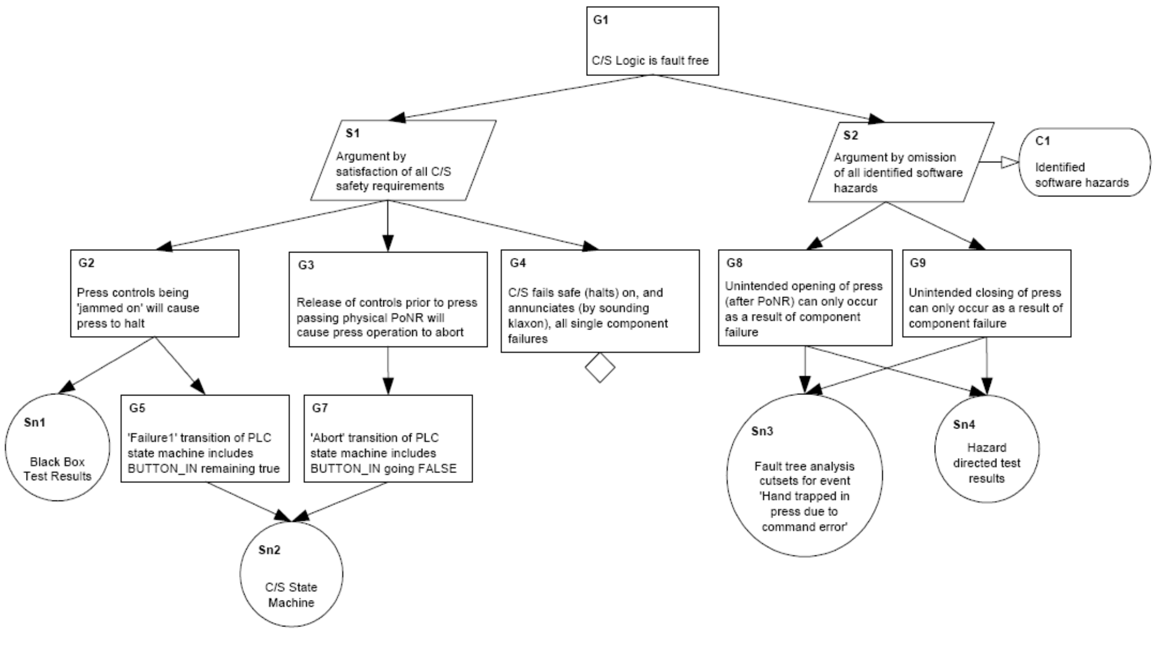

G1
C/S Logic is fault free
S1
Argument by satisfaction of all C/S safety requirements
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
G2
Press controls being 'jammed on' will cause press to halt
G3
Release of controls prior to press passing physical PoNR will cause press operation to abort
G4
C/S fails safe (halts) on, and annunciates (by sounding klaxon), all single component failures
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a result of component failure
Sn1
Black Box Test Results
G5
'Failure1' transition of PLC state machine includes BUTTON_IN remaining true
G7
'Abort' transition of PLC state machine includes BUTTON_IN going FALSE
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results
Sn2
C/S State Machine

Figure 2.7. Example of goal structure (KELLY, 2003).

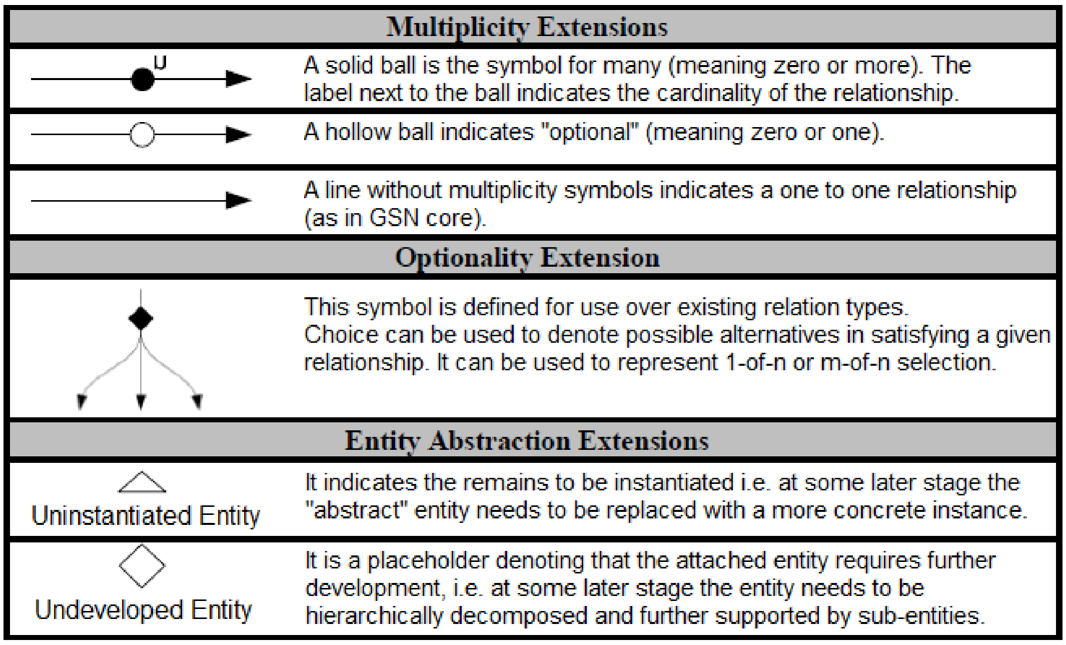
tion sub-goals, which are further supported by solution elements and other sub-goals. "*G2*" is supported by "*Sn1*" solution, referencing "*Black box Testing Results*" evidence item, and "*G5*" sub-goal. "*G3*" is supported by "*G7*" sub-goal. "*G4*" is a claim, which the reasoning is undeveloped in this argument structure. Finally, "*G5*" and "*G7*" are supported by "*Sn2*" solution element. Solution elements refer to artefacts produced throughout the development lifecycle, which provide the evidentiary support for claims.

By following the syntactic and semantic guidance provided by the GSN Standard (2011), clear and non-ambiguous hierarchical assurance cases can be created systematically, overcoming the limitations of free text and tabular argumentation approaches. GSN notation was further extended with patterns and modular extensions to support the assurance case reuse throughout product line development lifecycle. These extensions were used through this thesis to create assurance case patterns, and generating assurance cases for product lines instances with the support of model-based assurance cases technique (HAWKINS *et al.* 2015). GSN patterns and modular extensions are detailed in the following subsection.

2.4.2.2 GSN Patterns and Modular Extensions

The concept of assurance case pattern was created based on the principles of design patterns proposed by Alexander *et al.* (1977), and object-oriented software design patterns (GAMMA *et al.* 1995). Assurance case patterns define "*means of documenting and reusing successful safety argument structures*", by capturing argument structures considered to be common in constituting the backbone of an assurance case in a particular domain or across different domains (HABLI and KELLY, 2010; KELLY and McDERMID, 1997). GSN core was extended with two types of pattern abstractions, shown in Figure 2.8, to support the specification of assurance case patterns:

- **Structural abstraction**: it supports the specification of n-ary, optional, alternative relationships between GSN elements via multiplicity and optionality extensions; and
- **Entity abstraction**: it supports generalization/specialization of GSN elements by means of "*Uninstantiated*" and "*Undeveloped*" entity abstractions.



Multiplicity Extensions	
●ᵁ →	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 GSN core).
Optionality Extension	
◆	This symbol is defined for use over existing relation types. Choice can be used to denote possible alternatives in satisfying a given relationship. It can be used to represent 1-of-n or m-of-n selection.
Entity Abstraction Extensions	
△ Uninstantiated Entity	It indicates the remains to be instantiated i.e. at some later stage the "abstract" entity needs to be replaced with a more concrete instance.
◇ Undeveloped Entity	It is a placeholder denoting that the attached entity requires further development, i.e. at some later stage the entity needs to be hierarchically decomposed and further supported by sub-entities.

Figure 2.8. GSN pattern extensions (Adapted from Habli and Kelly, 2010).

GSN modular extensions were created to support the certification of highly-integrated, modular, and reconfigurable systems, e.g., systems developed based on Integrated Modular Avionics (IMA) (KEELY, 2007; BATE and KELLY, 2002), and safety-critical software product lines (HABLI and KELLY, 2010). GSN modular extensions support the development of "*modular and compositional assurance cases*", reducing the required effort for the reassessment of assurance cases after changes in the system. These extensions support the specification of an assurance case with well-defined and scoped argument modules, and the way as they are composed in a hierarchical argument structure. The specification of an "*Argument Module*" using the GSN modular extensions comprises (FEN *et al.* 2007; BATE and KELLY, 2002):

1. Goals (claims) addressed by the module;
2. Evidence (solution) referenced within the module;
3. Context defined within the module;
4. Goals requiring support from other modules;

Inter-module dependencies:

5. References to goals addressed in other modules;
6. References to evidence presented within other modules; and
7. References to context defined within other modules.

Inter-modules dependency elements 5, 6, and 7 respectively correspond to "*Away Goal", "Away Context",* and "*Away Solution"* GSN modular extension elements illustrated in the Figure 2.9. "*Away Goal"* is a "*goal reference"* used to support, or provide contextual backing for, an argument presented in one module, which the argument supporting that goal reference is presented in another module (HABLI and KELLY, 2010). "*Away Context"* is a reference to contextual information specified in other argument module. "*Away Solution"* is a reference to evidence information that is detailed in another argument module. "*Argument Contract"* aims to preserve the overall integrity of a modular assurance case when argument modules are modified. An "*Argument Contract"* is specified using interfaces from inter-related argument modules capturing "*rely-guarantee*" relationships between two argument modules (GSN STANDARD, 2011). Thus, *goals* requiring support from other modules (item 4), "*Away Goal", "Away Context",* and *"Away Solution"* elements in a module interface define the "*rely*" conditions, whilst the *goals* addressed by the module (item 1) define the "*guarantee*" conditions, i.e., "*context"* and "*evidence"* within the modules should be kept during the composition of two or more argument modules.

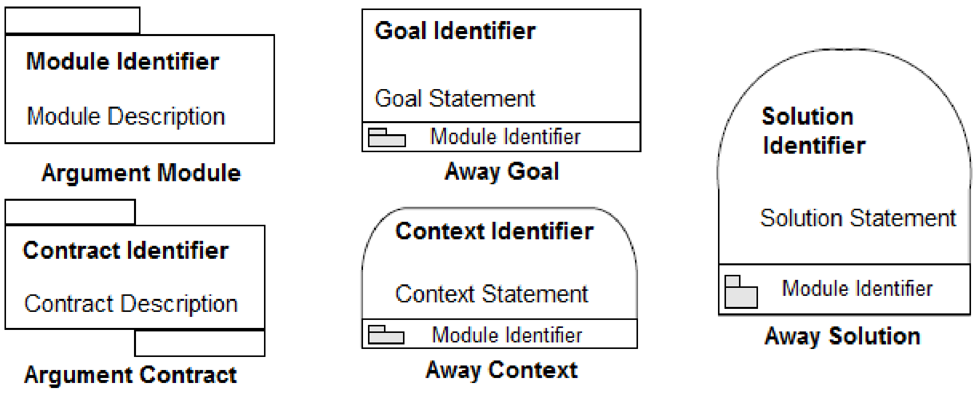


Figure 2.9. GSN modular extensions (HABLI and KELLY 2010).

In a product line assurance case, the following types of variation can be found: *intrinsic* and *extrinsic*. Intrinsic variation occurs when there is more than one argumentation style to support claims of a particular product line instance. For example, in the *Functional Decomposition* assurance case pattern (KELLY and McDERMID, 1997), shown in Figure 2.10, the safety of the system functions can be argued by one of the following argumentation styles: by arguing that "*interactions between system functions are non-hazardous*" *(G3)*; or by arguing that "*system functions are independent*" *(G4)*. The source of extrinsic variation is the reusable assets referenced in the assurance case such as feature, reference architecture, and failure models since these assets are expected to vary in how they are developed, composed

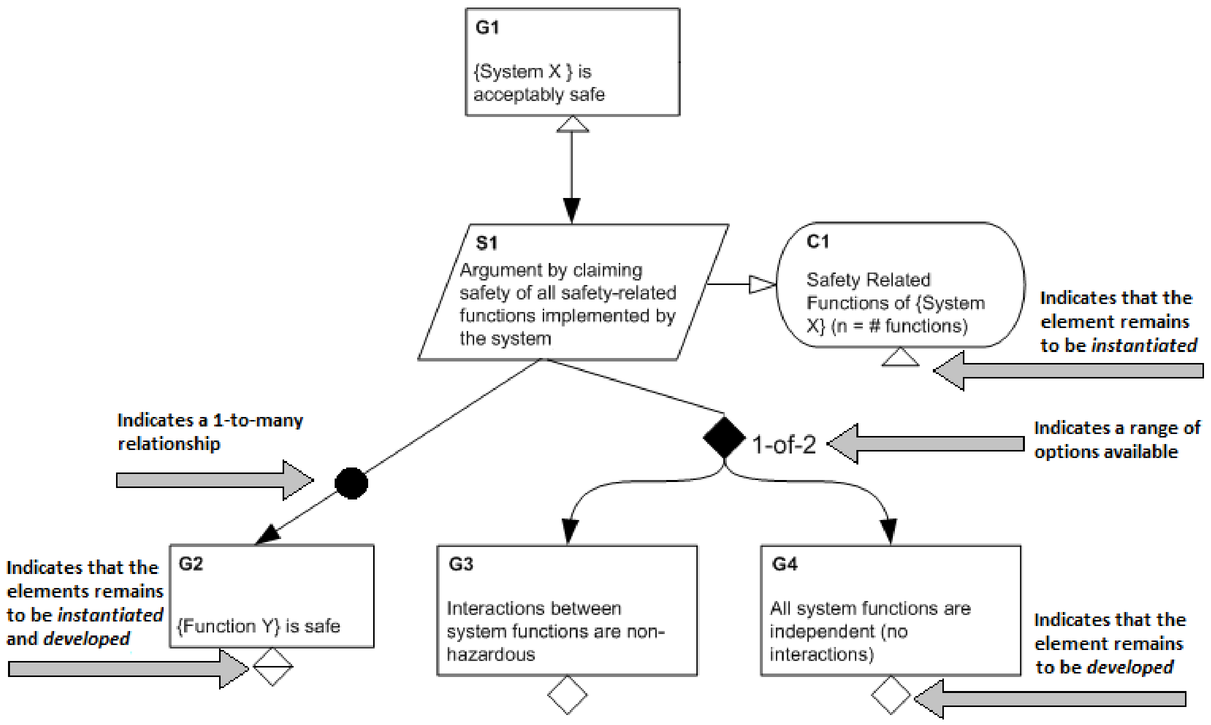

G1
{System X } is acceptably safe
S1
Argument by claiming safety of all safety-related functions implemented by the system
C1
Safety Related Functions of {System X} (n = # functions)
Indicates that the element remains to be *instantiated*
Indicates a 1-to-many relationship
1-of-2
Indicates a range of options available
Indicates that the elements remains to be *instantiated* and *developed*
G2
{Function Y} is safe
G3
Interactions between system functions are non-hazardous
G4
All system functions are independent (no interactions)
Indicates that the element remains to be *developed*

Figure 2.10. Functional decomposition assurance case pattern (Adapted from Kelly and McDermid, 1997).

and configured. Such variation may change the contributions of these assets to safety, consequently changing the way in which the safety of the system is justified in the assurance case.

Extrinsic variation can be found in un-instantiated GSN elements from *Functional Decomposition* pattern. The items in the curly brackets represent types of un-instantiated information (e.g., "*System X*" and "*Function Y*"). These items are associated with reference architecture and feature models that are the source of variation. Therefore, considering a hybrid braking system automotive product line (OLIVEIRA *et al.* 2014) discussed throughout this thesis, the instantiation of "*System X*" is restricted by seven architectural choices, e.g., "*Four Wheel Braking*", "*Front Wheel Braking*" or "*Rear Wheel Braking*" system variants, provided by the product line reference architecture model. Similarly, the instantiation of "*Function Y*" in "*G2*" goal is restricted by the number of wheel braking features associated with the given product line instance. The cardinality attached to the "*supported by*" relationship between "*S1*" and "*G2*" elements in Figure 2.10 indicates that an instance of "*G2*" will be created for each feature specified in the instance model. The manual instantiation of assurance case patterns and pattern elements from the system models, to create product line assurance cases may be time-consuming and error-prone. Therefore, traceability of extrinsic variation in product line assurance cases can be automated with the adoption of the model-based approaches (HAWKINS *et al.* 2015; DENNEY *et al.* 2015) to support the assurance case construction in product line engineering processes.

2.4.3 Model-Based Assurance Cases

Assurance case patterns capture the required structure of an assurance argument in a way that is abstract from the details of a particular argument. These patterns are used to create specific arguments by instantiating them according to the targeted application. An assurance case pattern defines the requirements in terms of the required information to instantiate assurance claims and evidence to support these claims (HAWKINS *et al.* 2015). Assurance case patterns are usually instantiated with information obtained from different sources of evidence such as design, assessment, analysis, and processes models or directly from an engineer. For high complex systems, the manual instantiation of assurance case patterns from the generated evidence can be time-consuming and potentially error-prone. Assurance cases for these systems are large and complex, with a great amount of explicit and implicit dependencies between assurance case pattern elements and the generated evidence. Therefore, the lack of integration with, and limited traceability to, development artefacts can undermine the confidence in the reasoning and evidence referenced in an assurance case. Existing model-based techniques (HAWKINS *et al.* 2015; DENNEY *et al.* 2015a; DENNEY *et al.* 2015; DENNEY *et al.* 2014; DENNEY *et al.* 2012; BASIR *et al.* 2008) support the automated traceability between development artefacts referenced in the assurance case and the assurance case itself, supporting the coevolution of the system design and the assurance case. It contributes to highlight weaknesses in the design, evidence, and argument early on the development lifecycle, improving the validity of the assurance case against the available evidence, which is a big challenge for assurance cases as stated in the Nimrod Accident Review (CAVE, 2006). Model-based assurance case approaches treat the assurance case as a model, bringing the benefits from the model-driven engineering such as automation, transformation, and validation for the assurance case construction process (HAWKINS *et al.* 2015).

The Model-Based Assurance Case (MBAC) (HAWKINS et al. 2015) is an approach to support the automatic generation of assurance cases based on assurance case patterns and model weaving (DEL FABRO *et al.* 2005). Model weaving links the reference information metamodels, i.e., design, assessment, and processes metamodels, to assurance case patterns. Model weaving is an approach to model transformation defined as an operation "*whose the* *primary objective is to handle fine-grained relationships between elements of distinct models, establishing links between them. These links are captured by a weaving model. It conforms to a metamodel that specifies the link semantics*" (DEL FABRO *et al.* 2005). Therefore, MBAC allows the interoperation between the assurance case, design, process, and assessment models and metamodels. MBAC supports the generation of assurance case models using the GSN notation that complies with the OMG Structured Assurance Case Metamodel (SACM) version 1.0 (OMG, 2013), but it can be adapted to support other notations such as Claim-Argument-Evidence (BISHOP and BLOOMFIELD, 2010). MBAC uses the models themselves to automatically instantiate assurance case patterns. In MBAC, the assurance case pattern instantiation involves both instantiating abstract elements, named "terms", in argument patterns, and making instantiation choices. In MBAC, the data required to instantiate assurance case pattern elements is automatically extracted from a diverse set of system models e.g., design and safety assessment models, based on mapping links defined in a weaving model. It ensures the traceability between the sources of information, e.g., in design, process and analysis models, and the assurance case. MBAC approach is tool and notation agnostic, placing no restrictions in the format of reference information models, it does not prescribes how, and with which tool the weaving model should be created, and it does not put restrictions on the adopted model management tools (HAWKINS *et al.* 2015).

A body of research on model-based assurance cases is focused on defining a formal basis for GSN arguments patterns and modules (DENNEY *et al.*, 2015; DENNEY *et al.* 2013; DENNEY and PAI, 2013; DENNEY and PAI, 2012). Denney and Pai propose a formal basis for GSN arguments (DENNEY and PAI, 2013) and patterns (DENNEY and PAI, 2012), and offer automated means, implemented in the AdvoCATE assurance case tool (DENNEY *et al.* 2012), to support the assembly of safety arguments and the instantiation of assurance case patterns. In both assembly and instantiation studies, the automatic generation of the assurance cases is based on an instantiation table that contains the data entries needed for populating the argument. Denney *et al.* (2013) have proposed the hierarchical assurance cases (*hicases*) technique to overcome the limitations that arise in manipulating large-size industrial assurance cases. *Hicases* consist in a basic hierarchical decomposition represented as indentations in a spreadsheet-based argument structure. *Hicases* work has evolved towards a formal basis for modular assurance cases, where the relationships between GSN intra-module abstractions (GSN STANDARD, 2011) were explicitly described and formalized (DENNEY *et al.* 2015; DENNEY *et al.* 2015a). Such formalization provides a rigorous basis for tool implementation, and it was preliminary implemented in AdvoCATE toolset.

A formal definition of an assurance case query language has also been proposed to support the management of complex and larger industrial assurance cases specified in Goal Structuring Notation (DENNEY *et al.* 2014a). Assurance case query language was implemented in AdvoCATE toolset, and illustrated in a fragment of assurance case for an unmanned aircraft system developed by NASA Ames. A conceptual framework for through-life cycle safety assurance, called dynamic assurance cases, intended to support the proactive safety management has also been established based on the concepts of assurance case query language. A formal basis for dynamic assurance cases has also been provided to support its implementation in model-based assurance case tooling (DENNEY *et al.* 2015). Similar to the MBAC approach (HAWKINS *et al.* 2015), dynamic assurance cases allow the coevolution of the system design and safety assessment artefacts, and the assurance case. Work on model-based assurance cases is also focused on a methodology to support the automated generation of assurance cases from formal verification (DENNEY and PAI, 2013a) and their integration with manually created assurance case fragments derived from system safety analysis (DENNEY and PAI, 2014).

Research in model-based assurance cases also covers a systematic approach to support the automatic generation of assurance cases from the information provided by formal verification (BASIR *et al.* 2008). The approach is based on a generic assurance case pattern that is instantiated with program-specific formal verification information. The approach is independent from the given assurance case argument pattern and program, and it is also independent of the underlying code generator. Such approach was evolved towards automatic integration of the outputs generated from AutoCert (DENNEY and TRAC, 2009) formal verification tool into an "*evidence-based argument"* (SUN, 2013), by encoding the argument reasoning in an assurance case pattern, and instantiating it using the outputs provided by formal verification (DENNEY and PAI, 2013a). This approach was implemented in AdvoCATE tool and used to generate an evidence-based argument for a real world unmanned aircraft system.

Unlike existing model-based approaches for assurance case construction (DENNEY *et al.* 2015; DENNEY *et al.* 2013; DENNEY and PAI, 2013; DENNEY and PAI, 2012), which the automatic generation of assurance cases is based on a table that should be filled with data entry needed for populating the argument, MBAC does not requires to predefine the instantiation data in a table. In MBAC, the assurance case pattern instantiation data is automatically extracted from a diverse set of system models, e.g., design and safety models, based on the mapping links captured in the weaving model. It contributes to ensure traceability between system models, and the assurance case, allowing the coevolution of the system design, safety assessment, and the assurance case (HAWKINS *et al.* 2015). In this thesis, the MBAC approach has been adopted to support the generation of assurance cases for product line instances into safety-critical software product line engineering. MBAC is built upon the GSN metamodel that complies with the OMG SACM specification, and Eclipse Modeling Framework platform (ECLIPSE, 2016). The MBAC approach is illustrated in Figure 2.11. GSN patterns, reference information models and metamodels (e.g., design, process and assessment models), and the weaving model are input artefacts for an instantiation program, which generates an instantiation model in compliance with GSN patterns, and it outputs a GSN assurance argument. The instantiation program performs the automated analysis of the mappings links between abstract terms defined in the GSN pattern and the required information to instantiate them provided by reference information models, specified in the weaving model. The weaving model is the core element of the MBAC approach. The following subsections detail each element of the MBAC approach.

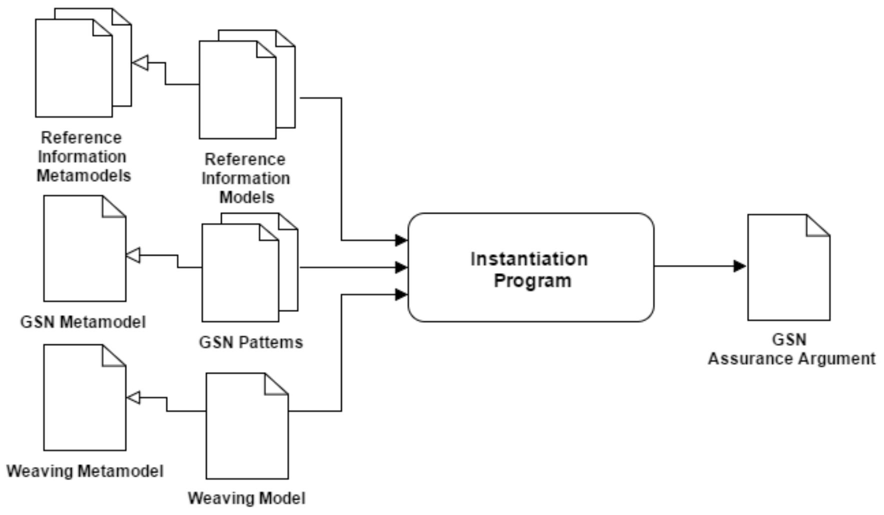


Figure 2.11. Model-based assurance cases approach (Adapted from Hawkins *et al.* 2015).

2.4.3.1 GSN Patterns

Assurance case patterns are intended to capture the structure of an assurance argument that abstract the details of an argument for a particular system. Assurance case patterns also define the required information to instantiate claims and the supporting evidence. Assurance case patterns can be captured using the GSN notation and its patterns extensions. The required information to instantiate assurance case patterns is provided by diverse source models such as design, assessment, and process models. The MBAC approach uses the models themselves to automatically instantiate these patterns, creating variant-specific assurance cases. It involves the instantiation of abstract "*terms*" specified in assurance case patterns, and the definition of instantiation choices. Abstract "*terms*" are instantiable entities that should be replaced with concrete information appropriate for the targeted system (HAWKINS *et al.* 2015). Abstract terms can be found in *Functional Decomposition* assurance case pattern shown in Figure 2.10. For example, the "*Function Y*" abstract term within "*G2"* claim, is represented in curled braces. *"Function Y"* must be replaced with the name of function associated with the targeted system. "*G2"* comprises a multiplicity relation, indicating that the number of required "*G2"* instances is determined by the number of functions associated with the given system. Thus, an instance of "*G2"* is created for each function specified in the system design. Such information can be provided by the system architecture model or feature model. Assurance case patterns can also represent instantiation choices for different argumentation approaches that may be adopted. "*G3"* and "*G4"* pattern elements defined in Figure 2.10 represent two argumentation approaches to argue that the system functions working together in a given environment are acceptably safe. Therefore, during the instantiation, the most appropriate argument approach for the targeted system is chosen. In MBAC, GSN assurance case patterns are considered as models that must conform to the syntax and semantics of the *GSN metamodel* (GSN STANDARD, 2011) defined in a graphical editor built based on the Graphical Modeling Framework (GMF) (HAWKINS *et al.* 2015). The editor outputs GSN pattern specifications in XML and *.gsnml* formats.

2.4.3.2 Reference Information Models

Reference information models represent a set of system models that provide the information required to instantiate assurance case patterns. These models represent different views of the targeted system, e.g., design, processes, safety assessment. Patterns specify the information required from different system models to instantiate their abstract "*terms*". Although these models are expected to be diverse, the MBAC approach only requires that each one of these models conform to a defined metamodel (HAWKINS *et al.* 2015). It is important to note that each different model intended to be used in the assurance case pattern instantiation must conform to a metamodel. This is necessary for mapping correspondence relations between information elements from these models, and abstract "*terms*" defined in GSN assurance case pattern models, in the weaving model.

Since an assurance case pattern is intended to provide a partial argumentation solution and does not typically describe the complete structure of a system assurance argument, an assurance case pattern specification for a system may be composed by a set of patterns that must be instantiated in sequence. For example, Kelly and McDermid (1997) have presented the *Hazard Avoidance* pattern, which decomposes the "system is acceptably safe" top-level claim into sub-claims arguing that the risk posed by each identified system hazard is acceptable; and Weaver *et al.* (2003) has presented a catalogue of argument patterns arguing the mitigation of component contributions to a particular hazard. These patterns can be used together for structuring an assurance case pattern for a given product line instance. Nevertheless, the instantiation of the most assurance case patterns may require information from a diverse set of system models. Considering the *Hazard Avoidance* pattern (KELLY and MCDERMID, 1997), regarding the failure behavior of the system, Table 2.6 shows the relationships between abstract "*terms*" and reference information models that provide information for instantiating them. For example, the system architecture model provides the "*system name*" information element required to instantiate the "*System X"* term, and "*hazard name*" information elements from the error behavior model can be used in successive instantiations of "*Hazard X"* abstract term. In the MBAC approach implementation proposed by Hawkins *et al.* (2015), reference information models and their respective metamodels should be specified using the EMF platform.

Table 2.6 – Abstract terms and reference information models.

Abstract Term	***Reference Information Model Element***	***Reference Information Model***
System X	System.name	System architecture model (e.g., AADL or Simulink model)
Hazard X	Hazard.name	Error behavior model (e.g., AADL Error Annex model or HiP-HOPS failure model)

It is important to highlight that for models that are informally defined, e.g., a textual hazard analysis document; the instantiation of abstract terms from patterns can be performed manually after the instantiation of other terms with the support of MBAC approach and tooling (HAWKINS *et al.*, 2015). In addition, inter-relationships between different models used in the assurance case pattern instantiation may exist. For example, error behavior model elements may be associated with components of the system architecture model. These inter-model relationships and mapping links between abstract terms and system models are captured in the weaving model.

2.4.3.3 Weaving Model

A weaving model represents correspondence relations in terms of elements from different models (DEL FABRO *et al.* 2005). Let's consider *M1 = (V, A)*, the GSN pattern model, and *M2 = (V', A')*, the error behavior model, and the given elements "*Hazard X*" ∈ *V* and "*Hazard.name*" ∈ *V'*; the mapping link between "*Hazard X*" and "*hazard.name*" elements is denoted by the triple *(Hazard X, Mw, Hazard.name)*, where *Mw* is the weaving model. The structure of the weaving model is defined in the weaving metamodel that defines the link semantics. Figure 2.12 illustrates the structure of the standard weaving metamodel (DEL FABRO et al. 2005) used in the MBAC approach. In this thesis, the standard weaving metamodel (DEL FABRO *et al.* 2005) is extended with the addition of "*Property*" entity to support linking the information contained in *attributes* from *reference information model* elements to abstract *terms* defined in assurance case patterns.

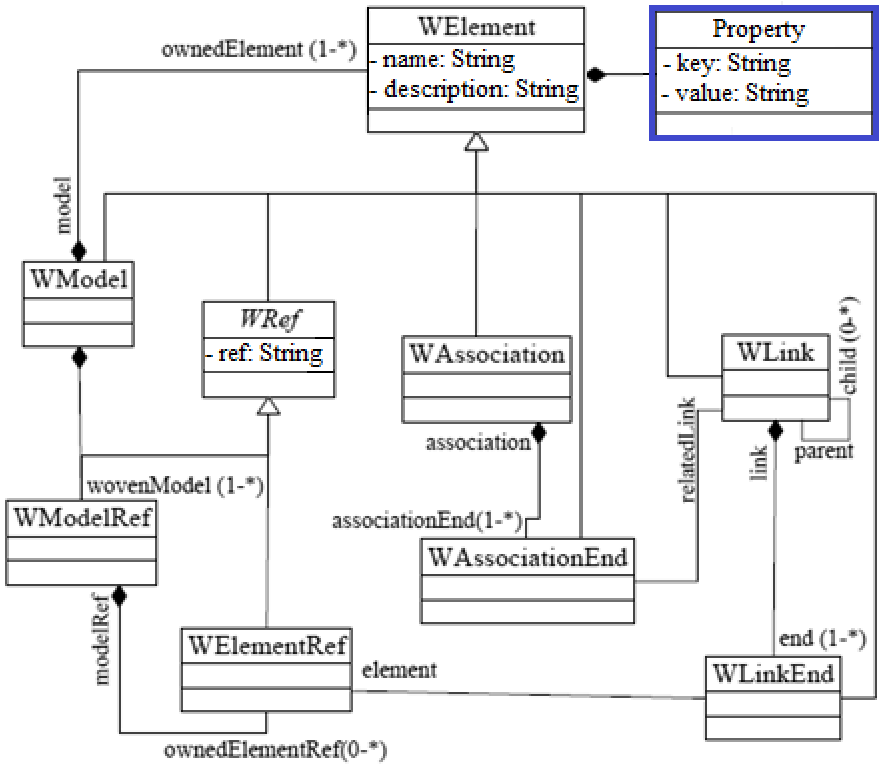


Figure 2.12. Weaving metamodel (adapted from Del Fabro *et al.* 2005).

In the following, a brief explanation of each weaving metamodel element shown in Figure 2.12 (DEL FABRO *et al.* 2005) is provided:

- ***WElement***: it is the base element of the metamodel, which is inherited by all other elements. It contains *name* and *description* attributes;
- ***WModel:*** is the metamodel root element. It is composed by weaving elements (WElement) and references to woven reference information models (WModelRef);
- **WLink:** it represents links between model elements. The *end* reference establishes links between an arbitrary number of elements. Weaving links can also be associated with other weaving links in a containment relation;
- ***WLinkEnd***: it indicates the extremity of a link that references the woven reference information model elements (i.e., right-hand side elements) through WElementRef;
- **Property**: it contains a pair “key/value”. It establishes the reference to an attribute from a reference information model element, instead of the whole object, required to instantiate a given abstract *term*. Therefore, a property is linked to, and associated with the *WLink* element, linking an abstract *term* to an attribute from the given *reference information model* element. Property allows specifying constraints between abstract terms, and reference information model elements involved in a model weaving operation;
- ***WRef***: is an abstract class that represents references. **WElementRef:** represents all referenced elements (right-hand side elements) of a woven model. The attribute *ref* contains the identification of the reference information model element (i.e., the model entity name). **WModelRef:** it references a metamodel/model being woven. It allows keeping track of woven metamodels/models. It contains references to model elements, (**WElementRef**);
- **WAssociation**: is intended to create association relationships between links;
- **WAssociationEnd:** it specifies the extremities of an association.

The weaving metamodel provides the minimal set of constructions to represent mapping links between models and association between these links, with no indication about what these links mean. Additional weaving semantics are expressed by weaving elements that extend the base weaving metamodel (DEL FABRO *et al.* 2005). Weaving metamodels provided by existing weaving modeling tools such as Atlas Model Weaver (AMW) (DEL

FABRO *et al.* 2005a) and yEd (yWORKS, 2016) define the semantics for weaving links. These tools provide graphical editors to support the specification of weaving models in conformance with a given weaving metamodel.

In the MBAC approach, the weaving model is used to capture the existing dependence relationships between GSN pattern abstract terms and elements from different reference information metamodels, and capturing the dependencies between a diverse set of system metamodels, e.g., architecture, fault tree, and the error behavior models used to instantiate GSN patterns (HAWKINS *et al.* 2015). The weaving model establishes the link semantics between metamodels that is valid for different instances. It allows using the same weaving model in multiple model transformations (HAWKINS *et al.* 2015a). In this thesis, it means that the same weaving model can be used to generate assurance cases for different product line instances.

2.4.3.4 Instantiation Program

The MBAC approach (HAWKINS *et al.* 2015) has been implemented over Eclipse Modeling Framework platform with the support of Epsilon (KOLOVOS *et al.* 2013) model management languages, which the following languages were used to develop the instantiation program:

- **Epsilon Object Language (EOL)**: an imperative language that allows manipulating, i.e., creating, querying, and modifying, models specified in the EMF platform;
- **Epsilon Generation Language (EGL)**: a template-based model-to-text language intended to generate code, documentation, and textual artefacts, e.g., a XML file from the analysis of a diverse set of models. EGL is used in the instantiation program to support the generation of XML and tabular GSN assurance case outputs.

The instantiation program requires GSN pattern models, reference information models and their respective metamodels, and the weaving model to generate an assurance case for the targeted system. Firstly, the program identifies the model elements that require instantiation in GSN pattern models; it determines which information is required to instantiate each abstract "*term"* defined in the GSN pattern by querying the weaving model; it obtains the information from the reference information models, and it outputs the instantiation information. In this thesis, the MBAC approach is used to support assurance case construction in product line engineering processes. The following section introduces product line engineering concepts and variability management techniques.

2.5 Software Product Lines

In the previous sections, safety assessment and assurance cases, and associated supporting model-based techniques and tools have been addressed in isolation from a particular development lifecycle. This thesis is focused on the provision of a model-driven approach to support the systematic reuse of functional hazard analysis and component failure analysis, and automated safety assessment and assurance case construction in safety-critical software product line engineering. This section presents an overview of software product line concepts, variability management, product line processes, and existing tool support.

A *Software Product Line* is defined as 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 (CLEMENTS and NORTROPH, 2001). *Features* represent the desired functionality from the user point of view (LEE *et al.* 2002). *Core assets* are reusable artefacts and resources that form the basis of a software product line. Core assets often include, but are not limited to, the architecture, reusable software components, domain models, requirements statements, documentation, specifications, performance models, schedules, budgets, test plans, test cases, and process descriptions (CLEMENTS and NORTHROP, 2001). In safety-critical product line engineering, reusable safety analysis and assessment artefacts are also part of the product line core assets (OLIVEIRA *et al.* 2016; OLIVEIRA *et al.* 2014; BAUMGART *et al.* 2014; DEHLINGER and LUTZ, 2009; DEHLINGER *et al.* 2007).

In a software product line, the architecture is the key asset among the collection of core assets. Therefore, a software product line can be seen as an architecture-driven development approach, based on a set of core assets produced in domain engineering process, and reused in application engineering process (CLEMENTS and NORTHROP, 2001; BOSCH, 2000; WEISS and ROBERT, 1999). Variability management is the key concept to enable the systematic reuse of core assets in product line engineering. Variability management identifies and controls how product line instances may vary in a given product line scope.

Variability establishes *"how the product line allows for and facilitates the differences"* (SVAHNBERG and BOSCH, 2000). With regard to variability and reuse, the product line architecture captures the commonalities and variability of the design of complex systems of a given domain, reducing the effort of further development of systems in the targeted domain. Product line architecture also has the role of delimiting the scope of variability by identifying points where product line instances can vary. Architectural variation typically occurs in functions, data, control flows, the underlying technology, environment and quality goals (BACHMANN and BASS, 2001). Variation is not limited to the architecture, but also exists in almost all product-line assets such as requirements, safety analysis that can vary from one product variant to another. Therefore, variability management is not straightforward; it requires explicit traceability of variation expressed in high level models such as feature model throughout low level models, e.g., design, safety analysis, verification, validation, and testing (POHL *et al.* 2005).

2.5.1 Product Line Variability Management

The development of a software product line is different from a conventional single system by the need to consider variability management during the whole lifecycle. Variability management distinct from configuration management by controlling variation over "space", i.e., variation between different product line instances, whereas configuration management controls variability over "time", i.e., different versions of the same artefact produced through the lifecycle (POHL *et al.* 2005). Variation in a software product line is manifested by means of *variation points*, which is the representation of variability in domain artefacts such as requirements, architectural design, implementation, and test plans where choices have been postponed until product instantiation (POHL *et al.* 2005; JACOBSON *et al.*1997). A *variant* represents a particular instance of variable artefacts specified in a variation point. A variant identifies a single option of a variation point and it can be associated with domain artefacts to indicate the artefacts that correspond to a particular option (POHL *et al.* 2005). A set of activities should be carried out to support variability management in software product line development lifecycle:

- **Variability identification:** it occurs during context and feature modelling. A feature model defines common, optional, and alterative functionalities of products in a particular domain (KANG *et al.* 1990). A context model defines variation in the environment associated with functional features, e.g., operating environment, hardware platform (HABLI, 2009). Additionally, new variation can be further identified in low level artefacts such as design and implementation (JARING and BOSCH, 2002; STEPHENSON *et al.* 2000);

- **Variability analysis:** it comprises technical and business analysis of the impact of variation. Technical analysis of variation considers the permitted selection of features and design strategies. On the other hand, business analysis involves the analysis of time and cost required for efficiently exercising and binding variations on design, implementation, and integration phases (BASS *et al.* 2004);
- **Variability representation:** it comprises the integration of variability into analysis and design models. For example, variability in product line feature model is represented by optional and alternative features (GOMAA, 2005);
- **Variability implementation:** it involves tactics and patterns related to modifiability and configurability used to implement variations. For example, in safety-critical system architectural design, variability patterns (WEILAND and MANHART, 2014; STEINER *et al.* 2013; BOTTERWECK *et al.* 2010) can be used to represent variability in the system architecture model; and
- **Variability maintenance**: as the requirements and consequently the design of the product-line core assets evolve, variation should be updated (BOSCH, 2000). It means that the variability model should be updated as products are added to, or removed from, the product line.

2.5.1.1 Product Line Variability Model

Since variability is the key concept of product line development that can occur in different levels of abstractions, e.g., requirements, design, implementation, testing, there are in the product line literature different proposals of metamodels to describe variability. Some proposals provide generic metamodels (BACHMANN *et al.* 2003; BECKER, 2003; GEYER and BECKER, 2002; BECKER *et al.* 2002; SALICKI and FARCET, 2001), and other more specific metamodels focused on managing variability in a specific level of abstraction such as viewpoints (AMERICA *et al.* 2004), design ((BASS *et al.* 2004; ZIADI *et al.* 2004; THIEL and HEIN, 2002; MUTHIG and ATKINSON, 2002; BACHMANN *et al.* 2001), and variability representation (GOMAA, 2005; LEE *et al.* 2002; KANG *et al.* 1990). Figure 2.13 illustrates the structure of a generic orthogonal variability metamodel (OVM). A variability metamodel essentially comprises *"variation point"* and *"variant"* concepts. A *"variation point"* describes places where variability can arise, and a *"variant"* provides a concrete instantiation for variable domain artefacts specified in a *"variation point"*, (POHL *et al.* 2005). Interdependencies between *"variation points"* and *"variants"* can be specified via "*requires*" and "*excludes*" constraints. For example, one "*excludes*" constraint can be used to indicate that the instantiation of one *"variation point"* restricts the instantiation of other *variation point*. A "*requires*" constraint can be used to indicate that the instantiation of a given *"variation point"* requires the instantiation of a specific *"variant"* from other *"variation point"*. In safety context, the selection of relevant operation modes and their criticality in the context model may determine the instantiation of certain variation points related to the level partitioning in the architectural design, specified in the feature model. It may impacts on how the instantiated architectural design can contribute to system level hazards identified in the safety model (HABLI, 2009), i.e., architectural variation might change hazard causes and allocated safety requirements to eliminate or minimize hazard effects.

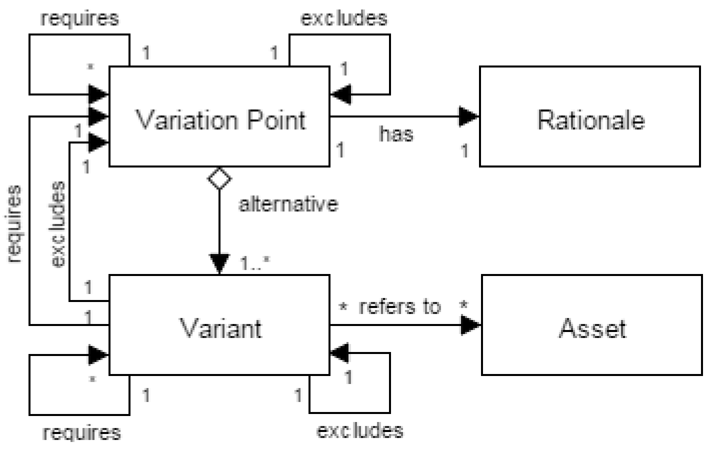


Figure 2.13. Variability model (adapted from Bachmann *et al.* 2003).

2.5.1.2 Product Line Feature Model

Feature modeling represents the product line variability at highest abstraction levels. Features express high-level functional and quality requirements of a given system architecture (POHL *et al.* 2005). A feature represents a distinct characteristic of the system visible to end-users. A feature model represents the standard features of a family of systems from a particular domain and relationships between them. Feature-Oriented Domain Analysis (FODA) (KANG *et al.* 1990) is the first and widely used feature modeling notation. FODA notation defines that a feature can be optional, alternative, or mandatory, and structural relationships between them such as decomposition, generalization, specialization, and parameterization. Figure 2.14 shows an example of feature model for a car product line, which "*horsepower*" is a mandatory feature, i.e., it is present in all cars, "*automatic*" and "*manual*" gears are alternative features, and air conditioning is an example of optional feature of a car. Constraints in the feature selection can be specified by means of composition rules. Whereas product line variation is not limited to features, being applicable to any product line asset, Gomaa (2005) has extended the UML with stereotypes to represent product line variation expressed in feature, requirements, design, behavioral, and implementation models showing how structural and behavioral models can be affected by variation points.

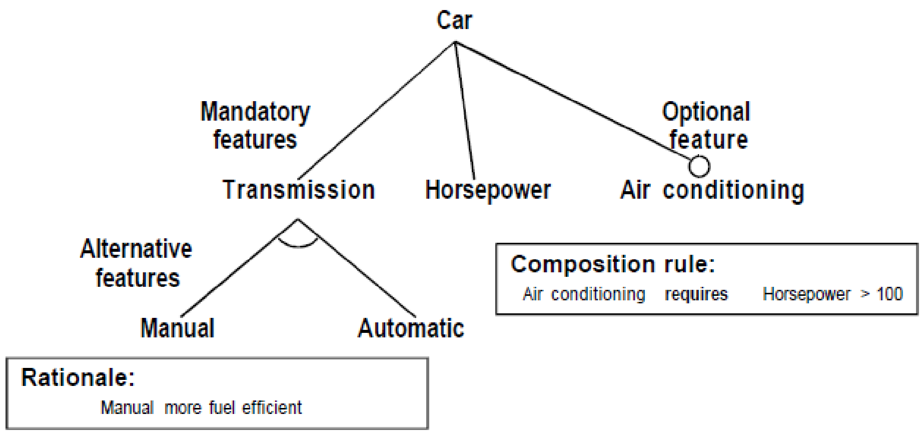


Figure 2.14. Feature model (KANG *et al*. 1990).

Context, feature and reference architecture models are the basis for the most product lines, even critical or non-critical (KANG *et al.* 1990). This is mainly because the context model defines the "*environment*" within which product-line instances may be deployed. The feature model captures the main system *functions* offered for reuse in a product line. The reference architecture model defines how architectural components are hierarchically organized to address the product requirements specified in the feature model, in the scope of environmental constraints defined in the context model. Specifically, the context model captures information regarding the structure of the physical, operating, support, maintenance and regulatory environment in the scope of product line instances (LEE *et al.* 2002; HABLI, 2009). The context model defines external constraints on the usage of functional features, i.e., system functions in product line instances, restricting the environment where functional features can be used. The context model provides the specification of contracts and dependencies between system features and their environment, which enable a clear understanding of the assumed behavior of product line instances (HABLI, 2009). From the perspective of safety-critical product lines, this is important to support the proper identification of variable failure behavior and allocated safety requirements during the product line safety analysis. The feature model provides an overview of functions from a particular domain and their interactions, which each feature represents a given functionality from the perspective of the stakeholders (CLEMENTS, 2002; JACOBSON *et al.* 1997). Conversely, features have explicit associations with other features, and with their operational environment, e.g., a functional feature can be deployed only in a specific environment, being not valid in other environment that requires, for example, redundant systems. Feature and context models capture the permitted functional and environmental variation in the product line scope. These models show how product line instance can vary in terms of functions and their nominal and failure behavior, according to the environment that these functions are deployed. Feature, context, and variability models provide abstractions to manage and trace product line variation throughout domain engineering and application engineering processes.

2.5.2 Product Line Processes

In software product line engineering, the core assets are developed during the product line domain engineering process and reused in application engineering (POHL *et al.*, 2005; VAN DER LINDEN, 2002). Product line domain engineering involves domain analysis, which identifies commonalities and variability in product line requirements. Domain engineering also involves the realization of variation by developing the product line architecture and implementing their components. These assets are typically stored in a repository, as illustrated in Figure 2.15. In product line application engineering, product line applications (instances) are created from the core assets defined in product line domain engineering (DEELSTRA *et al.* 2005). It exploits the product line variability and ensures the correct binding of variation points according to the product-specific requirements. Binding a variation point involves the choice of permitted/valid design options in the product line scope (JACOBSON *et al.* 1997).

After defining the product requirements and selecting the required product line features, the product architecture model is derived based on the product line reference architecture model. Thus, components are selected from the product-line asset base and configured according to the permitted variation established in the product line architecture. Since products may have specific requirements, after product derivation, product-specific components, which are not managed in the asset base, should be developed in the application engineering. The derivation of new products is not straightforward, requiring linking product

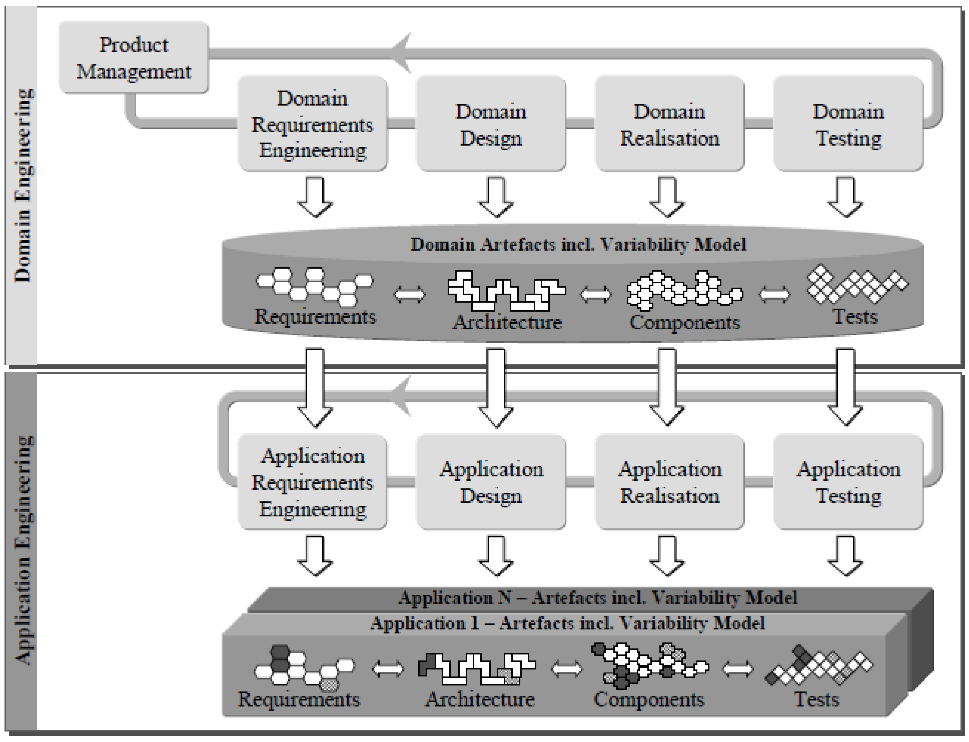


Figure 2.15. Product line processes (POHL *et al.* 2005).

line core assets such as design decisions, components, or test cases, to the rationale, assumptions, and processes behind their development. To enable the systematic reuse of product line core assets, these assets should be linked to the rationale embedded in common and variable features specified in the feature model. In safety-critical product line engineering, it involves linking functional and context features to their realization in architectural and safety analysis artefacts. There are in the literature a set of tools that provide automated support for variability management in product line assets. In this thesis, we have adapted some of these tools to enable the support for variability management in safety analysis models.

2.5.3 Variability Management Techniques

Product line variability is typically documented in two types of variability models: domain variability model, i.e., the feature model, and application variability model. In domain engineering, the product line variability is defined in the feature model. In application engineering, the variability defined in the feature model is bound to product line assets, e.g., architecture and components, in order to fulfill product-specific requirements (METZGER and POHL, 2014; HALMANS *et al.* 2008; POHL *et al.* 2005). The variability bindings for a specific application are documented in a respective application variability model. An application variability model establishes traceability links between application requirements, specified in the feature model, and domain artefacts. In the product line literature, application variability model is also known as variability realization model (HAUGEN and OGARD, 2014) or configuration knowledge (BEUCHE, 2012; KASTNER, 2008; RAATIKAINEN, 2005). Existing product line variability management techniques can be classified in two categories: integrated variability modeling and orthogonal variability modeling techniques (METZGER and POHL, 2014). In integrated variability modeling techniques, dedicated or specialized modeling concepts are introduced into existing modeling languages, e.g., the use of stereotypes to explicitly document variation points and variants in UML class diagrams. Integrated techniques are dedicated to document product line variability in the problem space (domain analysis), or directly in a specific product line asset such as a UML class diagram. On the other hand, in orthogonal variability management techniques the product line variability is documented in a dedicated model, i.e., the documentation of the product line variability is separated from the software development artefacts. Orthogonal variability management techniques relate the variability specified in the feature model with software development artefacts, documenting the realization of product line variability within product line assets. Orthogonal variability management techniques support the specification of variability in the solution space.

There are in the literature a number of extensions and modeling languages to support the specification of integrated variability, including annotations on use cases and test models (REUYS *et al.* 2006), stereotypes for UML class diagrams (HEUER *et al.* 2013; SHAKER *et al.* 2012), and domain specific languages (VOELTER and VISSER, 2011). Feature modeling is the most commonly used technique to support the documentation of product line variability (CAPILLA *et al.* 2013). After the introduction of FODA domain analysis technique (KANG *et al.* 1990), over 40 feature modeling dialects have been proposed (BENAVIDES *et al.* 2010). Some of these dialects are supported by graphical editors, e.g., FeatureIDE, Cardinality-Based Feature Modeling (CZARNECK *et al.* 2004). With regard to orthogonal variability, Base Variability Resolution (BVR) (VASILEVSKIY *et al.* 2015), Hephaestus (STEINER *et al.* 2013; BONIFÁCIO *et al.* 2009), Common Variability Language (CVL) (HAUGEN *et al.* 2008), GEARS (BIG LEVER, 2016), and pure::variants (PURE::SYSTEMS, 2016) are examples of existing orthogonal variability management techniques and tools. Specifically, BVR and CVL are concrete languages for orthogonal variability modeling under “standardization” within OMG. In this thesis we are interested in orthogonal variability management techniques and tools, especially BVR and Hephaestus. In this thesis, these tools were adapted to support the variability management in safety analysis models developed with the support of compositional safety analysis techniques, e.g., HiP-HOPS and AADL Error Annex. The following subsections present an overview of BVR and Hephaestus/Simulink variant management tools.

2.5.3.1 Base Variability Resolution

Base Variability Resolution (BVR) is a domain-specific language and toolset (VASILEVSKIY *et al.* 2015) to support the specification of variability on software product line MOF-based models (OMG, 2004). BVR language provides concepts to represent variability abstractions, mappings these abstractions to their realization in product line assets, and specifying operations to support product derivation. BVR language supports advanced feature modeling, reuse and variability realization on product line asset models. BVR tool chain comprises tools and graphical editors to support variability modeling on the domain problem and solution spaces. The editors and tools work integrated with stand-alone modeling tools. Whereas the BVR language defines variability orthogonally, BVR tool chain needs to communicate with third-party tools to define mapping links between product line assets, e.g., design, safety analysis, developed in a target language, and higher level variability abstractions specified in the feature model. Therefore, the BVR tool chain provides an abstract interface to enable the communication with third party model editors. The editors from the targeted languages should implement this interface to communicate with the BVR tool chain.

BVR defines variability orthogonally via feature trees and fragment substitutions, and it specifies means to yield a new product. Figure 2.16 illustrates the BVR variability resolution process. Product line engineers define substitution fragments on a base model by specifying how abstract features are realized in the target product line asset model (s). A base model is a model in which variability is defined in the BVR. A substitution fragment defines a set of elements to modify in a base model in order to derive a new product, i.e., places on the product line asset models that can vary. Further, these fragment substitutions are bound to product line features. Finally, the new product is derived by executing fragment substitutions according to a given instance (resolution) model. A resolution model is an abstract representation of a desired product configuration. In Figure 2.16, gray shaded features represent a car configuration, which the desired car has an engine with 110 horsepower (*hp-*

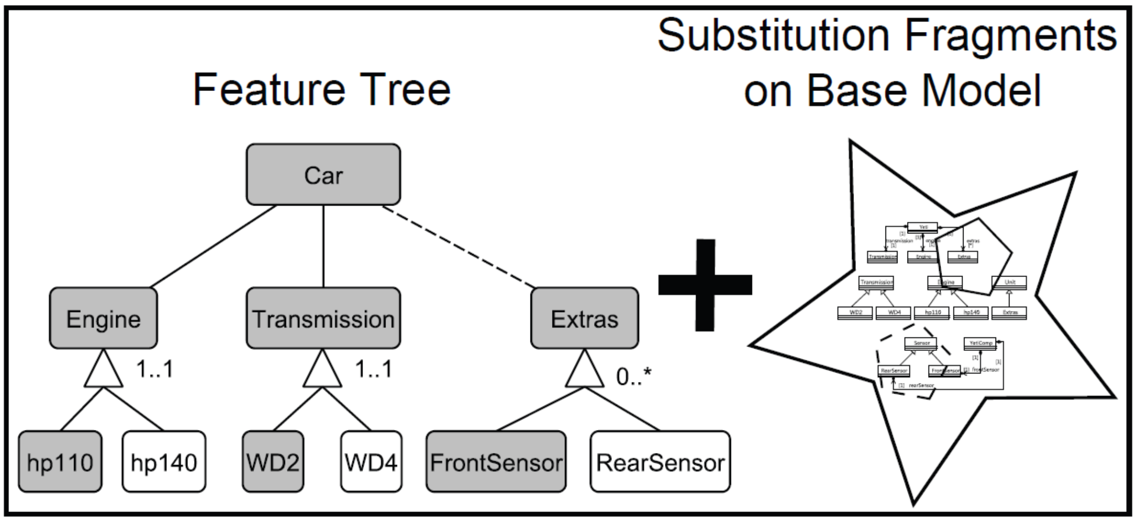
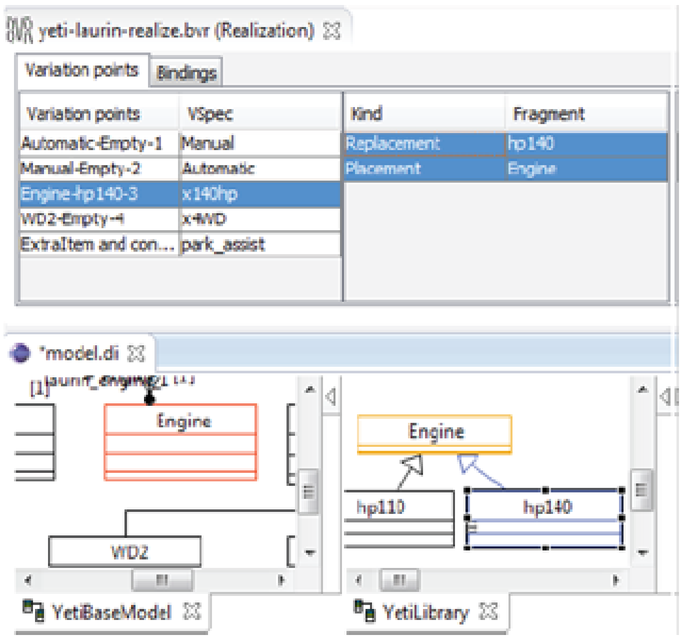


Figure 2.16. BVR variability modeling process (VASILEVSKIY *et al.* 2015).

110), two wheel drive transmission (*WD2*), and a front sensor (*FrontSensor*).

The BVR tool chain supports the following variability modeling activities: feature modeling (VSpec - Variability Specification Editor), mapping abstract features to concrete product line assets (Variability Realization Editor), and product derivation (Variability Resolution Editor and model transformations). BVR editors were developed upon the Eclipse Modeling Framework platform (ECLIPSE, 2016), which allows the seamless integration between these editors and third-party model editors. The BVR VSpec editor supports the product line feature modeling. The resolution editor supports the specification of product configurations by selecting features defined in the product line feature model. BVR variability realization editor allows the product line engineer to define the binding points, in the form of placement and replacement substitution fragments, linking product line asset models to abstract features. A fragment substitution removes from the base model elements specified in a placement fragment, and substitutes them with elements specified in a replacement fragment. To specify these fragments, the product line engineer selects elements in a base model that will be further modified to derive a new product. Thereafter, the product line engineer defines links between features specified in the feature model, and substitutions by attaching substitutions to features. Therefore, traceability links between abstract features and their realization in product line assets are established. Figure 2.17a illustrates an example of a mapping link between "*Engine-hp140"* feature from an automotive product line, placement and replacement fragments of a fragment substitution in the BVR variability realization model editor. When "*Engine-hp140"* feature is selected in a product configuration, the "*Engine"* class from the "*YetiBaseModel*" is removed from this model, and "*hp140"* class specified in



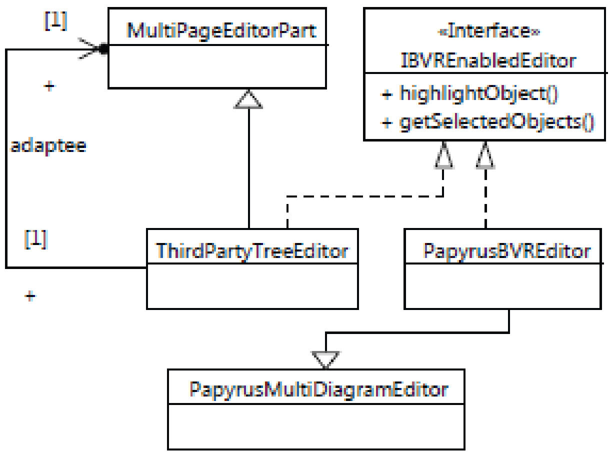
yeti-laurin-realize.bvr (Realization)

Variation points | Bindings

Variation points	VSpec
Automatic-Empty-1	Manual
Manual-Empty-2	Automatic
Engine-hp140-3	x140hp
WD2-Empty-4	x4WD
ExtraItem and con...	park_assist

Kind	Fragment
Replacement	hp140
Placement	Engine

(a)



(b)

Figure 2.17. (a) Features and fragment substitutions in BVR, (b) BVR third-party integration (VASILEVSKIY *et al.* 2015).

the "*YetiLibrary*" model is included in the base model to derive a new product.

The BVR tool requires the following input models: the variability specification model (feature model), the resolution model (instance model), variability realization model, and product line asset models e.g., UML class diagrams, safety analysis, developed using third-party editors that implement the "*IBVREnabledEditor*" interface provided by the BVR tool. By implementing this interface, third-party editors are linked to the BVR tool, as shown in Figure 2.17b. By default, BVR provides integration with Papyrus UML diagram editor and any EMF-based tree editors. To enable a third-party editor to be linked to the BVR tool chain it is necessary to create a class (*ThirdPartyTreeEditor*) that extends the "*MultiPageEditorPart*" class provided by the underlying EMF editor, and implementing the "*IBVREnabledEditor*" interface, by providing concrete definitions for "*highlighObject*" and "*getSelectedObjects*" abstract methods.

2.5.3.2 Hephaestus

Hephaestus is a variant management tool support for product derivation process, which comprises a suite of libraries that evaluates product line models and generates artefacts for specific product line instances. The current version of Hephaestus[6] supports variability management in requirements models, use case scenarios, source code, business processes, and component models (STEINER *et al.* 2013; BONIFÁCIO *et al.* 2009). Hephaestus provides a graphical interface that allows application engineers to select the input models for product derivation, i.e., product line feature model (FM), instance model (IM), configuration knowledge (CK), and product line asset (PLA) models. The feature model describes common and variable features of a product line. The IM represents the set of features of a given product configuration. PLA represents configurable artefacts of a product line, e.g., requirements, design, component models. The CK maps product line feature expressions to transformations to be performed in product line assets during the product derivation process (BONIFÁCIO *et al.* 2009). The configuration knowledge provided by the Hephaestus allows associating feature expressions to extensible transformations. Therefore, the Hephaestus configuration knowledge structure does not need to be changed when new transformations are defined in the tool to support variability management in other models. The Hephaestus product derivation process starts with the evaluation of each feature expression specified in the configuration knowledge against selected features in the instance model. In the following, all transformations associated with feature expressions evaluated as true are applied to the PLA assets during the product derivation process. These transformations select or modify parts of the asset models for the product that is being derived.

Hephaestus has been further extended to support variability management in MATLAB/Simulink models (STEINER *et al.* 2013). In this extension, the configuration knowledge associates each feature expression with a set of pairs, comprising a set of block identifiers, and a transformation to be applied to these blocks. The following two transformations have been implemented in Hephaestus to enable variability management in the Simulink model: "*selectSimulinkBlock*", which defines the block identifiers that will be selected for product derivation when a given feature or a group of features is selected in the instance model, and "*clearVariabilityMechanism*", which is associated with model blocks that implement variability mechanisms, i.e., similar to the concept of "*placement fragment*" from the BVR language (HAUGEN and OGARD, 2014). Firstly, "*selectSimulinkBlock*" transformations are executed to introduce into the instance model all blocks and their connections related to the selected features. After evaluating all "*selectSimulinkBlock*" transformations, "*clearVariabilityMechanism*" transformations are executed (STEINER *et al.*

[6]https://github.com/hephaestus-pl/hephaestus-base

2013). This transformation removes a block marked as the variability mechanism and connections associated to it from the Simulink model, and it reconnects this block connected to its entry in the block connected to its output.

Figure 2.18 shows an example of Hephaestus "*selectSimulinkBlock*" and "*clearVariabilityMechanism*" transformations applied to a simple Simulink model (Figure 2.18a). The "*Switch*" block is the variability mechanism in this model, and then it is associated with the "*clearVariabilityMechanism*" transformation. "*Input Kind A*" and "*Out1*" blocks are associated with "*selectSimulinkBlock*" transformation. This figure illustrates how Hephaestus applies the "*clearVariabilityMechanism*" transformation. The "*Switch*" variability block is removed from the model (Figure 2.18b), and the selected block connected to the input port from the block used as variability mechanism (i.e., "*Input Kind A*") is connected to its output block (i.e., "*Out1*") (Figure 2.18c). Thus, product-specific Simulink models generated with the support of Hephaestus only contain blocks related to the selected features.

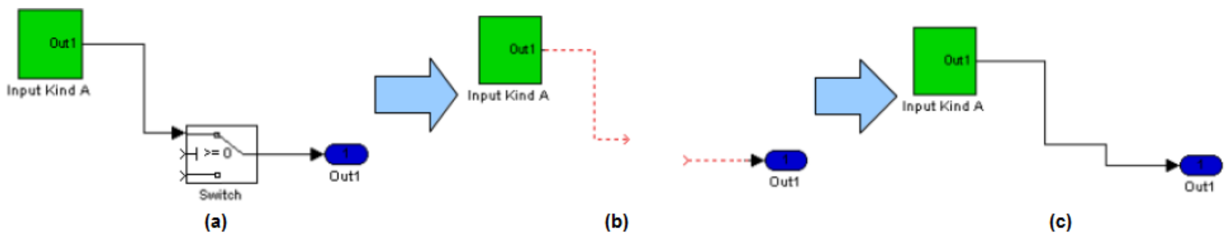


Figure 2.18. *ClearVariabilityMechanism* transformation in the Simulink model (STEINER *et al.* 2013).

Extending Hephaestus to support variability management in other product line assets requires the creation of data types and a parser for the underlying language of the targeted model, and writing model transformations (TURNES *et al.* 2011). This is necessary to support the generation of product-specific models during the product derivation where the selected model elements are included in the new product model, and model elements representing variability mechanisms are removed from the product model. Therefore, it is not necessary changing the structure of the configuration knowledge (i.e., the variability realization metamodel). The product derivation process of both BVR and Hephaestus/Simulink tools generate product models only with the respective model elements associated with the selected features. From the perspective of safety-critical systems development, this is important to guarantee the generation of product-specific architecture and implementation models without "dead" or "deactivated" elements that may lead to design mismatches, which can introduce unexpected errors on system models derived from a reusable system architecture model. This issue is addressed in the following section that presents a review of the related research on techniques to support variability management, safety assessment, assurance case construction in product line engineering, and certification of reusable components.

2.6 Existing Work on Safety Assessment, Assurance Cases, and Certification of Safety-Critical Product Lines

Safety standards have acknowledged the economic relevance in considering the reuse of previously developed components (RTCA, 2011; ISO, 2011; EUROCAE, 2010). In automotive domain, ISO 26262 have considered the use/reuse of commercial of the shelf (i.e., out-of-context) components in the development of automotive systems, and the development of reusable software components aimed to reduce the production costs. In aerospace domain, Federal Aviation Administration (FAA) has created the AC-120: Reusable Software Components advisory circular (FAA, 2004), to offer means to address the requirements of the aerospace software guidance DO-178C (RTCA, 2012) regarding reusable software components. Most safety standards introduce additional constraints on development, verification, integration and maintenance of reusable software components in order to ensure the safety of the overall system is not compromised by incorporating reusable components. For example, when a reusable component was not developed to the safety integrity level of a new system, reverse engineering and application of more rigorous techniques may be required. Although most safety standards do not explicitly address software product lines, they have considered the need of reusing previously developed artefacts. Safety standards often require additional activities to assess the impact of a reusable component on the safety of the system (HABLI, 2009).

With the advances and experiences in using model-based development and verification techniques, especially in the development of systems in aerospace and automotive domains, safety standards have also been considered the incorporation of these techniques in safety-critical system development lifecycle. This is justified by the fact that models may bring benefits of: unambiguous expression of requirements and architecture; the provision of automated support for code generation, verification, e.g., compositional model-based safety assessment supporting the generation of fault trees and FMEA artefacts, and simulation. Therefore, RTCA have published the "*Model-Based Development and Verification*

Supplement to DO-178C" document that provides guidance for incorporating model-based development and verification techniques in the development of aerospace software. In addition, SAE ARP 4754A has recognized the use of model-based techniques to support the safety assessment process. In automotive domain, ISO 26262 has recognized the use model-based techniques to support architectural design and formal verification guidance required to address stringent ASILs. The following sections presents a review of research on product line safety assessment, assurance cases and model-driven development.

2.6.1 Product Line Safety Assessment

Research in safety-critical product lines comprises safety assessment and assurance cases. Initial research efforts in product line safety assessment have focused on adapting traditional safety analysis techniques, such as FTA and FMEA to suit product line processes. The most notable work on this topic is the extension of Software Fault Tree Analysis (SFTA) to address the impact of product line variation in safety analysis (DEHLINGER and LUTZ, 2006; DEHLINGER and LUTZ, 2004; FENG and LUTZ, 2005). This approach is based on a technique for the development of a product line SFTA in the domain engineering phase, and a pruning technique to reuse such SFTA for the analysis of product line instances. It offers a systematic approach to treat SFTA results as a reusable asset. The product line SFTA approach enables semi-automated generation of product fault trees, by means of a pruning technique, with the support of domain expert reviews, improving the confidence in the resultant fault trees. This approach was later extended to integrate product line SFTA results with product line requirements (DEHLINGER *et al.* 2007) and state-based models (LIU *et al.* 2007). Such integration supports the generation of reusable test scenarios for examining the validity of the design. The SFTA approach is hierarchical and not sensitive to variation that distinguishes different product line instances, i.e., variation in architectural components and component failure modes. These studies consider fault trees as a reusable asset, managing variation at fault tree and FMEA models that can be auto-generated from Functional Failure Analysis (hazard analysis and component failure logic).

Further research in product line safety assessment (HABLI, 2009) overcomes the limitations from product line SFTA approaches related to the lack of mechanisms to manage the impact of product line variation on fault tree gates (DEHLINGER *et al.* 2007; STEPHENSON *et al.* 2004). Therefore, a catalogue of metamodels addressing the traceability of product line variation throughout safety assessment and assurance case assets, named

"*Product Line Safety Metamodel"*, has been proposed (HABLI, 2009). This catalogue is mainly composed by three safety assessment metamodels that define the conceptual structure of the relationships between product line commonalities and variability with safety assessment models at different levels of abstraction. At the functional abstraction level, the Functional Failure Metamodel (FFM) defines relationships between common and variable product line features and contexts with system hazards and contributing component failure modes identified during product line safety analysis, e.g., using Functional Failure Assessment technique. The FFM define the structural representation for product line functional failure assessment.

At the architectural level, the Architectural Failure Metamodel (AFM) links common and variable product line systems/subsystems/components to gates and leaf nodes of a fault tree. Finally, at the detailed design, the Component Failure Metamodel (CFM) defines the structure of the relationships between product line components, their failure modes and the effects of these failure modes on the overall safety (i.e., Failure Mode Effects - hazards). CFM establishes relationships between product line components, their failure modes and the contribution of these failure modes to the occurrence of system hazards in a FMEA fashion. Figure 2.19 shows the AFM elements and their relationships with fault tree elements. The "*Condition"* and *"Failure Mode"* elements represent events of a fault tree, whereas "*Causal Relationship*" elements correspond to fault tree gates. Fault tree *basic events* can be specified as "*Condition*" or "*Failure Mode*" elements with no identified causes. Figure 2.20 illustrates an example of relationships between common and variable product line assets and elements of a fault tree. Figure 2.20a illustrates the relationship between "*Late response*" fault tree event

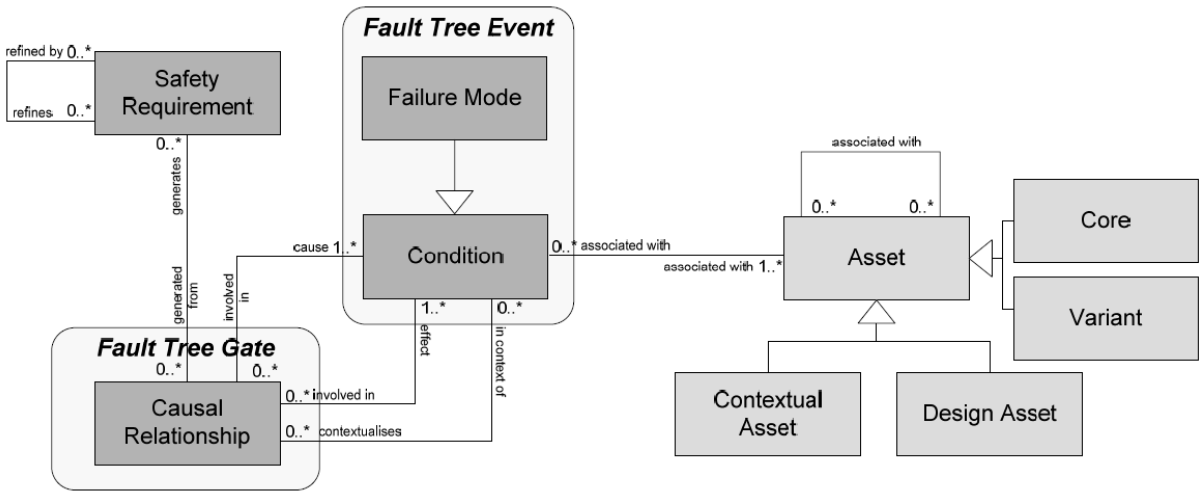


Figure 2.19. Architectural failure metamodel and fault tree elements (HABLI, 2009).

and "*Asset X*" product line asset. It indicates that the instantiation of "*Asset X*" is required for the "*Late response*" event exist. Whereas the choice of fault tree gates is tightly associated with product line context and architecture models, configuration choices on these assets can determine the lower-level failures that may contribute to an upper-level failure. Figure 2.20b shows the impact of configuration choices, expressed by a fault tree *qualifying condition*[7], on the relationships between a fault tree gate ("*Causal Relationship*") and component failure modes (fault tree events). The fault tree *qualifying condition* is linked to core and variable design and contextual assets, representing a valid configuration. This configuration influences the causal relationship between failure modes and conditions, defining the *structure* of the fault tree (HABLI, 2009). In this example, the qualifying condition determines that the "AND" gate is present in a product-specific fault tree whether an instance of "*Asset Y*" (*design asset*) is allocated to processors 1 and 2 (*contextual asset*), otherwise the structure of such fault tree does not include the "AND" gate. Therefore, the contribution of basic events to upper fault tree events may depend upon *variable* design or contextual assets associated with an upper gate. It confirms the assertion that the impact of product line variation can be *"dispersed throughout the tree*" (DEHLINGER *et al.* 2007).

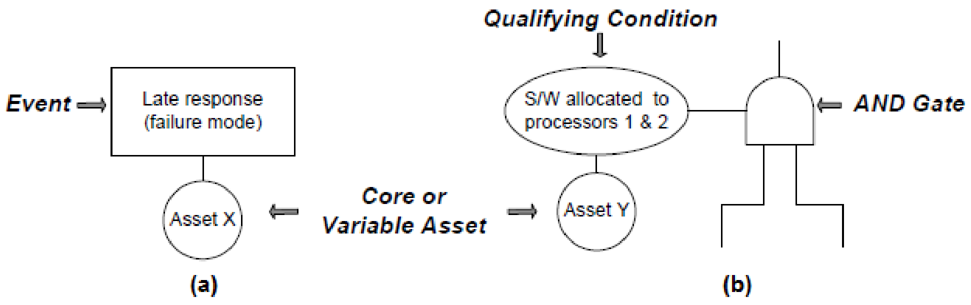


Figure 2.20. Relationships between product line assets and fault tree elements (HABLI, 2009)

The "*Product Line Safety Metamodel"* provides conceptual relationships to support the traceability of product line variation at different levels of abstraction throughout safety assessment models. It can be said that the "*Product Line Safety Metamodel"* provides a conceptual framework to support the management of the impact of product line variation throughout development, safety assessment and assurance case assets. These metamodels were instantiated and validated in case studies from aerospace domain. However, in the same way as product line SFTA approaches, the "*Product Line Safety Metamodel"* considers the management of variability in fault trees and FMEA assets, which can be generated from the

[7] It is a condition that should be satisfied before the input produce the output (NASA, 2002).

Functional Failure Assessment. This thesis fills the gaps left by previous research with regard to automated tool support and instantiation of the concepts defined in the "*Product Line Safety Metamodel"* in a systematic model-based safety assessment and assurance case construction approach for product lines (Chapters 3, 4, 5, and 6), supported by the integration of existing variant management, compositional safety analysis, and model-based assurance cases techniques. Thus, the automated traceability of variation throughout development, assessment, and assurance case assets produced across domain engineering and application engineering processes is achieved. Therefore, changes in the product line reference architecture and context are automatically propagated throughout safety assets and the assurance case.

2.6.2 Product Lines and Assurance Cases

The "*Product Line Safety Metamodel"* also defines conceptual relationships for linking variation on safety arguments and evidence to variation on product line feature, context, and reference architecture models. Thus, Habli and Kelly (2010) have proposed an approach to support the development of reusable product line assurance cases using GSN patterns and modular extensions. In such approach, GSN patterns extension is used for capturing variation in assurance cases and tracing them to their extrinsic source in the reference architecture model. GSN modular extension is further used for structuring the assurance case into loosely coupled core and variable modules using argument contracts. It supports the reuse of the product line assurance case, allowing the derivation of a compelling, comprehensive, and traceable assurance case for an individual product line instance. Variation in product line assets can impact in the structure of an assurance case. It may change the information that could be referenced in a product-specific assurance case, e.g., product definition and operating environment contextual elements, identified hazards and risk classification, the mitigation measures considered to minimize hazard effects, and risk mitigation decisions. Therefore, product line assurance cases should be highly reconfigurable in order to support the derivation of an assurance case for each product variant derived from the product line core assets.

Figure 2.21 illustrates the use of GSN patterns and modular extensions to represent extrinsic variability and structuring a product line assurance case. GSN argument modules and contract are used to contain the impact of design variation in "*Function X*" top-level argument

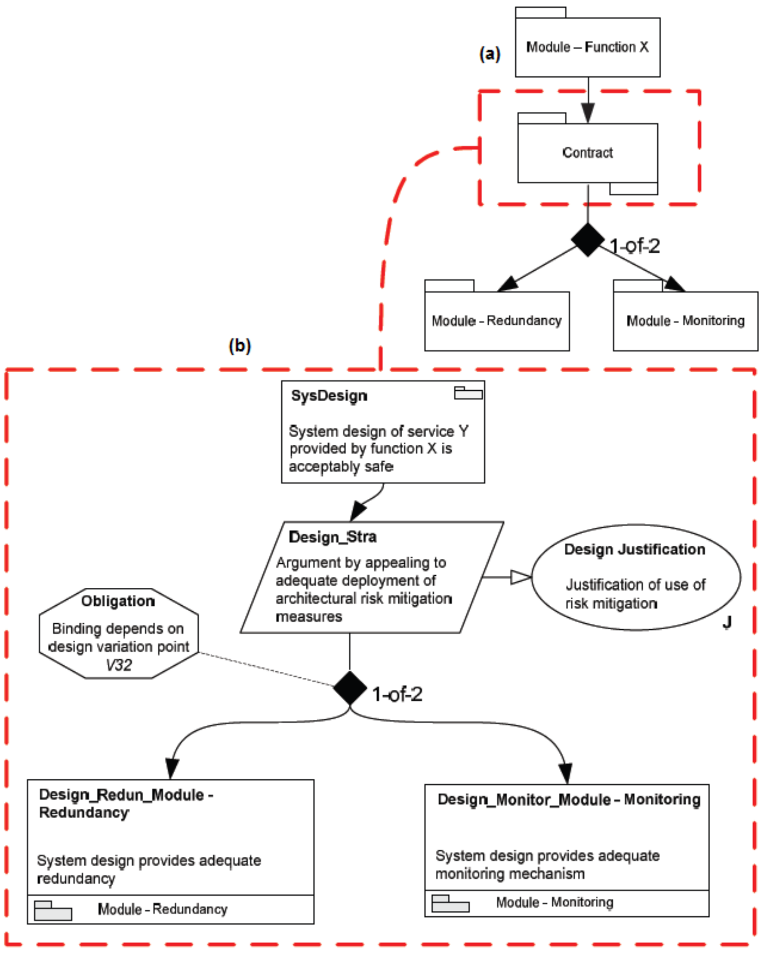

(a)
Module – Function X
Contract
1-of-2
Module - Redundancy
Module - Monitoring
(b)
SysDesign
System design of service Y provided by function X is acceptably safe
Design_Stra
Argument by appealing to adequate deployment of architectural risk mitigation measures
Design Justification
Justification of use of risk mitigation
J
Obligation
Binding depends on design variation point V32
1-of-2
Design_Redun_Module - Redundancy
System design provides adequate redundancy
Module - Redundancy
Design_Monitor_Module - Monitoring
System design provides adequate monitoring mechanism
Module - Monitoring

Figure 2.21. Design variation and argument contract (HABLI and KELLY, 2010).

module, which can be supported by either "*Redundancy*" or "*Monitoring*" alternative modules. So, if a new optional module is later added to support the "*Function X*" module, the change is contained within the argument contract and not propagated throughout the top-level module. Figure 2.21b shows the internal structure of the "*Function X*" module. "*SystemDesign*" is the public interface from this module, which is supported by an argumentation strategy based on either a claim of adequate redundancy, provided by "*Redundancy*" module, or a claim of adequate monitoring, provided by "*Monitoring*" module. The way that "*SysDesign*" is supported is dependent upon the binding of "*V32*" design variation point, i.e., denoted by the "*Obligation"* element attached to the GSN choice, which is encapsulated in the argument contract.

Research in assurance cases has evolved towards model-based assurance cases with focus on model weaving (HAWKINS *et al.* 2015) and formal verification techniques (DENNEY *et al.* 2015; DENNEY *et al.* 2015a; DENNEY *et al.* 2013; DENNEY and PAI, 2013; DENNEY and PAI, 2012). These techniques support the generation assurance cases based on the definition of assurance case patterns, an instantiation program, and a diverse set of system models, e.g., development, assessment, and processes models. Particularly, research in product line assurance cases has evolved towards a model-based approach to support the automated generation of product line assurance cases built upon the "*Product Line Safety Metamodel"*, argument modules and contracts (OLIVEIRA *et al.* 2015; OLIVEIRA *et al.* 2013), and Model-Based Assurance Cases (MBAC) (HAWKINS *et al.* 2015).

Oliveira *et al.* (2015) approach provides systematic guidance to support the generation of modular and reusable product line assurance cases from the assurance case patterns, an assurance case modeling notation, development and safety assessment models. The approach implicitly applies the concepts from model weaving technique (DEL FABRO *et al.* 2005), and it defines five steps to support the generation of product line assurance cases from development and assessment models: *1) Functional Failure Modeling*, which consists in generating multiple FFM instances from the information provided by product line reference architecture and failure models created with the support of model-based techniques; *2) Design of the Modular Assurance Case*, which defines how product line variation expressed in the feature model is represented in a modular assurance case architecture; *3 & 4) Architectural and Component Failure Modeling* produce multiple instances of AFM and CFM from the information provided by fault trees and FMEA results; and finally, *5) Argument Module Design* delivers the internal structure of each argument module, defined in the modular assurance case architecture, from the information provided by architecture and component failure models. Whereas the assurance case is an artefact that can be generated from a diverse set of system models, where variability is already resolved, the MBAC approach and its EMF-based tool support can be used to generate variant-specific assurance cases. Thus, Oliveira *et al.* (2015) approach has evolved towards MBAC to support the generation of assurance cases for product line instances (Chapter 6).

Since safety standards have considered the systematic reuse of third-party (out-of-context) components to cope with the reduction of the time-to-market and production costs, research in safety-critical component reuse has extended the traditional contract formalism to the notion of safety contract. A safety contract captures safety-related information, e.g., functional safety requirements, to support the systematic reuse of safety-critical software components (SLJIVO *et al.* 2013). A safety contract specifies strong and weak assumption/guarantee contracts for out-of-context reusable components. A strong contract captures information that need to hold to allow component reuse in all contexts, i.e., that are out-of-context, and weak contract captures more context-specific information. A safety contract comprises strong assumptions and guarantees, and multiple weak assumption/guarantee pairs. Strong assumptions/guarantee must be always satisfied to enable the component reuse, whereas weak assumption/guarantee pairs offer the information, besides strong assumptions, that need to hold to reuse the component in specific contexts. Thus, safety contracts allow specifying strong assumptions about the environment (A) and functional safety requirements (G) that should be addressed to reuse the component in all contexts, and weak assumptions (B) and safety requirements (H) that should be addressed to reuse the component in specific contexts. Strong and weak contracts distinguish between properties that hold for all contexts and those that are context-specific.

In order to reduce the cost to achieve safety certification of reusable components, an approach to support the automatic generation of argument-fragments for out-of-context components from assumption/guarantee safety contracts has been proposed (SLJIVO *et al.* 2014). Safety contracts capture safety claims related to the component being reused, including safety requirements, and supporting evidence. The approach provides a set of rules to support the automatic generation of argument-fragments. Such approach was further evolved towards a method to support: the derivation of safety contracts for components from the failure logic analysis results (safety analysis), and automated generation, based on the "*Absence of Hazardous Software Failure Mode*" assurance case pattern (WEAVER, 2003), of argument-fragments that include evidence related to the safety analysis (SLJIVO *et al.* 2015). Research in safety contracts also comprises the definition of a conceptual framework to support the management of safety properties of components in order to guarantee their reuse in different contexts. In the context of this thesis, safety contracts are defined on the product line variability realization model that describes how assumptions about different usage contexts in which product line components can be reused are linked to architecture and component failure logic models (OLIVEIRA *et al.* 2014). Thus, context-specific architecture and component failure logic are generated during the product derivation phase. In addition, with the support of model-based techniques, context-specific assessment artefacts, e.g., fault trees and FMEA, and assurance cases can be auto-generated from the reused architecture and safety analysis models.

2.6.3 Product Line Safety Assessment and Model-Based Development

Research in product line safety assessment and model-based development have focused on safety analysis approaches based on product line feature model and design simulation (SIERLA *et al.* 2015), model-based approaches (DOMIS *et al.* 2015; SCHULZE *et al.* 2013) and processes (KÄβMEYER *et al.* 2015; LANDUYT *et al.* 2014) to support the management of the impact product line variation on safety models. Schulze *et al.* (2013) have integrated feature models in pure::variants with elements of architecture, fault trees, FMEAs, and safety concepts in medine tool. Whereas both tools are based on the EMF platform, these tools are integrated in a single Eclipse product, which uses the EMF feature mapping extension from pure::variants to reference safety artefacts created with the support of medine toolset. Such approach provides mechanisms to manage variability in fault trees and safety goals model elements. In this approach, design and safety analysis information are stored in the architecture model, in this case a SysML model. This approach was further evolved towards integration of pure::variants within Component Integrated Component Fault Trees (C_2FT) (ADLER *et al.* 2010; DOMIS and TRAPP, 2008) models, allowing handling variability of C_2FTs integrated with variability in components (DOMIS *et al.* 2015). Similarly to HiP-HOPS and AADL Error Annex compositional safety analysis techniques, Domis *et al.* approach's is based upon a UML model enhanced with C_2FT annotations, supported by a C_2FT UML profile. This approach can be used in any UML tool that supports profiles, and it can also be used together with other variant management tools such as GEARS (BIG LEVER, 2016), or BVR (VASILEVSKIY *et al.* 2015).

A model-based approach to support change impact analysis, which integrates model-based development, product line engineering, and safety engineering (KÄβMEYER *et al.* 2015), has been proposed in the context of "*Software Platform Embedded Systems 2020 XT*" project (SPESXT, 2016). Käβmeyer *et al.* approach establishes pre-conditions and engineering tasks to achieve a product line engineering process to support an integrated change impact analysis, and systematic reuse of safety-related artefacts for different product variants. Such approach is built upon the ISO 26262 automotive safety standard. In the same way as the approach proposed in this thesis, Käβmeyer *et al.* (2015) approach has recognized the need to manage the impact of product line variation on safety-related properties, e.g., functions and contexts, malfunctions and corresponding safety integrity requirements, safety goals, component failure modes, failure rates, fault tolerance time intervals, and design choices. Käβmeyer *et al.* change impact analysis process was further evaluated with respect to variant management and reuse (KÄBMEYER *et al.* 2015a). Such approach was instantiated by integrating Enterprise Architect (EA) modeling tool, *I-SafE* failure modeling plugin built upon the EA, which is an implementation of *Open Safety Metamodel* (HOUDEK and LOWEN, 2015), defined in the context of SPES XT project, and pure::variants variant management tool. The integration between variant management and safety artefacts is achieved by means of an annotative approach where all variants are modeled together in a single model. In this model, both architectural and safety analysis model elements are tagged with variant constraints, linking these elements to product line features, allowing a product to be configured. On the other hand, in this thesis, the integration of variant management and safety-related artefacts is achieved externally to the model in the variability realization model (see Chapter 4). It contributes to reduce the complexity of the variability model as the product line size increases.

Käβmeyer *et al.* (2015) have recognized that the approach to support variability management in hazard and risk analysis, and component failure analysis proposed in this thesis (Chapters 4 and 5), earlier presented in Oliveira *et al.* (2014), provides a suitable solution, in comparison with their approach, to integrate variant management, compositional safety analysis, and model-based development to support change impact analysis and systematic reuse of safety-related artefacts. Although similarities with Oliveira *et al.* (2014) approach detailed in this thesis, the instantiation of the Käβmeyer *et al.* approach (KÄβMEYER *et al.* 2015a) is restricted to a particular domain, specific compositional safety analysis and variant management techniques.

Research in model-based safety assessment and variant management also covers approaches to support product derivation and formal verification of functional safety requirements in product variants (BESSLING and HUN, 2012), and derivation of functional safety requirements from fault tree analysis (MARTINS and OLIVEIRA, 2014). With regard to safety integrity requirements decomposition and process-based certification, ongoing research involves implementations of exhaustive (BIEBER *et al.* 2011; PAPADOPOULOS *et al.* 2010), linear (DHOUIBI *et al.* 2014; MADER *et al.* 2012), genetic (PARKER *et al.* 2013; PAPADOPOULOS *et al.* 2011), and meta-heuristic (AZEVEDO *et al.* 2013) optimization algorithms to support the automated decomposition of system safety integrity requirements throughout component level failures. These algorithms provide near-optimal solutions for the allocation of safety integrity requirements design optimization problem in safety-critical system architectures. Design optimization algorithms have the potential to address the allocation and decomposition of safety integrity requirements without incurring unnecessary development costs. These algorithms were further used in a method and tool proposed in this thesis to support near-optimal allocation of safety integrity requirements to product line components to address cost-effective process-based certification (OLIVEIRA *et al*. 2015a).

2.7 Summary

This chapter has presented a review of the state-of-art on traditional and model-based product line safety assessment and assurance case construction techniques. Although we have found studies that separately deal with product line safety assessment and assurance cases, variant management, and model-driven development few research effort has targeted to integrate these areas in a holistic approach. With regard to model-driven techniques and product line safety certification, existing design optimization techniques to support process-based certification, and model-based assurance cases techniques to support goal-based certification are dedicated to single systems. The following chapter presents an overview of the proposed systematic approach to integrate compositional safety analysis, variant management, model-based development, and model-based assurance cases into safety-critical software product line engineering processes.

Chapter 3

A MODEL-BASED APPROACH TO INTEGRATE SAFETY ASSURANCE INTO SOFTWARE PRODUCT LINE ENGINEERING

3.1 Introduction

Software Product Lines (SPL) provides the engineering basis for systematic reuse of development, assessment, and management assets in safety-critical systems. The development of a safety-critical product line differs from conventional software product lines by the emphasis on the reuse of safety assets, e.g., hazard and risk analysis, and component failure analysis, which are expensive to generate. In addition, reused safety analysis assets provide information that allows safety analysts, with the support of model-based techniques, generating fault tree analysis and FMEA safety assessment artefacts, and assurance cases. These assets are required for assurance and certification of product line instances. Therefore, safety-related assets should be managed in a prescribed way together with requirements and architectural design.

This chapter presents an overview of a systematic approach to support the integration of variant management, compositional safety analysis, assurance case construction, and model-driven development into Safety-Critical software Product Line Engineering (SC-PLE) proposed in this thesis. The approach aims to support the systematic variability management in the safety analysis model and their impact on safety assessment and assurance case construction into safety-critical software product line engineering processes. The approach is built upon product line safety assets and processes concepts defined in the "*Product Line Safety Metamodel*" proposed in earlier research on product line safety assessment (HABLI, 2009). This metamodel defines abstractions to support the systematic management and reuse of safety assets such as safety analysis and fault trees, in safety-critical software product line engineering.

In this thesis we have made a distinction between reusable safety assets, which variability should be managed, and those that should be generated from the reused safety assets. Hazards and component failure modes identified in domain engineering are reusable assets where variability should be managed, whilst fault trees and FMEA assessment assets, and assurance cases are considered to be generated for a given product variant in application engineering. These assets can be generated from the reused architecture and safety analysis assets. Such approach supports the automated traceability of context and design variation throughout architecture, safety analysis, safety assessment, and assurance cases in product line engineering processes. Thus, variant-specific safety assessment and assurance cases are integrated with other product line assets, such as feature model and architectural design, in a trustworthy manner, e.g., by addressing the justification of the reuse of product line features and their related architectural components along with their safety analysis and assessment assets (HABLI, 2009).

The proposed approach comprises processes definitions to support: the management of the impact of context and design variation in architectural and safety analysis assets by integrating variant management into compositional safety analysis (Chapter 4), the integration of compositional safety analysis and design optimization (Chapter 5), and model-based assurance cases (HAWKINS *et al.* 2015) (Chapter 6) into safety-critical software product line engineering processes. This chapter is an introduction to the following three chapters, presenting an overview of the proposed approach to support the management of safety-related variability, safety assessment and assurance case construction in safety-critical software product line engineering. Thus, the following supporting processes are presented: management and resolution of variability in product line architecture and safety models, product line compositional safety analysis and design optimization, and model-based assurance cases. For each supporting process, the phases, input artefacts and milestones[8], dependence relationships between artefacts and relationships with the "*Product Line Safety Metamodel*" (HABLI, 2009) are detailed.

[8]A milestone describes a significant event in a development project, such as a major decision, completion of a deliverable resultant of the completion of a project phase (OMG, 2015a).

Section 3.2 presents an overview of the proposed approach and relationships with product line processes. Section 3.3 presents the phases of the variability management in product line architecture and safety models. Sections 3.4 and 3.5 presents the product line compositional safety analysis and design optimization, and model-based assurance cases processes. Section 3.6 presents a summary of this chapter.

3.2 Approach Overview

This section presents an overview of the proposed approach to integrate system safety engineering into software product line engineering processes. The proposed approach differs from traditional software product line engineering processes (SEI, 2016; POHL *et al.* 2005; GOMAA, 2005) by considering safety assurance throughout product line processes as illustrated in Figure 3.1. The approach integrates variant management, compositional safety analysis, and model-based development into safety-critical software product line engineering. Thus, hazard and risk analysis, component failure analysis, assurance case construction, and variability management in safety analysis artefacts are incorporated into product line domain engineering. After product derivation, variant-specific safety analysis, and generation of fault trees analysis, FMEA results, and assurance cases are incorporated into product line application engineering. The approach defines supporting processes for managing the impact of variation in context and functional features in the safety properties, e.g., hazard and component failures, early on the product line domain engineering. In application engineering, the approach defines processes to support the automated generation of safety assessment artefacts and assurance cases. Functional features represent system functions relevant for the stakeholders, whereas usage context features represent constraints on the usage of functional features, defining how and when they can be used.

In this thesis, the combination between functional and usage context features have been used to define safety-related variants to support the systematic reuse of product line safety analysis in the application engineering. In the same way as functional variants are linked to product line architectural components in the variability model, safety-related variants should also be linked to safety properties defined in product line safety analysis. The variability model is a management artefact that contains the specification of mapping links between abstract features defined in the feature model, and their realization in the product line

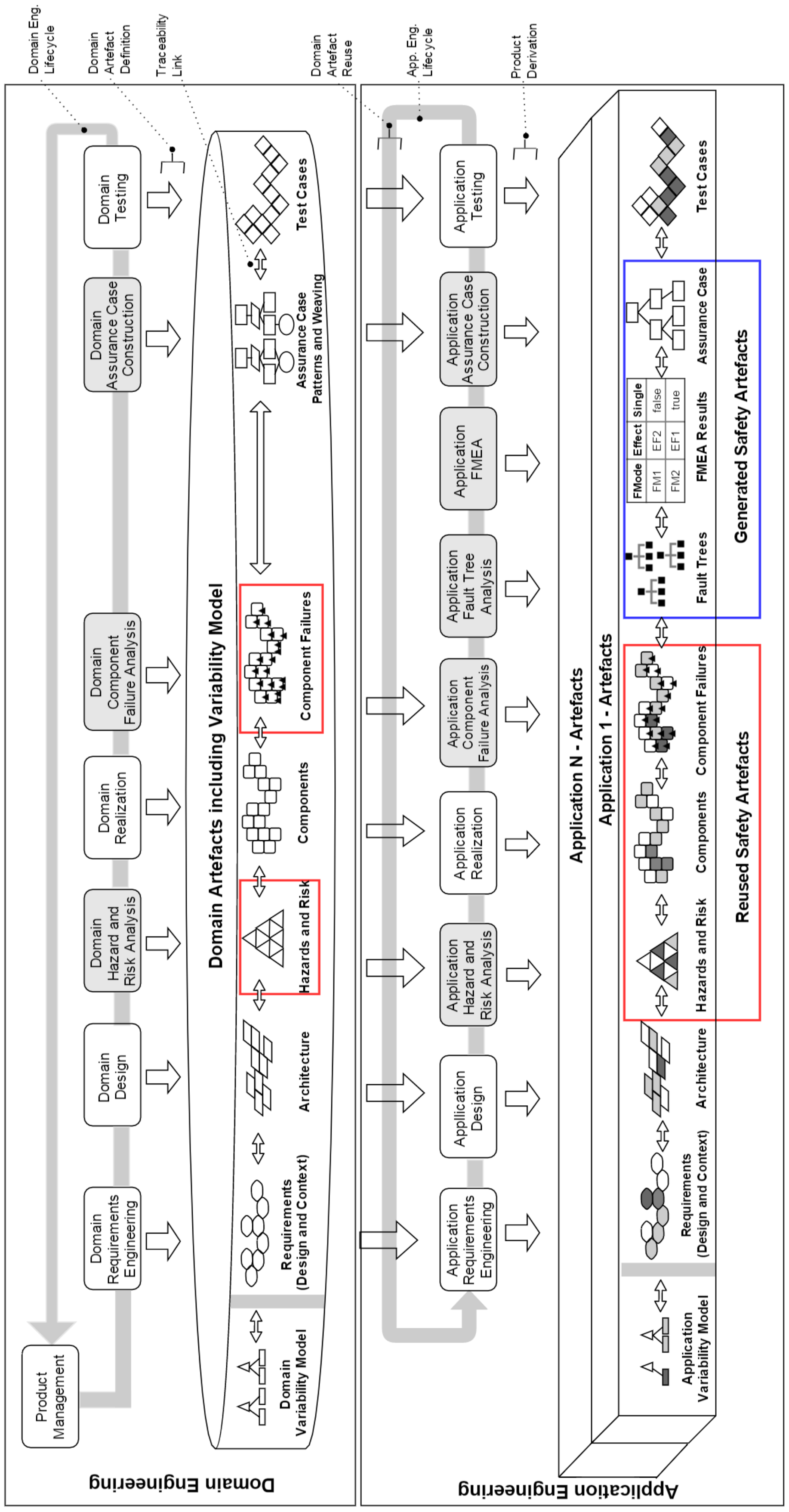

Domain Eng. Lifecycle
Domain Artefact Definition
Traceability Link
Domain Artefact Reuse
App. Eng. Lifecycle
Product Derivation
Domain Engineering
Product Management
Domain Requirements Engineering
Domain Design
Domain Hazard and Risk Analysis
Domain Realization
Domain Component Failure Analysis
Domain Assurance Case Construction
Domain Testing
Domain Artefacts including Variability Model
Domain Variability Model
Requirements (Design and Context)
Architecture
Hazards and Risk
Components
Component Failures
Assurance Case Patterns and Weaving
Test Cases
Application Engineering
Application Requirements Engineering
Application Design
Application Hazard and Risk Analysis
Application Realization
Application Component Failure Analysis
Application Fault Tree Analysis
Application FMEA
Application Assurance Case Construction
Application Testing
Application N - Artefacts
Application 1 - Artefacts
Application Variability Model
Requirements (Design and Context)
Architecture
Hazards and Risk
Components
Component Failures
Reused Safety Artefacts
Fault Trees
FMode Effect Single
FM1 EF2 false
FM2 EF1 true
FMEA Results
Assurance Case
Generated Safety Artefacts
Test Cases

Figure 3.1. Integrating system safety engineering into safety-critical software product line engineering processes (Adapted from Pohl *et al.* 2005).

assets such as requirements, architecture, components, and test cases. In safety-critical software product line engineering, in addition to architecture and components, safety analysis assets such as hazards and their causes, and component failure data, should also be considered as product line assets that contribute for the realization of product line features. Thus, in safety-critical product line variability modeling, features should also be linked to their realization in the product line safety analysis, i.e., hazard, risk, and component failure analysis. In this thesis, existing variant management techniques and tools, specifically Base Variability Resolution (BVR) (VASILEVSKIY *et al.* 2015) and Hephaestus/Simulink (STEINER *et al.* 2013), have been considered to support variability management in architecture and safety models developed using compositional safety analysis techniques, as detailed in Chapter 4. In product line application engineering, architectural variants represented by functional features and safety-related variants represented by combinations between functional and context features are then, selected, and a variant-specific architecture and its respective failure model is derived.

Variation in design and context directly impact in product line safety analysis, which different hazards with different causes, allocated safety requirements, and contributing component failure modes may arise according the selected product line instance and its context. Therefore, in order to support the systematic reuse of safety analysis artefacts together with architectural components, a supporting process for product line safety analysis aware of interactions between variation in the design and context has also been defined (Chapter 5). This process is supported by existing compositional safety analysis techniques, which in this thesis we have considered HiP-HOPS (PAPADOPOULOS *et al.* 2011) and OSATE AADL (DELANGE and FEILER, 2014). Thus, variability in hazard and risk analysis, and component failure analysis are identified and managed in domain engineering. In application engineering, the reused safety analysis data are inputs to generate variant-specific FTAs and FMEA, and safety integrity requirements decomposition (see Section 2.2.4.2) assessment artefacts, with the support of compositional safety analysis and design optimization techniques. FTA identifies causal chains of failures in components of the system architecture that can lead to system-level failures. FMEA identifies the effects of component failure modes on the overall safety of the system. Such analysis supports the identification of highly critical components where more safety measures are necessary to be applied to address system safety. Model-based safety assessment in product line application engineering is also intended to support the automatic decomposition of safety integrity requirements, in terms of

SILs[9], allocated to system-level hazards throughout the contributing component failure modes, and the automatic generation of assurance cases. The analysis of multiple SIL decomposition results, generated with the support of design optimization techniques, when performed early on domain engineering, provides feedback to the product line development processes in order to achieve process-based certification. Product line compositional safety analysis and design optimization approach is detailed Chapter 5. On the other hand, the integration of Model-Based Assurance Cases (MBAC) (HAWKINS *et al.* 2015) into safety-critical software product line engineering processes allows the automatic generation of assurance cases to support goal-based certification of product line instances.

Development and assessment artefacts represent the evidence of assurance referenced in an assurance case. MBAC is intended to generate, from a diverse of set of design and safety assessment models, a compelling and justifiable assurance argument linking safety objectives of a particular system to their supporting evidence items, e.g., hazard and risk analysis, fault trees and FMEA results. The integration of Model-Based Assurance Cases into software product line engineering demands assurance case construction activities in domain engineering and application engineering processes (Figure 3.1). In domain engineering, it is necessary to: define the structure of variant-specific assurance cases to be generated in assurance case patterns (see Section 2.4.3.1), and defining mapping links between assurance case pattern elements and system model elements that provide information for pattern instantiation in the weaving model (see Sections 2.4.3.2 and 2.4.3.3). In application engineering, it is necessary to configure the given model-based assurance case tooling with the following input artefacts: the reusable assurance case pattern specification and the weaving model produced in domain engineering; and variant-specific design and safety assessment models e.g., hazard and risk analysis and fault trees. Finally, the tool is executed to generate the assurance case model for the given product variant. The integration of Model-Based Assurance Cases into product line processes is detailed in Chapter 6.

The proposed approach to support software product line engineering for safety-critical systems makes the explicit distinction between safety artefacts in which variability should be managed and then reused, and those that should be generated from the reused safety artefacts. Product line hazard and risk analysis, and component failure analysis (safety analysis) are considered reusable safety artefacts, which variability should be managed in the variability model, to enable the systematic reuse of safety assets in application engineering. Whereas

[9]A SIL represents a range of target likelihood of failures of a safety function (IEC, 2015).

fault trees, FMEA, and assurance cases are dependent upon variant-specific architecture and safety analysis, these artefacts should be generated in application engineering, with the support of model-based techniques, from the reused architecture and safety models (Figure 3.1).

The following sections present the Safety-Critical Product Line Engineering (SC-PLE) approach showing the phases and artefacts associated with the following supporting processes: "*Variability Management in Product Line Architecture and Safety Models"*, "*Product Line Compositional Safety Analysis and Design Optimization"*, and "*Product Line Model-Based Assurance Cases"*. The Software and System Process Engineering Metamodel (SPEM) version 2.0 (OMG, 2008), supported by Eclipse Process Framework Composer[10], was used to specify activity and work product dependence diagrams associated with SC-PLE supporting processes. The relationships between the chosen model-based techniques to support each process defined in the SC-PLE approach, and the "*Product Line Safety Metamodel"* is also illustrated.

3.3 Variability Management in Product Line Architecture and Safety Models

Figure 3.2 illustrates the phases of the model-based approach to support the variability management in product line architecture and safety models in a SPEM activity diagram. In domain engineering, the input artefacts for the "*SC-PLE-1: Product Line Requirements Elicitation"* phase are dependent upon the underlying product line development strategy (KRUEGER, 2002). If a proactive strategy is chosen, then, the domain expert knowledge and other sources of domain information are inputs to this phase. On the other hand, if a reactive or an extractive strategy is adopted, similar systems available in the domain are inputs to this phase.

The requirements document is the input artefact for "*SC-PLE-1*" phase, in which variability in architecture and safety analysis is specified in the form of functional and usage context features. A functional feature represents a system function relevant for the stakeholders. A system is defined as a "*specific purpose or objective to be accomplished, whi-*

[10]https://eclipse.org/epf/

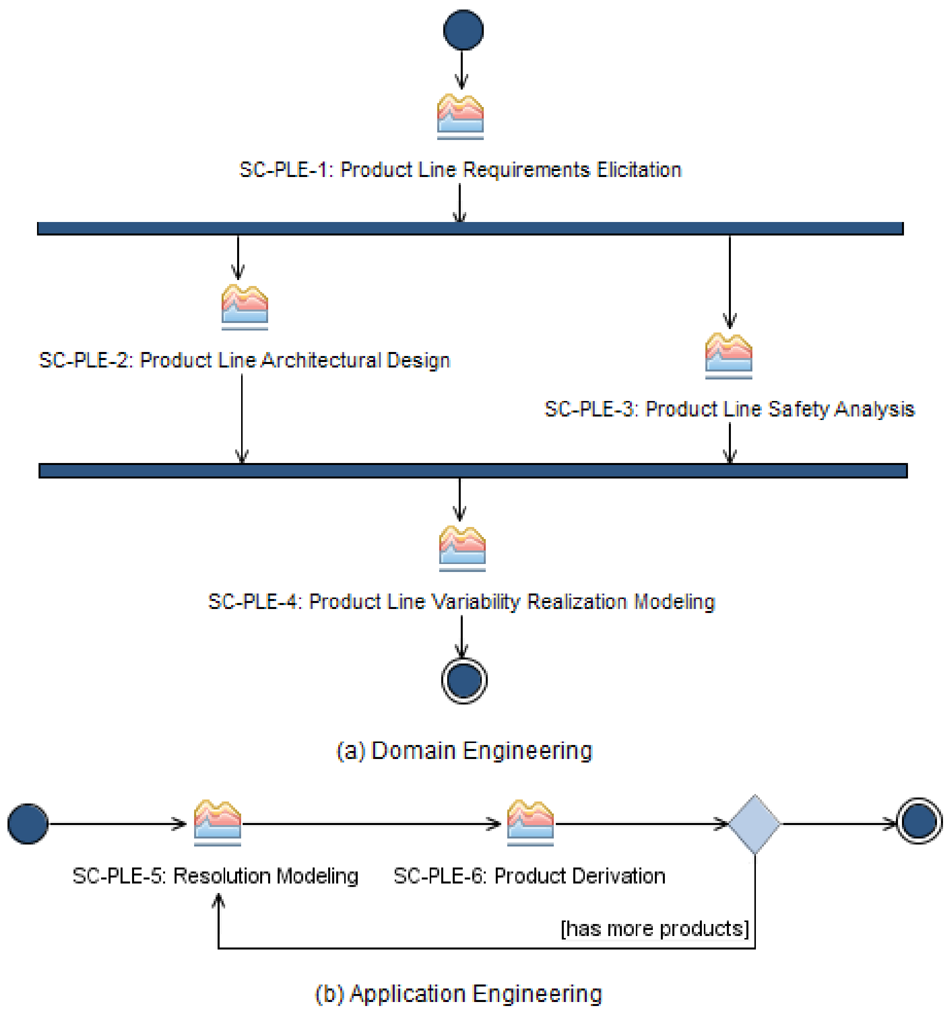

SC-PLE-1: Product Line Requirements Elicitation
SC-PLE-2: Product Line Architectural Design
SC-PLE-3: Product Line Safety Analysis
SC-PLE-4: Product Line Variability Realization Modeling
(a) Domain Engineering
SC-PLE-5: Resolution Modeling
SC-PLE-6: Product Derivation
[has more products]
(b) Application Engineering

Figure 3.2. Model-based variability management in safety-critical product lines.

ch can be specified or described without reference to the physical means of achieving it" (IEC, 2010). The wheel braking is an example of a system function of an automotive hybrid braking system (DE CASTRO *et al.* 2011). Usage context features define the characteristics of the operational environment where functional features can operate. This is important because different usage context assumed for a particular functional feature may lead to different failure conditions, hazards with different causes and allocated safety requirements. For example, safety integrity requirements, specified in terms of Development Assurance Levels (DALs), allocated to the "*Autopilot*" feature from the Tiriba UAV flight control product line (BRAGA *et al.* 2012) is level "*C*" when "*Agriculture*", "*Light UAV*" and "*Controlled Airspace*" usage context is assumed. On the other hand, when "*Autopilot*" feature is assumed to operate in a "*Defense*" application, with a "*Small UAV*" in a "*Controlled Airspace*", DAL level "*A*" is assigned. Feature modeling tools can be used to support this phase. At the end of this phase, product line feature and context models are delivered.

After specifying product line requirements in functional and usage context features, the product line architectural design is defined on the targeted modeling language (*SC-PLE-2: Product Line Architectural Design* phase in Figure 3.2). Existing system modeling languages can be used to support this phase. In this thesis we have considered MATLAB/Simulink (MATLAB, 2015) and Analysis and Architectural Description Language (AADL) (SAE, 2012) modeling language supported by OSATE 2 AADL[11] environment. At this phase, firstly, architectural patterns are defined, followed by the specification of architectural components for the realization of core functional features. In the following, components representing variable functional features are specified, and variability mechanisms are defined. Variability mechanisms refer to model elements used to represent variability in the product line architecture model. Examples of variability mechanisms are "*Switch*" blocks, "*Enabler Subsystems*", and "*Variant*" blocks in MATLAB/Simulink (BOTTERWECK *et al.* 2010). Variability mechanisms can also be defined outside the product line architecture model with the support of existing variant management techniques, e.g., BVR (VASILEVSKIY *et al.* 2015), Common Variability Language (HAUGEN *et al.* 2008), Pure::variants (PURE::SYSTEMS, 2016), and Hephaestus/Simulink (STEINER *et al.* 2013). At the end of this phase, the product line architecture model is delivered.

The "*SC-PLE-3: Product Line Safety Analysis*" phase intends to identify system-level hazards, their causes, allocating safety requirements according to the targeted safety standard, and identifying contributing component failure modes associated with design and context variations. This phase can be performed from the initial stages of the product line architectural design when a preliminary architecture is defined. In this thesis, existing compositional safety analysis techniques such as HiP-HOPS and OSATE 2 AADL/Error Annex were considered to support this phase. At the end of this phase, the hazard and risk analysis, and component failure analysis results in a range of usage scenarios, stored into the product line failure model, is delivered. From the product line context and feature models, architecture and failure models, mapping links between product line features and their realization in architectural components and safety analysis data are defined in the variability model during the "*SC-PLE-4: Variability Realization Modeling*" phase. In this thesis, variant management tools integrated to model-based development and compositional safety analysis tools can be used to provide automated support for this phase. At the end of this phase, the variability realization model is delivered. The variability realization model and the integration between variant

[11] http://www.aadl.info/aadl/osate/stable/

management, compositional safety analysis, and model-based development support the systematic reuse of the product line architecture and safety analysis models. In product line application engineering, “*Resolution Modeling*” and “*Product Derivation*” phases can be performed iteratively. In the *“SC-PLE-5: Resolution Modeling”* phase, architectural and safety-related variants defined in the product line feature model are selected, and an instance model is obtained. In *“SC-PLE-6: Product Derivation”* phase, with the support of variant management tools, variant-specific architecture and safety models are derived according to the instance model.

3.3.1 Dependencies between Product Line Development and Management Artefacts

Variability management in product line architecture and safety models phases generate a set of development and management work products and deliverables. These artefacts have dependence relationships that should be captured and traced throughout product line domain engineering and application engineering. Figure 3.3 shows the dependence relationships between product line development and management assets. Product line feature and context models are primary work products resultant from “*Product Line Requirements Elicitation*”

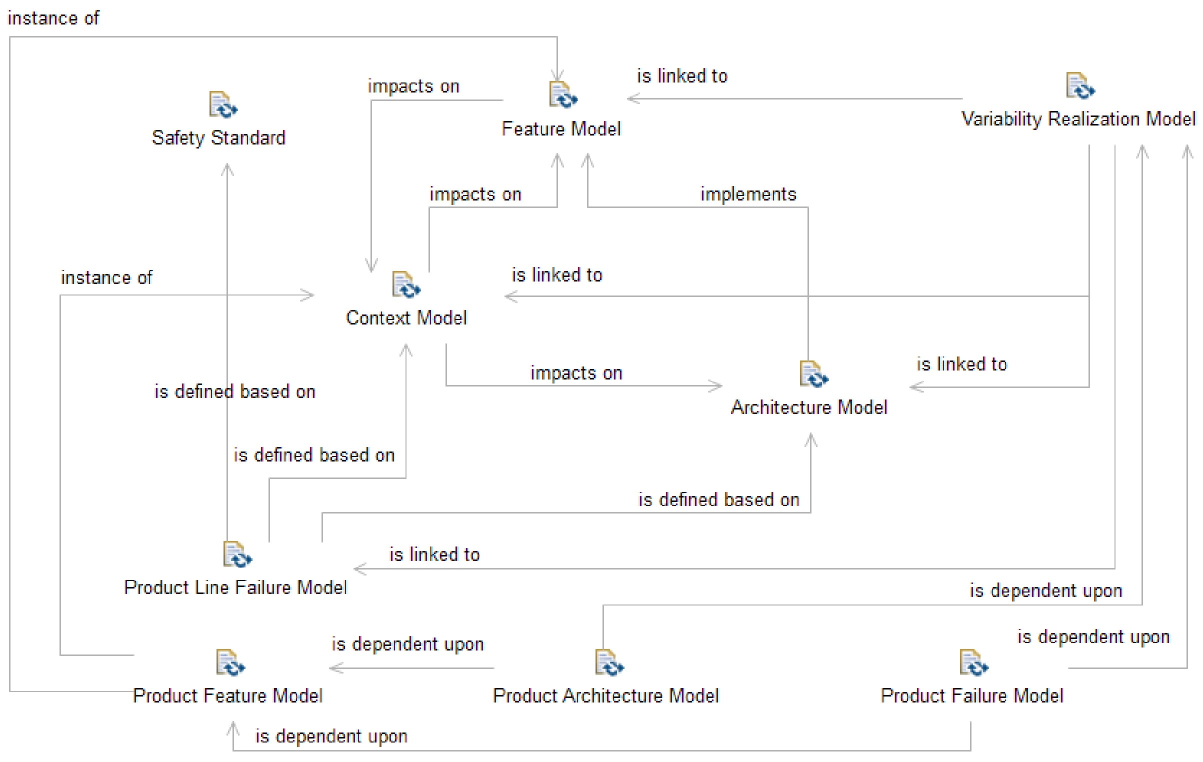


Figure 3.3. Development and management work product dependence diagram.

phase. The feature model defines functional variants, and the context model defines the impact of characteristics of the operational environment and functional variants in the system safety properties. The product line feature model impacts in the definition of the context model. On the other hand, the product line context model imposes constraints on the usage of functional features. For example, the context model might delimit the interactions of a given functional feature with other features, limiting the way in which functional features can be combined to derive product variants in the product derivation phase. In addition, the product line context model might impose safety requirements in the form of functional safety features, e.g., redundancy, or safety integrity requirements. The product line architecture model represents the realization of functional variants specified in the feature model. The product line context model might also impact in the definition of the product line architecture model, leading to the addition of architectural components that contribute for the realization of functional safety requirements associated with usage context features. Examples of usage context features are operating environment, which may require the inclusion of different sensors and actuators (BRAGA *et al.* 2012).

The product line failure model is resultant from the safety analysis activities such as hazard and risk analysis, and component failure analysis. The product line failure model is defined based on the targeted safety standard, and from the analysis of functional and usage context features, and the product line architecture model. Changes in the context or in the product line architecture may lead to different hazards with different causes and effects on the overall system safety. Changes on usage context features may also impact in the allocation of different safety integrity requirements. The management of variation in design and context, expressed in the feature and context models, and their relationships with architectural and failure model elements is achieved in the variability realization model. This deliverable is resultant from the *"SC-PLE-4: Product Line Variability Realization Modeling*" phase. The variability realization model defines mapping links between functional variants, specified in the feature model, and product line architectural model elements and their connections, which represent the realization of functional variants at the design level. The variability realization model also defines mapping links between safety-related variants defined in the product line context model, and product line failure model elements, e.g., hazards, their causes, allocated safety requirements, and component failure data.

In the product line application engineering, the product feature model, i.e., the instance model, is a deliverable resultant from the *"SC-PLE-5: Resolution Modeling"* phase. Product feature model represents the selection of functional and safety-related variants specified in the product line feature and context models. Thus, the product feature model is an instance of these models defined in the product line domain engineering. Product architecture and failure models are deliverables generated as a result of the *"SC-PLE-6: Product Derivation"* phase. The definition of these deliverables is dependent upon the product feature model and the product line variability realization model.

3.3.2 Product Line Development and Management Models and Product Line Safety Metamodel

The "*Product Line Safety Metamodel"* (HABLI, 2009), shown in Figure 3.4, prescribes that "*Product Line Management"* assets define traceability links between variation specified in product line *Contextual* and *Feature* assets throughout *Design Components* and *Product Line Assessment* assets. Examples of assessment assets are hazard and risk analysis, and component failure data. In this metamodel, the "*Functional Failure Model"* defines relationships between feature and contextual assets, hazard and risk analysis data such as hazards, their causes, effects, risk classification, and allocated safety requirements. The "*Architectural Failure Model"* defines the relationships between system-level hazards and contributing component failure modes in a causal chain, in the same way as Fault Tree Analysis technique (NASA, 2002). In the product line domain engineering, the "*Architectural*

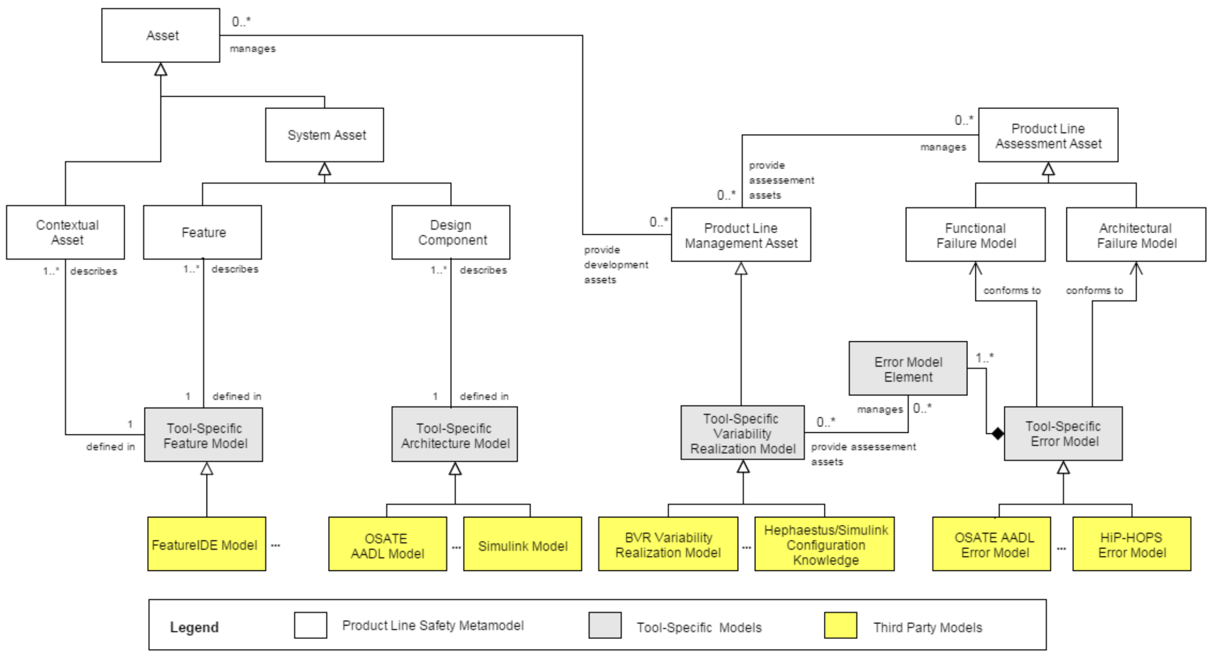

Asset
0..*
manages
System Asset
Contextual Asset
Feature
Design Component
1..* describes
1..* describes
1..* describes
1 defined in
1 defined in
1 defined in
Tool-Specific Feature Model
Tool-Specific Architecture Model
FeatureIDE Model
OSATE AADL Model
Simulink Model
Product Line Assessment Asset
0..* manages
provide assessement assets
0..*
0..*
Product Line Management Asset
provide development assets
Functional Failure Model
Architectural Failure Model
conforms to
conforms to
Error Model Element
1..*
manages 0..*
Tool-Specific Variability Realization Model
0..*
provide assessement assets
Tool-Specific Error Model
BVR Variability Realization Model
Hephaestus/Simulink Configuration Knowledge
OSATE AADL Error Model
HiP-HOPS Error Model
Legend
Product Line Safety Metamodel
Tool-Specific Models
Third Party Models

Figure 3.4. Variability management in the safety model and product line safety metamodel.

Failure Model" represents assumptions about output and input deviations of components, and internal failures that can lead to system level failures in a range of usage scenarios. In the product line application engineering, the "*Architectural Failure Model"* represents leaf nodes and gates of a fault tree for a given system-level hazard.

The proposed model-based approach to support variability management in architecture and safety models provides guidance to integrate existing variant management, compositional safety analysis, and model-based development techniques that comply with the "*Product Line Safety Metamodel*". In the "*SC-PLE-1: Product Line Requirements Elicitation"* phase, tool-specific feature models, e.g., FeatureIDE, Pure::variants, BVR, and Hephaestus/Simulink feature models, can be used to specify contextual and functional features. In the "*SC-PLE-2: Product Line Architectural Design"* phase, design components can be specified in AADL or MATLAB/Simulink models. The "*SC-PLE-3: Product Line Safety Analysis"* phase can be performed with the support of existing compositional safety analysis techniques, e.g., OSATE ADDL Error Annex (DELANGE and FEILER, 2014) and HiP-HOPS (*PAPADOPOULOS et al.* 2011). These techniques provide failure model specifications that comply with "*Functional and Architectural Failure Models".* Finally, variant management techniques support the specification of mapping links between design and context variation, defined in the feature model, and their realization in architecture and safety models, providing management assets to support the systematic reuse of architecture and safety analysis. For example, the BVR variability realization model (HAUGEN and OGARD, 2014) and Hephaestus/Simulink configuration knowledge (STEINER *et al.* 2013) link product line features to references to model objects representing architectural design components, failure model elements, and other product line assets. The variability realization model defines compositional rules to supports the resolution of architectural and safety-related variants in the "*SC-PLE-6: Product Derivation"* phase.

3.4 Product Line Compositional Safety Analysis and Design Optimization

The *"SC-PLE-3: Product Line Safety Analysis"* is the first phase of the product line compositional safety analysis and design optimization process illustrated in Figure 3.5. Product line safety analysis comprises "*Functional Hazard Assessment"* and "*Component Failure Analysis"* activities to be performed in product line domain engineering. Functional

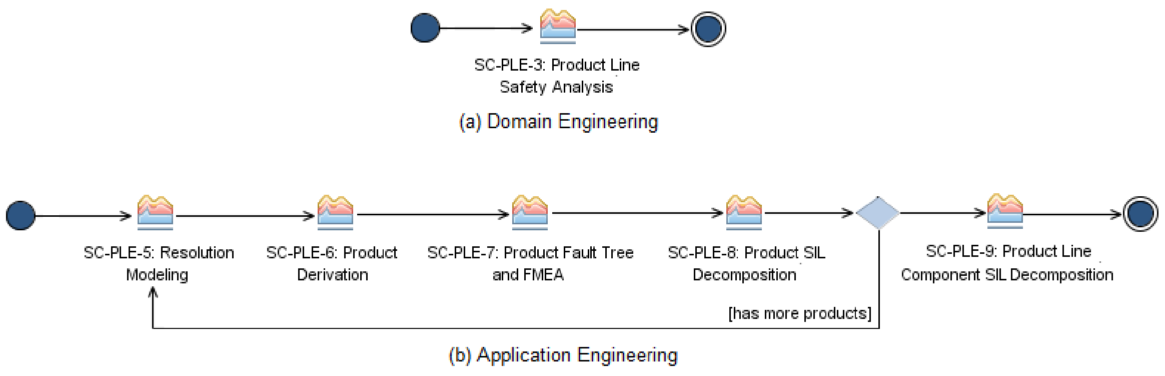

SC-PLE-3: Product Line Safety Analysis
(a) Domain Engineering
SC-PLE-5: Resolution Modeling
SC-PLE-6: Product Derivation
SC-PLE-7: Product Fault Tree and FMEA
SC-PLE-8: Product SIL Decomposition
[has more products]
SC-PLE-9: Product Line Component SIL Decomposition
(b) Application Engineering

Figure 3.5. Compositional safety analysis in software product line engineering.

hazard assessment is performed from the analysis of the relationships between functional and context features, and the product line architecture model. From such analysis, a range of usage scenarios that impact in the system safety properties are derived, by combining architectural and contextual variants. Functional hazard assessment is performed by considering the possible threats to system safety, their causes, and allocated safety requirements associated with each usage scenario. Further on the analysis, during the *"Component Failure Analysis",* assumptions about the contributing output and input deviations, and internal failures of components are made by considering the usage scenarios earlier defined in functional hazard assessment. At the end of this phase, the product line failure model is delivered.

In the product line application engineering, after the completion of the "*SC-PLE-6: Product Derivation*" phase, fault trees and FMEA results are generated with the support of compositional safety analysis techniques in the "*SC-PLE-7: Product Fault Tree and FMEA*" phase. Variant-specific architectural and failure models are input artefacts to generate fault trees and FMEA results for a given product variant. For each variant-specific hazard, a fault tree provides the failure mode causal chains showing how combinations of failures in architectural components contribute to the occurrence of a system-level hazard. FMEA shows the component failure modes, their effects, and the required mitigation mechanisms to eliminate or minimize the effects of a given component failure mode. Variant-specific FMEA is useful to identify highly critical architectural components, supporting architectural decisions to increase the fault tolerance of the product line and variant-specific architectures. In this thesis, compositional safety analysis techniques, e.g., OSATE AADL Error Annex and HiP-HOPS, have been adopted to support the automatic synthesis of fault trees and FMEA results from the product-specific architecture and failure models. Fault trees and FMEA are input artefacts for the "*SC-PLE-8: Product SIL Decomposition*" phase.

3.4.1 SC-PLE-8: Product SIL Decomposition Phase

Product safety integrity requirements decomposition is intended to iteratively decompose safety requirements allocated to variant-specific hazards into safety requirements to be allocated to the contributing component failure modes. Safety integrity requirements decomposition allows a safety-critical system architecture meeting a particular target SIL allocated to a given system-level failure, i.e., a hazard, without all contributing component failures having to meet that target. If a system-level hazard is caused only when two independent components fail together, these components can share the responsibility of meeting the SIL allocated to that hazard, rather than each one having to meet the original SIL. Safety standards in aerospace and automotive domains define "*algebras*" for safety integrity requirements decomposition based on rules about combining failure probabilities defined in these standards.

In SIL *algebra*, each safety integrity level is equivalent to an integer value. For example, in ISO 26262 (EUROCAE, 2010), the Automotive Safety Integrity Level (ASIL) algebra is: QM (Quality Management – no effect) = 0, A = 1, B = 2, C= 3, and D = 4 (most stringent). ASIL algebra defines that if *n* components must fail simultaneously to cause a given hazard[12], the total ASIL assigned to these *n* components must add up to the ASIL allocated to the system hazard they originate, as illustrated in Figure 3.6. Thus, two redundant components assuring an ASIL D system function might individually only be required to meet ASIL B because together they produce the total required ASIL value (2 + 2 = 4). Higher ASILs mean higher costs, because meeting stringent safety requirements requires more safety measures, more effort, and higher-quality components. Therefore, component SILs could significantly impact in both development and production costs. SIL decomposition allows safety analysts to efficiently allocating safety integrity requirements so that safety requirements can be met without being unnecessarily stringent or expensive.

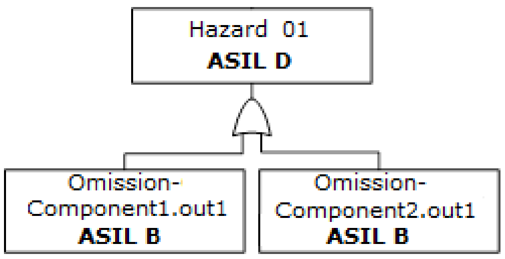


Figure 3.6. ASIL decomposition example.

[12]*Potential source of harm caused by malfunctioning behavior of the item"* in ISO 26262: Road Vehicles - Functional Safety.

Existing design optimization techniques support the automatic decomposition of safety integrity requirements throughout component failure modes from the potential system fault propagation specified in fault trees (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; PARKER *et al.* 2013; BIEBER *et al.* 2011). These techniques comply with ISO 26262 automotive safety standard, and DO-178C (RTCA, 2012) and ARP 4754A (EUROCAE, 2010) aerospace standards. Design optimization SIL decomposition techniques do not guarantee finding optimal solutions, but they are capable of providing near optimal SIL decomposition solutions. In this thesis, design optimization techniques were considered to support the *"SC-PLE-8: Product SIL Decomposition"* phase. The output of this phase is the decomposition of safety integrity requirements allocated to variant-specific hazards throughout contributing component failure modes. At the end of this phase, we can derive another product variant, generating fault trees, FMEA, and safety integrity requirements decomposition results for that variant, or, if has not more variants to work with, performing the analysis of multiple variant-specific SIL decomposition results to identify the required safety integrity requirements to support product line process-based certification.

The analysis of SIL decomposition results from multiple product line instances early on the product line design provides feedback to the product line domain engineering process, guiding product line engineers in developing components that address safety across a range of product variants without being unnecessarily stringent or expensive. Such analysis is performed during the "*SC-PLE-9: Product Line Component SIL Decomposition"* phase with the support of the product line component safety integrity requirements decomposition method and tool developed in the course of this thesis. The tool performs the analysis of multiple SIL decomposition results to find the required SILs for each product line component addressing safety across the range of product variants considered in the analysis. The tool was developed as an extension of the HiP-HOPS compositional safety analysis tool, and it supports the analysis of ASIL and DAL allocation results. Product line compositional safety analysis and product line component SIL decomposition method and tool are detailed in Chapter 5.

3.4.2 Dependencies between Product Line Assessment Artefacts

During the product line compositional safety analysis and design optimization phases, a set of assessment work products and deliverables are generated in both domain engineering and application engineering processes. Figure 3.7 shows the dependence relationships between these assets in a work product dependence diagram. The product line failure model is produced during the safety analysis phase in domain engineering. During the functional hazard assessment, hazard and risk analysis data are added to the product line failure model. This asset is defined by considering the targeted safety standard, and from the analysis of the product line context and architecture models. Such analysis allows safety engineers scoping the product line safety analysis to a range of product variants and usage scenarios. After the identification of potential hazards that can arise in a range of product variants, it is identified how input/output deviations, and internal failures of components, i.e., component failure logic, can contribute to the occurrence of hazards in a range of scenarios. The component failure logic is defined upon the product line hazard analysis, context and architecture models.

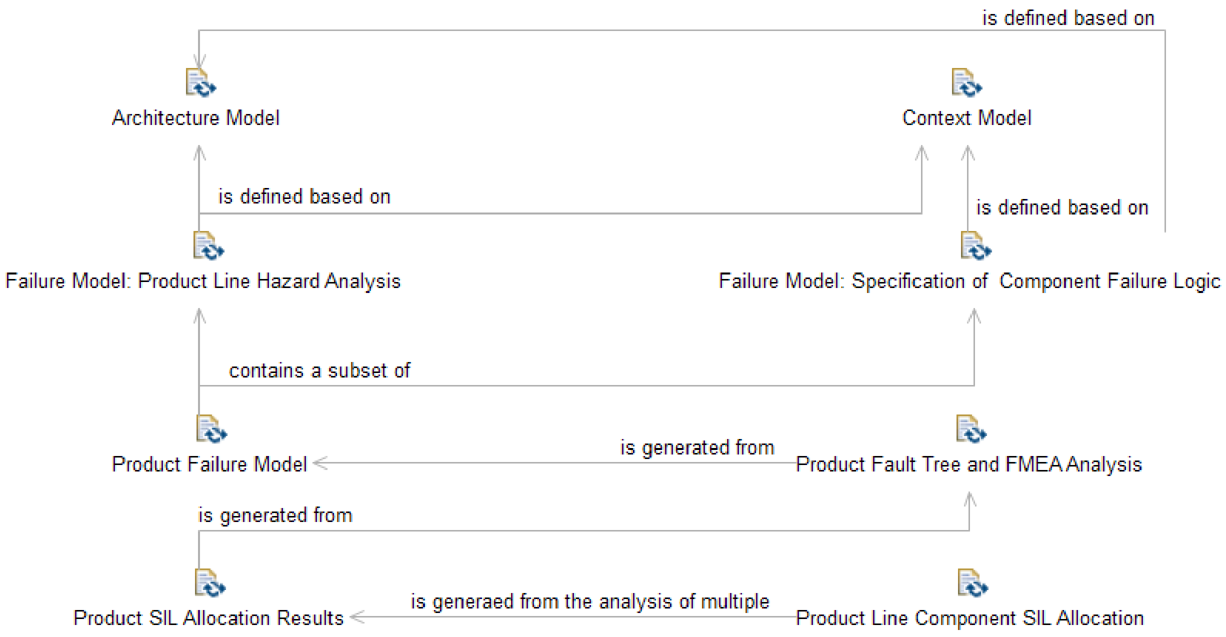

is defined based on
Architecture Model
Context Model
is defined based on
is defined based on
Failure Model: Product Line Hazard Analysis
Failure Model: Specification of Component Failure Logic
contains a subset of
Product Failure Model
is generated from
Product Fault Tree and FMEA Analysis
is generated from
Product SIL Allocation Results
is generaed from the analysis of multiple
Product Line Component SIL Allocation

Figure 3.7. Model-based safety assessment work product dependence diagram.

The product failure model generated in application engineering is a subset of product line hazard and risk analysis, and component failure analysis, which represents safety-related information of a particular product variant. Product fault trees and FMEA results are generated from the product failure model. The SIL allocation result for a given product variant is generated from product fault tree analysis. Finally, the product line component SIL allocation work product is generated from the analysis of multiple variant-specific SIL allocation results. Assessment artefacts produced in the product line application engineering provide safety assessment of a variant-specific architecture model. Variant-specific development and assessment models are input artefacts to generate assurance cases for a given product variant with the support of model-based assurance cases techniques (HAWKINS *et al.* 2015).

3.4.3 Product Line Assessment Models and Product Line Safety Metamodel

As described in Section 3.2.2, tool-specific failure models conforms to *"Functional* and *Architectural Failure"* metamodels defined in the "*Product Line Safety Metamodel"* (HABLI, 2009), as shown in Figure 3.8. Tool-specific failure models as provided by OSATE AADL and HiP-HOPS compositional safety analysis techniques contain abstractions that comply with *"Functional Failure Metamodel"* (HABLI and KELLY, 2009). Figure 3.9 illustrates how HiP-HOPS Error metamodel conforms to the "*Functional Failure Metamodel"*. In the "*Functional Failure Metamodel"*, associations between *"Failure Condition"* and "*Failure Condition Effect"* elements that represent a system-level hazard and its causes, is defined via "*Causal Relationship"* element. The equivalent relationship is expressed by associations between *"Cause"* and *"Hazard"* elements defined in the HiP-HOPS Error metamodel (HiP-HOPS TEAM, 2013). Therefore, *"Cause"* and *"Hazard"* model elements are respectively equivalent to *"Failure Condition"* and *"Failure Condition Effect"* elements defined in the *Functional Failure Metamodel*. A *"Failure Condition"* is associated with a "*Classification"* via "*Classification Relationship"*. In a safety-critical product line, the risk "*Classification"* of a failure condition is dependent upon functional and contextual assets. Thus, a *"Failure Condition Effect"* might have different "*Classifications"*, i.e. *Safety Integrity Levels*, according to the selection of functional and usage context features.

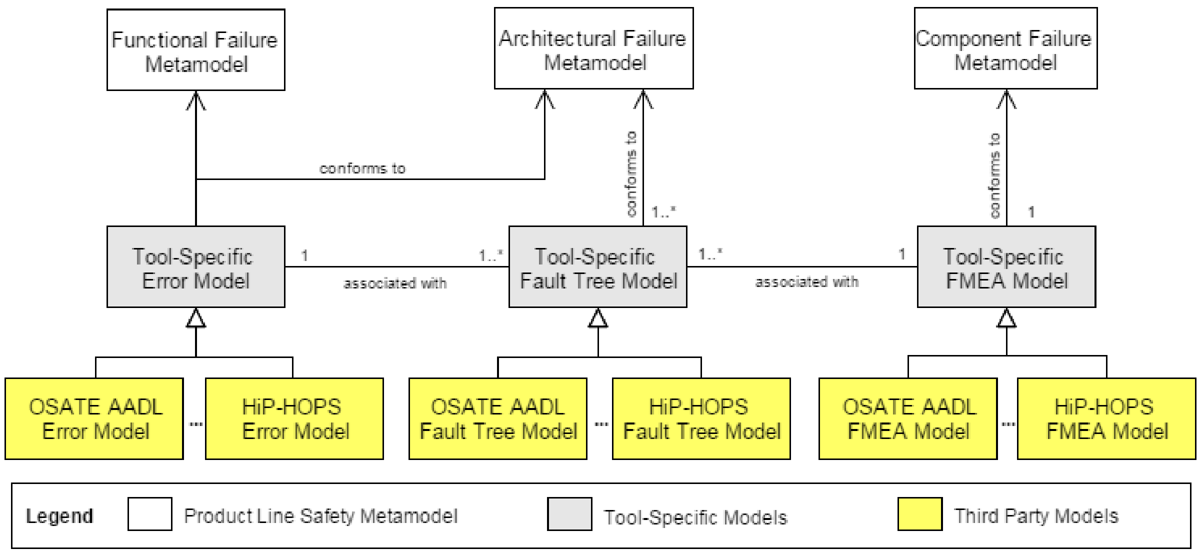


Figure 3.8. Model-based safety assessment artefacts and failure metamodels.

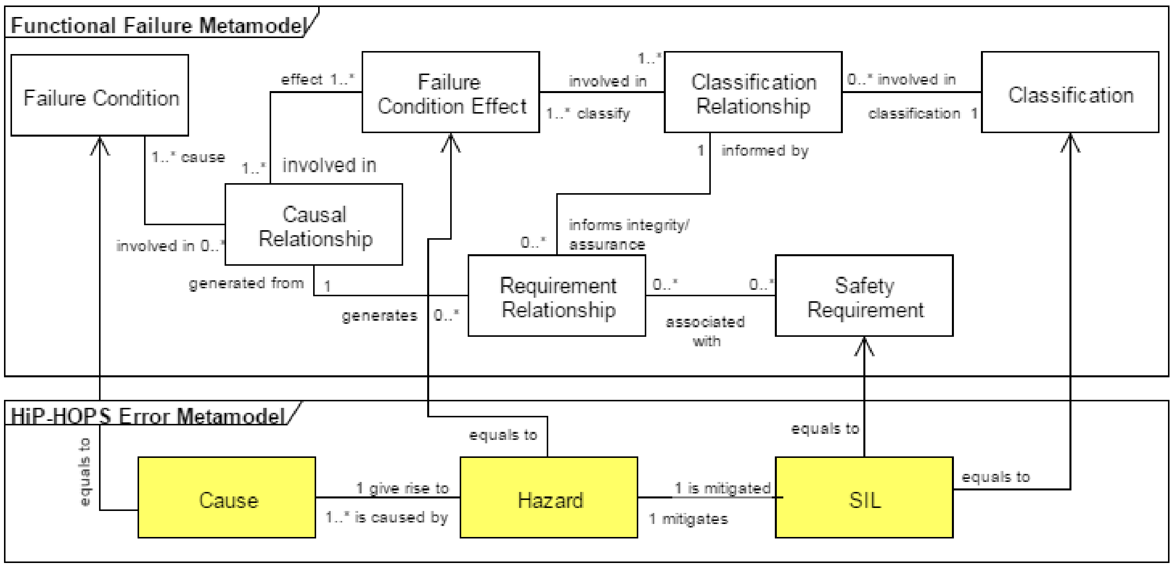


Figure 3.9. Functional failure and HiP-HOPS error metamodels.

Equivalent relationships in the HiP-HOPS Error metamodel are expressed by associations between *"Hazard"* and *"SIL"* elements.

In the "*Functional Failure Metamodel"*, "*Classification Relationship"* informs the integrity of a *"Failure Condition Effect"* via "*Requirements Relationship",* which is associated with the required "*Safety Requirements"* to mitigate a *"Failure Condition Effect"*. "*Safety Requirement"* can be architectural decisions, or a set of development and assessment activities prescribed by process-oriented safety standards according to the integrity of a given *"Failure Condition Effect"*. A "*SIL"* element in the HiP-HOPS Error metamodel is equivalent to *"Classification"* and "*Safety Requirement"* abstractions defined in the "*Functional Failure Metamodel"*. Fault Tree and FMEA models provided by HiP-HOPS and OSATE AADL compositional safety analysis techniques conform to "*Architectural Failure* and *Component Failure"* metamodels defined in the *"Product Line Safety Metamodel"*. The "*Architectural Failure Metamodel"* defines the relationships between variation in design and context and their impact in fault tree leaf nodes and gates. The "*Component Failure Metamodel"* specifies relationships between product line variation and component *"Failure Modes"* and their "*Effects"* on the system safety, i.e., it shows how variation in architectural components impacts in the component failure modes and their effects.

3.5 Product Line Model-Based Assurance Cases

Assurance Case construction requires a safety argument linking evidence items to claims of conformance to dependability requirements (SEI, 2016a). Model-based assurance case brings the benefits of automation, transformation, and validation from model-based engineering to support the assurance case construction process (HAWKINS *et al.* 2015). Some of the existing model-based assurance case techniques are based on the *Model Weaving* approach (DEL FABRO *et al.* 2005). *Model weaving* allows the interoperability between design, process, and assessment models and metamodels and the assurance case. Thus, changes in design, process, and assessment models are further propagated throughout the assurance case. Integrating model-based assurance cases into software product line processes demands assurance case construction activities to be performed in domain engineering to support the automatic generation of variant-specific assurance cases in application engineering. Figure 3.10 shows the model-based assurance case phases in product line domain engineering and application engineering.

In domain engineering, assurance case patterns are defined in the "*SC-PLE-10: Assurance Case Pattern Modeling"* phase. Assurance case patterns represent the structure of an assurance case with abstract elements named terms, which represent information elements provided by different system models required to instantiate assurance case patterns. In the "*SC-PLE-11: Asset Metamodeling"* phase, the structure of each system model to be considered in model weaving transformations (HAWKINS *et al.* 2015) is defined in a metamodel. Assurance case patterns and system metamodels are input artefacts for the "*SC-PLE-12: Weaving Modeling"* phase. In this phase, assurance case pattern abstract terms are linked to model elements defined in the system metamodels in the weaving model. The mapping links specified in the weaving model defines model transformation rules to instantiate assurance case patterns, for a given product line instance, from the information provided by variant-specific design, process, and safety assessment models. Assurance case patterns, asset metamodels, and the weaving model are reusable assurance case construction assets used to support model-based assurance cases in product line application engineering.

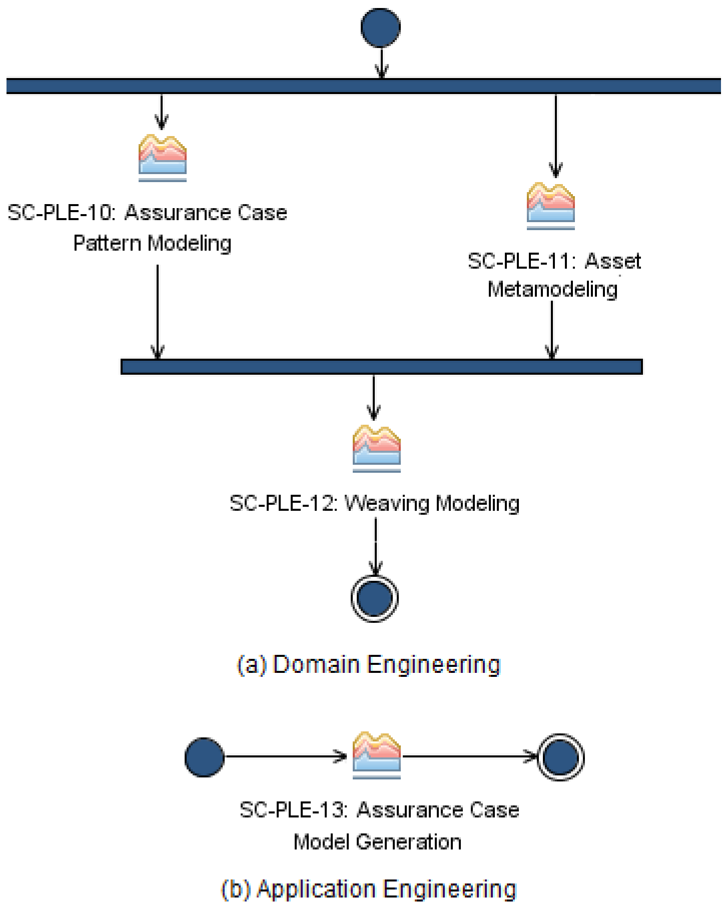

SC-PLE-10: Assurance Case Pattern Modeling
SC-PLE-11: Asset Metamodeling
SC-PLE-12: Weaving Modeling
(a) Domain Engineering
SC-PLE-13: Assurance Case Model Generation
(b) Application Engineering

Figure 3.10. Model-based assurance cases in software product line engineering.

In application engineering, reusable assurance case patterns, asset metamodels, and the weaving model, together with variant-specific design, process, and assessment models are inputs artefacts for the *"SC-PLE-13: Assurance Case Model Generation"* phase. In this phase, an assurance case model for a given product variant is generated by configuring a model-based assurance case tool (HAWKINS *et al.* 2015) with the input artefacts mentioned earlier. The generated assurance case provides the assurance of a given variant-specific architecture by linking its associated design, process, and assessment models to claims of conformance with dependability requirements. The product line model-based assurance cases approach is detailed in Chapter 6.

3.5.1 Dependencies between Product Line Model-Based Assurance Case Artefacts

Relationships between product line model-based assurance case artefacts produced in domain engineering and application engineering are captured in the work product dependence diagram shown in Figure 3.11. In domain engineering, assurance case patterns are defined in conformance with the underlying assurance modeling notation that can be Goal Structuring Notation (KELLY, 2003), Claim-Argument-Evidence (BISHOP and BLOOMFIELD, 2010), or other notations in compliance with the OMG Structured Assurance Case Pattern Metamodel (OMG, 2015a). Asset metamodels are defined according to the structure of design and safety assessment (e.g., failure, fault trees and FMEA) output models provided by model-based development and compositional safety analysis techniques. Mapping links between assurance case pattern elements and elements from a diverse set of asset metamodels are defined in the weaving model. The weaving model must conform to a weaving metamodel. In application engineering, variant-specific asset models, which are input artefacts to model-based assurance cases, must conform to their respective metamodels. Finally, the product assurance case model is generated from the information provided by variant-specific system models, e.g., design, process and assessment artefacts. A product assurance case model is an instance of assurance case patterns defined in domain engineering.

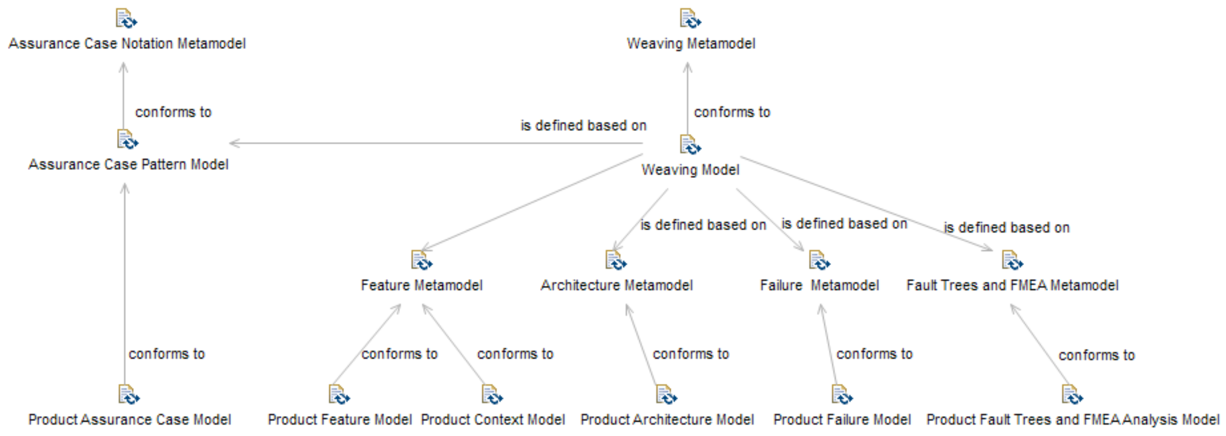


Figure 3.11. Model-based assurance work product dependence diagram.

3.5.2 Model Based Assurance Cases and Product Line Safety Metamodel

Product line model-based assurance cases supports the automated traceability of design and context variation throughout design, safety assessment, and the assurance case as defined in the "*Product Line Safety Metamodel*" and illustrated in Figure 3.12. Therefore, changes in design and assessment models are further propagated throughout the assurance case. It means that references to variant-specific model elements embedded into *"Argument"* and "*Evidence*" assurance case elements are dynamically linked to "*System Assets*" e.g., feature and reference architecture models, *Contextual Assets*, and assessment assets, e.g., failure model, fault trees and FMEA models. Assessment assets are directly linked to

"*System*" and "*Contextual*" assets associated with a given product line instance. Changes in product-specific "*System*" and "*Contextual*" assets are automatically propagated throughout failure model, fault trees and FMEA results produced in application engineering. Such traceability is specified in the weaving model via mapping links between assurance case pattern elements and references to design, process, and assessment model elements that contain the information required to instantiate assurance case patterns for a given product line instance.

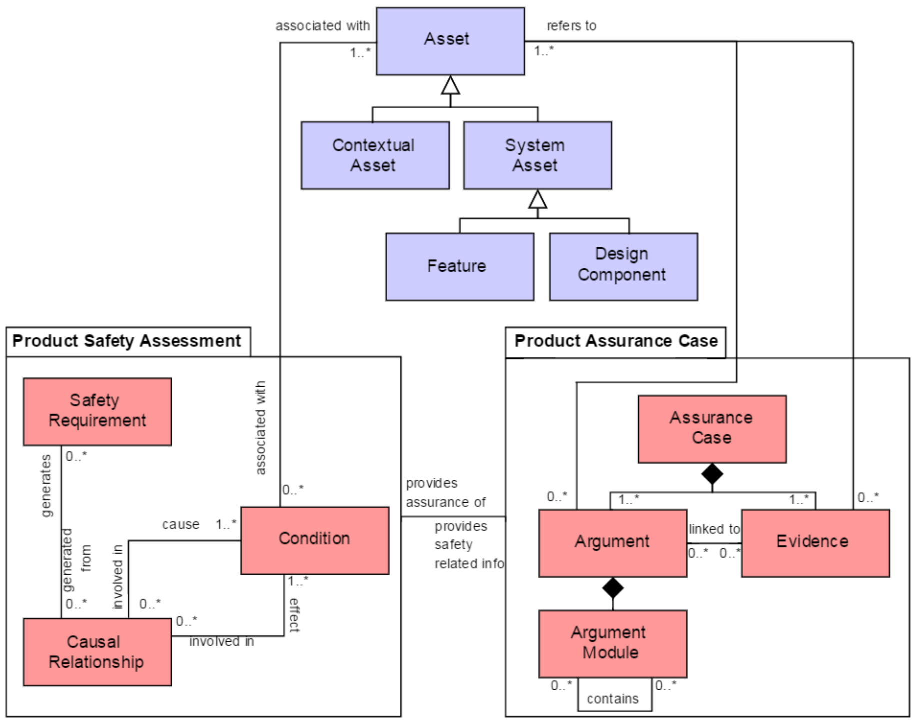


Figure 3.12. Product safety assessment and assurance case metamodels (Adapted from Habli 2009).

3.6 Summary

This chapter presented an overview of the proposed systematic approach to integrate variant management, compositional safety analysis, and model-based assurance cases into safety-critical software product line engineering processes. The approach provides supporting processes for managing and resolving variability in architecture and safety models, and integrating compositional safety analysis and model-based assurance cases into software product line engineering processes. The following three chapters detail the variability management in product line architecture and safety analysis models (Chapter 4), product line compositional safety analysis and design optimization (Chapter 5), and product line model-based assurance cases (Chapter 6) supporting processes.

Chapter 4

VARIABILITY MANAGEMENT IN PRODUCT LINE ARCHITECTURE AND SAFETY MODELS

4.1 Introduction

Safety-critical software product line engineering requires the integration of system safety engineering into product line processes. Compositional safety analysis provides the automated support for system safety engineering and seamless integration between the system design and safety analysis. Thus, architectural design and safety analysis can be performed in the same model. It contributes to reduce the complexity of the product line safety analysis. Whereas safety-critical software product line engineering involves system safety engineering, safety analysis and variability management in the safety model should be considered throughout product line processes. Safety analysis should be performed aware of the impact of context and design variation in the safety properties to enable the systematic reuse of product line architecture and safety analysis assets. For example, context and design variation may change hazards, their causes, and the risk they pose for the overall system safety. As safety properties may change according to the selection of product variants, variability in the safety model should also be managed in safety-critical product line engineering. Thus, mapping links between context and design variation and their realization in the safety model should be defined in the variability model.

Existing variant management techniques such as GEARS (BIG LEVER, 2016), BVR (VASILEVSKIY *et al.* 2015), Hephaestus/Simulink (STEINER *et al.*, 2013), pure::variants (PURE::SYSTEMS, 2016), and Common Variability Language (CVL) (HAUGEN *et al.* 2008), provide support for managing variation on requirements, architecture, components, source code, and test cases. However, these techniques were not originally designed to support variability management in safety models developed with the support of compositional safety analysis techniques, e.g., OSATE 2.0 AADL/Error Annex[13], and HiP-HOPS integrated to MATLAB/Simulink[14], and SimulationX[15]. Variability management in the safety model enables the traceability of context and design variation throughout safety assessment artefacts and systematic reuse of architectural components and safety analysis assets. The reuse of safety analysis information in safety-critical product line engineering contributes to reduce the complexity, effort and costs in performing safety analysis for a specific product variant, since such analysis is not performed from scratch. Therefore, with the support of compositional safety analysis techniques, assessment artefacts such as fault trees and FMEA results can be generated for a given product variant from the reused safety analysis.

This chapter presents a model-based approach to support the systematic reuse of product line architecture and safety models in safety-critical software product line engineering. The approach differs from traditional SPLE approaches by integrating variant management into compositional safety analysis. Such integration allows the systematic reuse of both product line architecture and safety analysis models (failure model). The approach presented in this chapter is part (see Section 3.2) of a systematic approach to integrate safety assurance into safety-critical product line engineering presented in Chapter 3. The approach comprises variability modeling and management phases in product line domain engineering and application engineering processes. This chapter also presents a method to adapt and integrating existing variant management techniques into compositional safety analysis. In this thesis, this method has been used to adapt BVR tool chain[16] and Hephaestus/Simulink. The variability management approach and tooling is validated in a Hybrid Braking System automotive safety-critical product line case study (DE CASTRO *et al.* 2011).

Section 4.2 presents the proposed approach to support variability management in architecture and safety models in software product line engineering. Section 4.3 presents a method for adapting variant management techniques to support variability management in safety models, and the adaptations performed into BVR and Hephaestus/Simulink variant management tools to support variant management in HiP-HOPS and AADL Error Annex

[13] https://wiki.sei.cmu.edu/aadl/index.php/OSATE_2_download_page

[14] http://www.mathworks.com/products/simulink/index-b.html

[15] http://www.simulationx.com/

[16] http://modelbased.net/tools/bvr-tool/

failure models. Section 4.4 illustrates the approach's instantiation using the BVR toolset and its adapters for MATLAB/Simulink and HiP-HOPS compositional safety analysis tools. Section 4.5 presents the approach's limitations, and Section 4.6 presents the summary of this chapter.

4.2 Management and Resolution of Variability in Architecture and Safety Models

This section presents the proposed model-based approach to support the systematic variability management in architecture and safety analysis models in safety-critical product line engineering. The approach has been defined from the analysis of existing variant management techniques covering architectural design (BIG LEVER, 2016; STEINER *et al.* 2013; PURE::SYSTEMS, 2016) and safety assessment (KÄβEMEYER *et al.* 2015; GOMEZ *et al.* 2011; LIU *et al.* 2007; DELHINGER *et al.* 2005), and existing product line engineering methods: Software Product Line Engineering Framework (SPLEF) (POHL *et al.* 2015) and Product Line UML-based Software Engineering (PLUS) (GOMAA, 2005).

The variability management approach has been built as an adaptation of SPLEF and PLUS methods to incorporate variability management in the safety analysis model into software product line engineering processes. The approach is holistic, which means that it is applicable with the support of different variant management and compositional safety analysis techniques, by adopting a proactive, a reactive or an extractive product line development strategy. In a proactive strategy, the product line is developed from scratch. In a reactive strategy, the development starts with a small product line, possibly consisting of a single product, which is incrementally expanded with new features and implementation artefacts. Finally, in an extractive strategy, the development starts with a collection of existing products that are incrementally refactored to obtain the product line (KRUEGER, 2002). Figure 4.1 illustrates the proposed variability management approach.

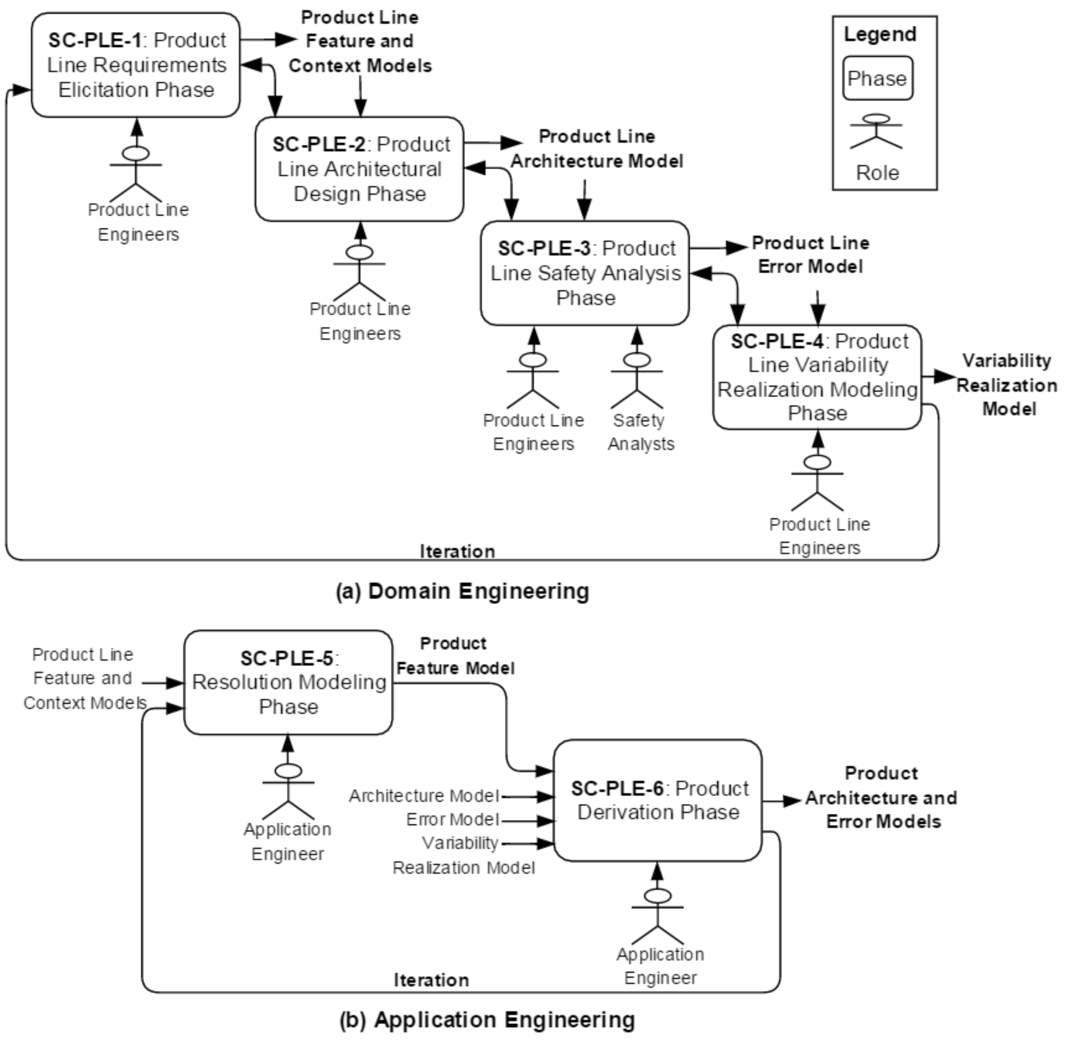

SC-PLE-1: Product Line Requirements Elicitation Phase
Product Line Feature and Context Models
SC-PLE-2: Product Line Architectural Design Phase
Product Line Architecture Model
SC-PLE-3: Product Line Safety Analysis Phase
Product Line Error Model
SC-PLE-4: Product Line Variability Realization Modeling Phase
Variability Realization Model
Product Line Engineers
Safety Analysts
Iteration
Legend
Phase
Role
(a) Domain Engineering
Product Line Feature and Context Models
SC-PLE-5: Resolution Modeling Phase
Product Feature Model
Application Engineer
Architecture Model
Error Model
Variability Realization Model
SC-PLE-6: Product Derivation Phase
Product Architecture and Error Models
Iteration
(b) Application Engineering

Figure 4.1. Model-based management of safety-related variability in product line engineering.

The proposed variant management approach distinct from traditional software product line engineering methods, e.g., PLUS and SPLEF, by considering: the impact of design and context variation in the product line hazard, risk, and component failure analysis; and management of variability in the safety model in product line variability modeling to support the systematic reuse of architectural and safety assets. The approach comprises four phases in domain engineering, and two phases in application engineering, which can be applied iteratively and incrementally. In domain engineering, during the *"Product Line Requirements Elicitation"* phase (SC-PLE-1), the product line scope is established, and mandatory, optional and alternative functional and context features are defined in the product line feature model. In this thesis, Feature-Oriented Domain Analysis (KANG *et al.* 1990) has been considered to support this phase. In the following, product line requirements defined in the feature model are used to design the product line architecture (*SC-PLE-2*), by implementing common and variable functional features. The *"SC-PLE-3: Product Line Safety Analysis"* phase is intended to identify, earlier on the design, potential threats to the system safety and applicable mitigation measures to eliminate or minimize their effects, in the scope of a set of different product variants (i.e., scenarios). Still in product line safety analysis, the component failure analysis is intended to identify how architectural components can fail and contributing to the occurrence of system-level hazards in a range of product variants. As the approach is built upon model-based design and compositional safety analysis, product line architectural design and safety analysis phases can be performed in parallel. After the identification of variability in product line architecture and safety analysis, in the *"SC-PLE-5: Product Line Variability Realization Modeling"* phase, design and context variation expressed in the feature model are linked to their realization in the architecture and safety analysis models. The variability realization model defines how architectural and safety assets are composed to derive a product variant. In this thesis, variant management tools integrated to model-based development and compositional safety analysis tools were considered to support variability management phases in domain engineering and application engineering. With the support of a variant management tool, in application engineering, variant-specific requirements are specified by choosing product line features to be included in the resolution model (SC-PLE-6), and then, the product variant is derived (*SC-PLE-7*).

Product line engineers are responsible for conducting "*Product Line Requirements Elicitation"*, "*Architectural Design"*, and "*Variability Realization Modeling"* phases. Safety analysts are responsible for conducting the "*Product Line Safety Analysis"* phase. Application engineers are responsible for defining product requirements and performing product derivation. Each phase defined in the proposed approach is supported by a method that describes the activities to be performed and their execution order. Activities are decomposed into a set of tasks. Milestones are used to indicate the end of each method and the key deliverables. The variability management approach is described with the support of SPEM (Software & Systems Process Engineering Metamodel Specification) (OMG, 2008), which is a standard notation for process specification. A method plugin has been created in the Eclipse Process Framework Composer to enable further instantiation of the approach in a particular safety-critical software product line engineering project, integrated with project management tools. The proposed approach has been previously validated in a hybrid braking system automotive product line case study (OLIVEIRA *et al.* 2014) used throughout this thesis. The approach phases are detailed in the following subsections.

4.2.1 SC-PLE-1: Product Line Requirements Elicitation

The elicitation of safety-critical product line requirements is the starting point of the proposed variability management approach. In this phase, product line engineers delimit the product line scope, by defining its degree of commonality and variability, and the number of product variants (GOMMA, 2005). From the requirements document, product line engineers also define core and variable functional features and their usage context in a feature model. Functional features represent system functions relevant for the stakeholders such as capability, operating environment and domain/implementation technique features (BRAGA *et al.* 2012). Context features define usage constraints for functional features, establishing how and where these features can be used. The input artefacts for this phase are dependent upon the adopted product line development strategy. In a proactive strategy, the product line engineer knowledge together with the target safety standard is input for this phase. If an extractive or reactive strategy is chosen, one or a set of systems from the targeted domain, together with the targeted safety standard are inputs to this phase. At the end, the product line feature and context models are delivered, indicating the completion of this phase. Feature-Oriented Domain Analysis and supporting feature modeling tools were considered to support this phase. Functional and usage context features guide the derivation of safety-critical product variants by defining the binding of common and variable architectural and safety assets. Figure 4.2 illustrates the set of activities to be performed in the "*SC-PLE-1*" phase. These activities can be performed in an incremental and iterative fashion.

SC-PLE-1.1: Product Line Scoping: Product line engineers establish the product line boundaries where the degree of common and variable functionality is defined, and a preliminary number of potential product variants are identified.

SC-PLE-1.2: Define the Target Safety Standard: Safety-critical systems should comply with process requirements/guidance established by safety standards of the targeted domain. For example, DO-178C (RTCA, 2012) and ARP 4754a (EUROCAE, 2010) define guidance for developing aircraft and aerospace systems, and ISO 26262 (ISO, 2011) defines guidance and recommendations to be addressed in the development of automotive electronics systems. Therefore, safety standards directly impacts on the development of the safety-critical product line architecture. Here, product line engineers and safety engineers analyze and choose the target safety standard to be considered throughout the product line development lifecycle. This is important to address regulatory requirements imposed by countries, compa-

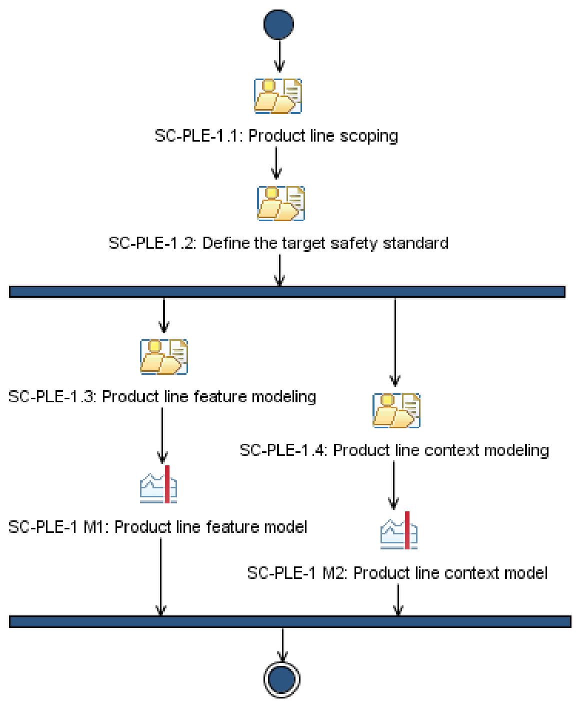

SC-PLE-1.1: Product line scoping
SC-PLE-1.2: Define the target safety standard
SC-PLE-1.3: Product line feature modeling
SC-PLE-1.4: Product line context modeling
SC-PLE-1 M1: Product line feature model
SC-PLE-1 M2: Product line context model

Figure 4.2. SC-PLE-1: product line requirements phase.

nies, and certifying authorities, and to guide the safety integrity requirements allocation process in order to achieve process-based certification.

SC-PLE-1.3: Product Line Feature Modeling: In this task, common and variable functional features are identified. Functional features can be classified into: capability, features associated with end-user visible characteristics/functions; operating environment, which capture features associated with the targeted environment where features can operate, e.g., sensors and actuators; domain/technology, which are features representing specific domain techniques or tools that can be used to develop the product line, e.g., real-time operational system feature; or implementation technique features representing specific implementation strategies, e.g., redundant and non-redundant control system architectures (BRAGA *et al.* 2012). Therefore, product line engineers identify: (i) core, (ii) optional, and (iii) alternative features. Product line engineers can also identify parameterized functional features. Parameterized features provide a parameter whose value needs to be defined at configuration time. The specification of a parameterized feature requires the definition of the type of the parameter, the range of allowed values, and a default value (GOMMA, 2005). The result of this activity is the Product Line Feature Model (SC-PLE-1 M1).

SC-PLE-1.4: Product Line Context Modeling: The functional features identified during the feature modeling may have architectural and safety-related constraints with regard to their usage. Architectural constraints define allowed and prohibited feature interactions.

Safety-related constraints define context feature interactions that may impact in the safety properties, e.g., changes in hazards and component failure modes. Examples of such constraints are: the range of applications in which a product variant can be used, the targeted hardware platform and other systems interacting with the variant, characteristics of the environment where a product variant can operate (BRAGA *et al.* 2012), and the targeted safety standard already identified in SC-PLE-1.2. Here, product line engineers identify (i) usage context features that impact in architectural constraints, and (ii) usage context features that impact in the safety properties. By combining functional and usage context features, a set of candidate variants can be derived. These variants can be further considered to perform product line safety analysis. The relationships between functional and context features directly impact in the definition of the product architecture and its safety properties, e.g., the safety integrity requirements of components. The result of this activity is a Product Line Context Model (SC-PLE-1 M2).

4.2.2 SC-PLE-2: Product Line Architectural Design

Safety-critical product line architectural design is intended to specify the realization of common and variable functional features resulting in a product line architecture composed by hierarchical models representing hardware components, e.g., sensors and actuators, service-provision software components, and their connections. In this phase, product line engineers build the product line architecture by firstly considering architectural patterns to be adopted, followed by the definition of components that represent the realization of core, optional, alternative, and parameterized functional features, which comprise both hardware and software. In the following, product line engineers choose language-specific variability mechanisms to represent optional, alternative, and parameterized features in the product line architecture model. The product line architecture model is derived indicating the completion of this phase.

Existing model-based development techniques and languages, e.g., Real-Time UML (OMG, 2008a), SysML (OMG, 2015), MATLAB/Simulink (MATHWORKS, 2015), and AADL (SAE, 2012) are recommended to support the "*Product Line Architectural Design*" phase. In this thesis, we have considered MATLAB/Simulink and AADL, which are mature model-based development environments. Architectural patterns for safety-critical systems (DOUGLASS *et al.* 2002), and existing variability patterns (WEILAND and MANHART 2014; BOTTERWECK *et al*. 2010; STEINER *et al.* 2013) can be used to support the design and specification of variability mechanisms in the product line architecture model. In addition, variability information can also be expressed externally, i.e., outside the model, with the support of variant management techniques (VASILEVSKIY *et al.* 2015; STEINER *et al.* 2013). Figure 4.3 shows the activities associated with "*SC-PLE-2: Product Line Architectural Design*" phase. The design of core and variable functional features comprises both hardware and software features and their interactions.

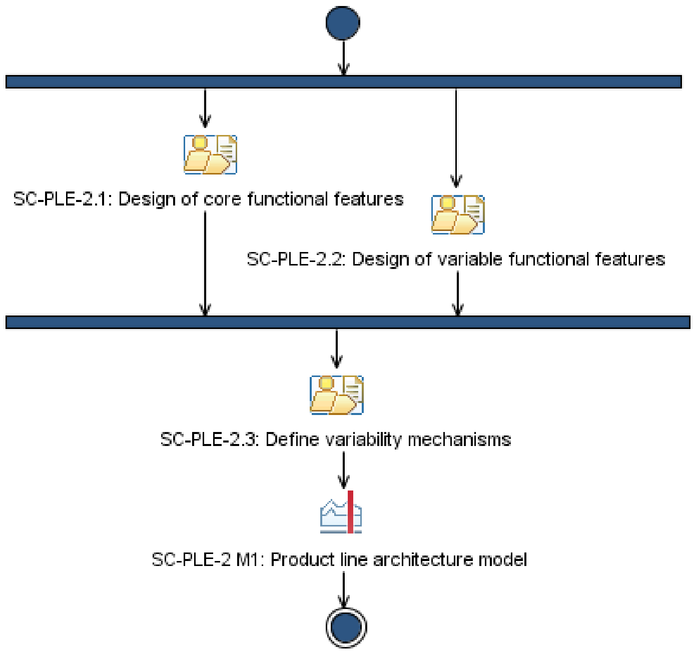

SC-PLE-2.1: Design of core functional features
SC-PLE-2.2: Design of variable functional features
SC-PLE-2.3: Define variability mechanisms
SC-PLE-2 M1: Product line architecture model

Figure 4.3 - SC-PLE-2: product line architectural design phase.

SC-PLE-2.1: Design of Core Functional Features: core functional features are implemented in the product line architecture model. Firstly, for each functional feature, systems/subsystems/components architectural elements, their communication ports and connections are specified in the structural model. Examples of language-specific architectural models are block diagrams in MATLAB/Simulink and packages in AADL. If applicable, the component behavior is specified in finite state machines. State machines define states and state transitions that may change the structural model leading to changes in component configuration modes and port values. In this activity, as shown in Figure 4.4, product line engineers specify: (i) systems, their constituent subsystems and components that represent each core functional feature, (ii) interfaces between systems/subsystems/components, i.e., data, event, event data ports, (iii) connections between architectural elements, and if applicable, (iv) state machines associated with one or a group of functional features.

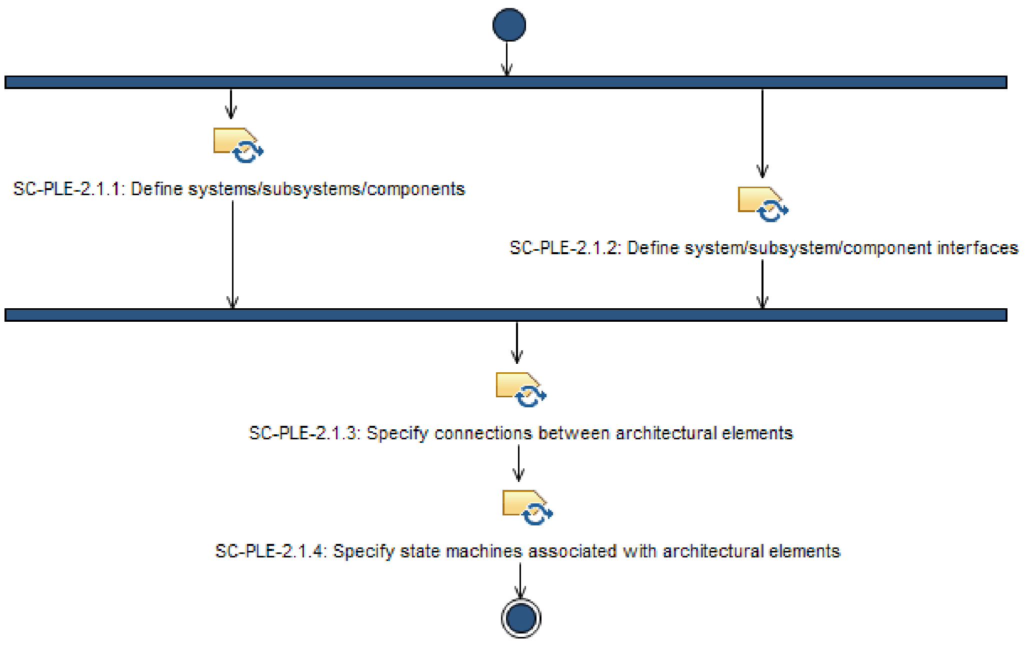

SC-PLE-2.1.1: Define systems/subsystems/components
SC-PLE-2.1.2: Define system/subsystem/component interfaces
SC-PLE-2.1.3: Specify connections between architectural elements
SC-PLE-2.1.4: Specify state machines associated with architectural elements

Figure 4.4. SC-PLE-2.1: design of core functional features.

SC-PLE-2.2: Design of Variable Functional Features: optional, alternative, and parameterized features are implemented in the product line architecture model following the same steps defined in SC-PLE-2.1 activity. In addition, for parameterized features, data components are specified and connected to parameterized systems, subsystems, or components. Examples of data components are *Constant* blocks in MATLAB/Simulink, and *Data* subcomponents in AADL.

SC-PLE-2.3: Define Variability Mechanisms: variability should be explicitly specified inside or outside the model. Internal variability mechanisms can be classified into the following types (WEILAND and MANHART, 2014):

- *Model adaptation*: model elements representing variation points, where the resolution of variability leads to a structural adaptation of the architecture model;
- *Conditional model elements:* mechanisms to represent optional (*Enabler* subsystems), alternative (*Switch*), and inclusive-or (*Integrator*) features (BOTTERWECK *et al.* 2010), variability patterns to configure features with hierarchical and dependency relationships (i.e., based on the *Enabler subsystem* and *Comparator* blocks); and patterns to configure variability in finite state machines based on: *Subsystem* and *Constant* blocks; *Enabler subsystem* and *Constant* blocks; and *Switch* with *Constant* blocks. These patterns are also applicable to structural and state machine models other than Simulink; and

- *Data variability:* variant-specific values for calculations denoted as calibration data.

External variability mechanisms are specified with the support of a variant management tool by defining mapping links between product line features and their realization in architectural assets. In this activity, the product line engineers specify variability mechanisms to represent optional, alternative and parameterized features. These mechanisms can be defined in the architecture model or outside the model with the support of a variant management tool. At the completion of this phase, the product line architecture model is delivered (SC-PLE-2 M1 in Figure 4.3).

4.2.3 SC-PLE-3 Product Line Safety Analysis

The product line safety analysis phase encompasses functional hazard assessment and component failure analysis in product line domain engineering. Functional hazard assessment should be performed from the perspective of a range of product variants and their usage, called scenarios. These scenarios are expected to assist safety analysts to precisely identify the threats to the system safety, and applicable mitigation strategies across the product line design. Such analysis also guides safety analysts in identifying and managing variability in the safety analysis, before the generation of assessment artefacts such as fault trees and FMEA results. At end of the analysis, the information about variant-specific hazards, their causes, severity, and allocated safety requirements are stored into the product line failure model. Since architectural and contextual variability directly affect safety properties, component failure analysis is intended to identify how architectural components can fail and contribute to the occurrence of hazards in a range of usage scenarios. At the completion of this phase, the product line failure model with hazard and risk analysis, and component failure data is delivered.

In this thesis, we prescribe a set of techniques to support product line safety analysis phase. Use case-based analysis techniques (ALLENBY and KELLY, 2001) and sequence diagrams (FENG and LUTZ, 2005) can be used to support the identification of relevant scenarios to perform functional hazard assessment. The ICPL algorithm (JOHANSEEN *et al.* 2012) supports combinatorial analysis of feature models for generating relevant product variants to be considered in software product line testing. ICPL algorithm can be used to perform combinatorial analysis of safety-critical product line feature model to support the automatic identification of critical variants to be considered during the product line safety analysis. In this thesis, HiP-HOPS (PAPADOPOULOS *et al.* 2011), and AADL Error Annex

(DELANGE and FEILER, 2014) compositional safety analysis techniques were considered and used to support product line functional hazard assessment and component failure analysis. "*Product Line Safety Analysis"* is detailed in Chapter 5.

4.2.4 SC-PLE-4: Product Line Variability Realization Modeling

Variability realization modeling is intended to establish mapping links between functional and usage context features and their realization in architectural components and safety analysis assets in the product line variability realization model. In safety-critical software product line engineering, these mapping links describe how architecture and safety assets can be bound to assembly variant-specific architecture and failure models.

Feature and context models provide the main information required to specify the variability realization model for a safety-critical product line. In this phase, as shown in Figure 4.5, product line engineers define:

i) A set of composition rules to derive a range of product variants. These rules consist in different combinations between functional and context features grouped into feature expressions that represent product variants (SC-PLE-4.1). Each feature expression represents an architectural or a safety-related variant, e.g., a variant associated with a variation point defined in the product line safety analysis model;

ii) Mapping links between feature expressions (i.e., design variants) and their realization in the product line architecture model such as systems, subsystems, components, their ports and connections, data parameters, and finite state machines associated with these structural elements, considered to be included (replacement) and excluded (placement) when functional variants are selected (SC-PLE-4.2); and

iii) Mapping links between feature expressions representing safety-related variants and safety-related information, such as hazards and their causes, and component failure analysis stored into the product line failure model, considered to be included and excluded when variability in the safety model is resolved for the chosen safety-related variant specified in the resolution model (SC-PLE-4.3). After performing these tasks, the binding points for the realization of architectural and safety-related variants are stored into the variability realization model and delivered (SC-PLE-4 M1), and this phase is completed. These binding points support the derivation of variant-specific architecture and failure models. The reused safety model contributes to reduce the complexity of variant-specific safety analysis, and allows the automatic generation fault trees and FMEA results from the reused safety assets as detailed in Chapter 5.

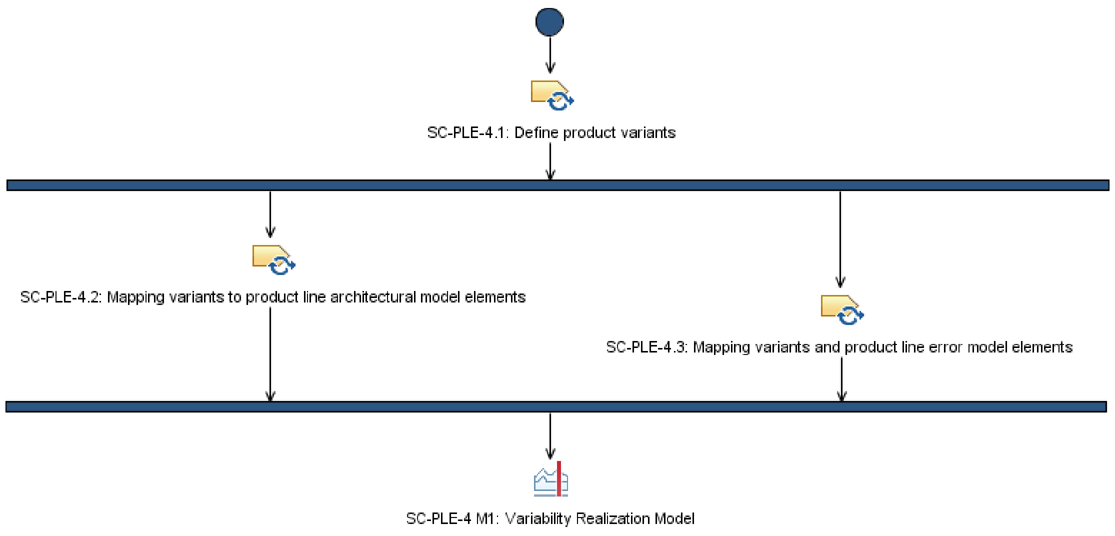

SC-PLE-4.1: Define product variants
SC-PLE-4.2: Mapping variants to product line architectural model elements
SC-PLE-4.3: Mapping variants and product line error model elements
SC-PLE-4 M1: Variability Realization Model

Figure 4.5. SC-PLE-4: product line variability realization modeling phase.

In this thesis, variant management techniques were considered to support the variability realization modeling phase. Pure::variants (PURE::SYSTEMS, 2016) provides support for negative variability (CLEMENTS and NORTHROP, 2001), i.e. the product derivation is based on the activation and deactivation of model elements according to the selection of product variants. Model transformation tools, e.g., BVR (VASILEVSKIY *et al.* 2015) and Hephaestus/Simulink (STEINER *et al.* 2013), provide support for positive variability, meaning that product derivation yields variant-specific models that contain only model elements that correspond to the chosen variants. Existing variant management techniques do not provide mechanisms to support variability management in the safety model. Therefore, their variability management mechanisms should be adapted by extending the variability realization model to support the management of variability in the safety model. The steps to integrate variant management into compositional safety analysis are detailed in Section 4.3.

SC-PLE-4.1: Define product variants: from the analysis of functional and context features, feature model constraints, and usage scenarios used to conduct product line safety analysis, product line engineers define architectural and safety-related variants. These variants can be represented by feature expressions. A feature expression can be composed by one or more features connected via “AND”, OR”, “NOT” logical operands required to describe a given variant. Variants are mapped to transformations to be performed into the product line architectural and failure models during the *"SC-PLE-7: Product Derivation"* phase. Firstly, application engineers define variants associated with functional features, and safety-related variants associated with context features are defined later.

SC-PLE-4.2: Mapping variants to product line architecture model elements: after specifying functional variants, product line engineers define, for each functional variant, the architectural model elements to be (i) included and/or (ii) excluded, when functional variants are resolved during the product derivation phase.

SC-PLE-4.3: Mapping variants to product line failure model elements: safety-related variants should be linked to their realization in the product line failure model. In this task, product line engineers and safety analysts define how safety-related variants can be composed for deriving different product variants. For each usage scenario considered during the product line safety analysis, i.e., product variant and its associated usage context, the following is defined: (i) the functional hazard assessment data to be included, and (ii) for each architectural component that realizes a given functional variant, the corresponding failure data associated to that component is considered to be included when safety-related variants are resolved in product derivation phase. Thus, the variability realization model is delivered and this phase is completed (SC-PLE-4 M1).

4.2.5 SC-PLE-5: Resolution Modeling

Resolution modeling is the first variability management phase in application engineering. Resolution modeling is intended to choose product line architectural and safety-related variants that address the requirements of a particular product variant. Based on the product requirements, the application engineer creates a product feature model by selecting functional and usage context features. Then, the product feature model (i.e., the resolution model) is delivered and this phase is completed.

4.2.6 SC-PLE-6: Product Derivation

After specifying the rules to bind architectural and safety analysis assets to design and context features in the variability realization model, and defining the resolution model, variability is resolved with the support of a variant management tool, and a product variant is derived. In existing variant management tools such as BVR and Hephaestus/Simulink (STEINER *et al*. 2013), the product line engineer should provide the following input artefacts for product derivation: product line feature model (FM - Feature Model), the resolution model (IM - Instance Model), product line architecture and failure models (PLA - Product Line Assets), and the product line variability realization model (CK - Configuration Knowledge) as illustrated in Figure 4.6. The feature model specifies common and variable functional and context features. Instance model defines variant-specific functional and context features. Product line assets represent architectural model elements, e.g., blocks in MATLAB/Simulink or AADL system components, and failure behavior information. The variability realization model establishes mapping links between features to their realization in product line architecture and failure models. Whereas existing variant management techniques do not provide native support for variability management in the safety model, they should be adapted by following a set of guidelines presented in section 4.3. In this thesis we have adapted the BVR tool to support variability management in HiP-HOPS and OSATE AADL Error Annex failure models, and Hephaestus/Simulink to manage variability in the HiP-HOPS failure model.

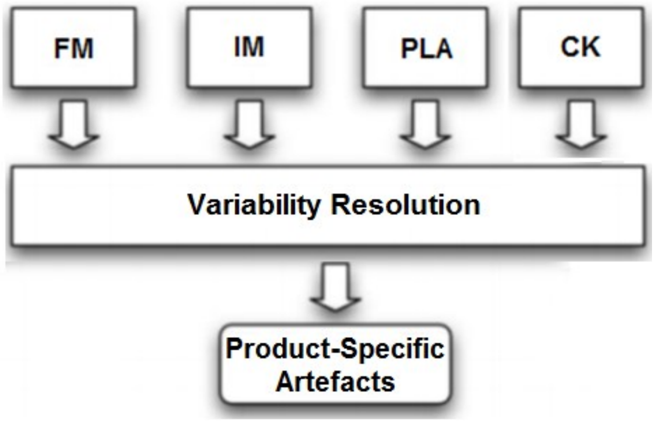


Figure 4.6. Product line variability resolution process (Adapted from STEINER *et al.* 2013).

Variability resolution in the BVR (VASILEVSKIY *et al.* 2015) and Hephaestus/Simulink (STEINER *et al.* 2013) tools usually starts with evaluation of features specified in the instance model and feature expressions (i.e., variants) specified in the variability realization model (CK). Model transformations associated with variants evaluated as "*true*", i.e., when model elements have to be included in the final product, and variants evaluated as "*false*", i.e., when model elements have to be excluded in the final product, are applied to the product line architecture and failure models. The output of this phase is a variant-specific architecture and failure models, e.g., a MATLAB/Simulink model enhanced with HiP-HOPS failure annotations, or an AADL model and its respective failure model.

Variant-specific compositional safety analysis can be performed from the reused architecture and failure models, to generate fault trees and FMEA safety assessment artefacts.

Variant-specific requirements not covered by product line architecture can be developed and integrated to the derived product architecture. The addition of functional requirements to a given product variant requires performing safety analysis, in the same way as described in the SC-PLE-3 phase, to identify potential hazards that can emerge from the interactions between newer system functions and reused architectural components. After that, safety assessment artefacts such as fault trees and FMEA results can be generated, with the support of compositional safety analysis techniques, from the reused safety analysis information. Additionally, variant-specific system functions and their associated safety analysis information can be further incorporated to the product line asset base. In this case, product line engineers should update the product line feature (SC-PLE-1), architecture (SC-PLE-2), failure (SC-PLE-3 and SC-PLE-4), and variability realization models (SC-PLE-4) in domain engineering to support the systematic reuse of the newer functions and their safety-related data.

4.3 Integrating Variant Management into Compositional Safety Analysis

Existing variant management techniques and tools were not originally designed to address variability management in safety analysis models. Therefore, to support the management and resolution of safety-related variability found in safety models, variant management tools should be adapted to support the specification of mapping links between features and their realization in the safety model in the variability model. In addition, it is also necessary to adapt their variability resolution process to support the resolution of variability in the safety model during product derivation phase. This section presents a method to adapt variant management tools to support the variability management in safety models. The proposed method was extracted from the analysis of existing variant management tools, which it has been identified that they share similar concepts to manage variability in product line assets.

Table 4.1 shows the identified concepts shared by four different variant management tools. In both tools, variability is specified in the feature model, mapping links between features and their realization in product line assets are defined in the variability realization model, and the instance model is obtained by choosing variants defined in the feature model. These tools provide pluggable interfaces to enable support for variability management in different product line assets. For example, pure::variants (PURE::SYSTEMS, 2016) provides adapters to support variability management in different models such as UML/SysML and MATLAB/Simulink. Hephaestus/Simulink is a connector plugin that implements the Hephaestus (TURNES *et al.* 2012; BONIFÁCIO *et al.* 2009) adapter interface to support variability management in MATLAB/Simulink models. BVR (VASILEVSKIY *et al.* 2015) and CVL (HAUGEN *et al.* 2008) provide adapter interfaces that can be implemented to enable variability management in different models from different languages based on the Eclipse Modeling Framework platform.

Table 4.1 – Input models and product line tools.

Product Line Tool/ Model	Feature Model	Instance Model	Variability Realization Model	Asset Model
BVR/CVL	Variability Specification Model	Variability Resolution Model	Variability Realization Model	Abstract Asset Model Interface, Default Connector: Papyrus UML2
Hephaestus-Simulink	Feature Model	Instance Model	Configuration Knowledge	Abstract Asset Model Interface, Simulink Connector
pure:variants	Feature Model	Instance Model	Variant Model	Abstract Asset Interface Connectors: Simulink, C++, Papyrus UML2

Figure 4.7 shows the core structure of a variant management tool, created based on the concepts shared by variant management tools shown in Table 4.1. The core module comprises the variability realization model, variability specification model, variability resolution model, and asset model components, and a variability resolution control component. The "*Variability Resolution*" component provides a set of transformations to be applied to product line assets during product derivation. The "*Variability Resolution*" component supports product derivation by analyzing the information provided by variability specification, resolution, and realization models, to apply transformations into product line assets, e.g., requirements, architecture, to derive variant-specific models during product derivation phase. The "*Variability Resolution*" component is connected to the variability specification, resolution and realization data components via "*Data Access*" interfaces. "*Variability Specification*" and

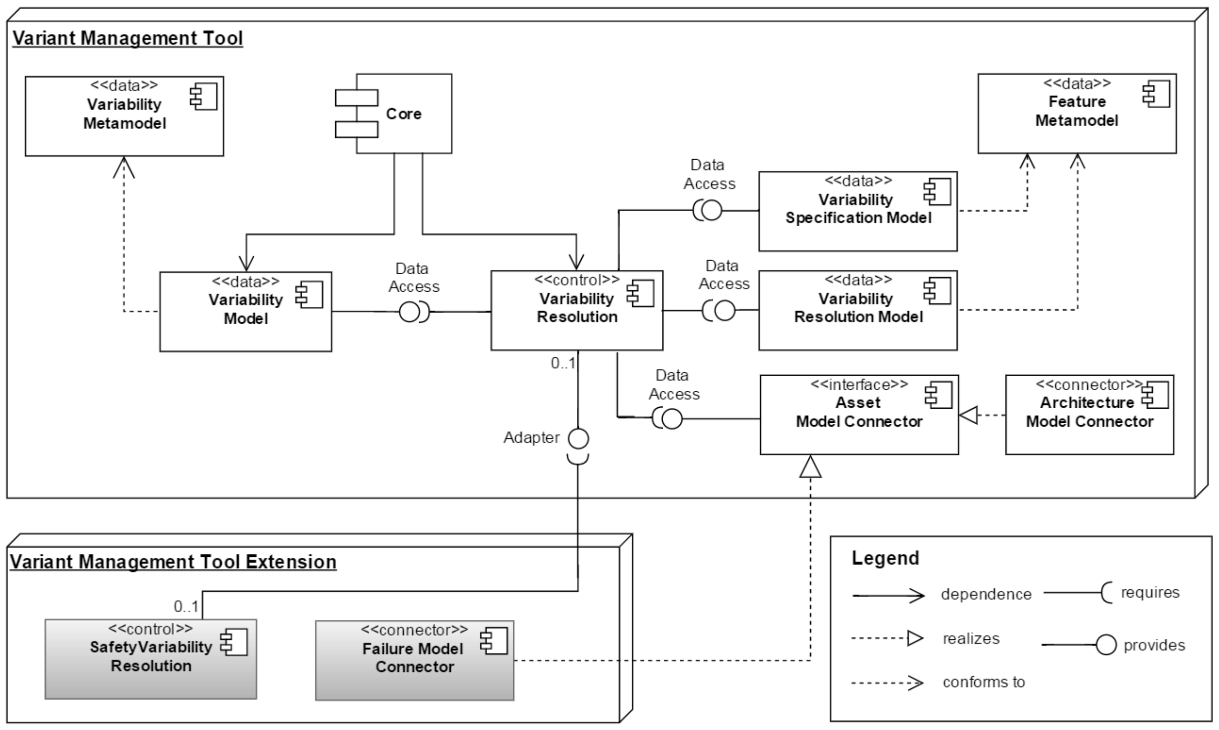

Variant Management Tool
<<data>> Variability Metamodel
Core
<<data>> Feature Metamodel
Data Access
<<data>> Variability Specification Model
<<data>> Variability Model
Data Access
<<control>> Variability Resolution
Data Access
<<data>> Variability Resolution Model
0..1
Data Access
<<interface>> Asset Model Connector
<<connector>> Architecture Model Connector
Adapter
Variant Management Tool Extension
0..1
<<control>> SafetyVariability Resolution
<<connector>> Failure Model Connector
Legend
dependence
requires
realizes
provides
conforms to

Figure 4.7. Extending product line tools to support variability management in the safety model.

"*Resolution*" models conform to a "*Feature*" metamodel, which may vary from a variant management tool to another. For example, in BVR, variability specification and resolution models comply with FeatureIDE metamodel (CZARNECKI *et al.* 2004). The variability realization model also conforms to a tool-specific variability metamodel. Variability metamodels provided by existing variant management tools conform to the concepts of "*Variation Point*", "*Variant*", and relationships between variants and product line assets defined in the generic variability metamodel proposed by Bachmann *et al.* (2003) detailed in Section 2.5.1.1.

Variant management tools such as BVR, CVL, Hephaestus, and pure::variants provide an "*Asset Model Connector"* interface for interoperability between different product line assets. It allows the development of adapters to support variability management in different types of product line assets. In general, variant management tools provide connectors for requirements, design, architecture (i.e., "*Architectural Model Connector*" shown in Figure 4.7), component, source code, or test case artefacts. Implementations of "*Asset Model Connector"* interface provide the structure of the product line assets, which variability should be managed, and parsers that allow variant management tools handling these assets during product derivation.

Adapting a variant management tool requires: defining the structure and a parser for the safety analysis model (failure model) in which variability is intended to be managed ("*Failure Model Connector*" in Figure 4.7), and in some cases, e.g., Hephaestus, defining model transformations to be applied to the targeted safety model during product derivation process ("*Safety Variability Resolution*" component in Figure 4.7). BVR and CLV variant management tools do not require the definition of transformations to be applied to the target safety analysis model since their variability resolution mechanisms are capable of interfacing with third-party editors via "*IBVREnabledEditor*" interface. Therefore, the targeted compositional safety analysis editor ("*Failure Model Connector*" in Figure 4.7) has to implement this interface or providing adapters for linking the target compositional safety analysis editor to the BVR toolset. BVR is detailed in Section 2.5.3.1. The BVR architecture supports seamless integration with different model editors in the Eclipse Modeling Framework platform, e.g., OSATE AADL/Error Annex, Papyrus UML. The "*IBVREnabledEditor*" adapter interface provided by BVR is equivalent to the "*Asset Model Connector*" interface shown in Figure 4.7.

The proposed method is applicable to orthogonal variant management techniques whose variability is documented in a dedicated model, i.e., in the product line variability model (METZGER and POHL, 2014). In this thesis, we have applied the method presented in this section to adapt the BVR toolset to support variability management in MATLAB-Simulink/HiP-HOPS, and OSATE AADL/Error Annex models; and Hephaestus/Simulink to support variability management in HiP-HOPS failure model. The following sections describe the adaptations performed in BVR and Hephaestus/Simulink variant management tools. BVR and Hephaestus/Simulink were chosen by interfacing with MATLAB/Simulink and Eclipse Modeling Framework platform and by being open source.

4.3.1 BVR Adapters

In this thesis, adapters have been developed to integrate BVR variant management tool to manage variability in architectural and safety analysis models developed with the support of MATLAB-Simulink/HiP-HOPS and OSATE AADL/Error Annex model-driven development and compositional safety analysis tools. Figure 4.8 illustrates the relationships between the developed adapters, third-party modeling tools, and the BVR toolset. The BVR adapters extend third-party modeling editors by implementing the "*IBVREnabledEditor*" interface to allow the BVR toolset communicating with third-party model editors to manage

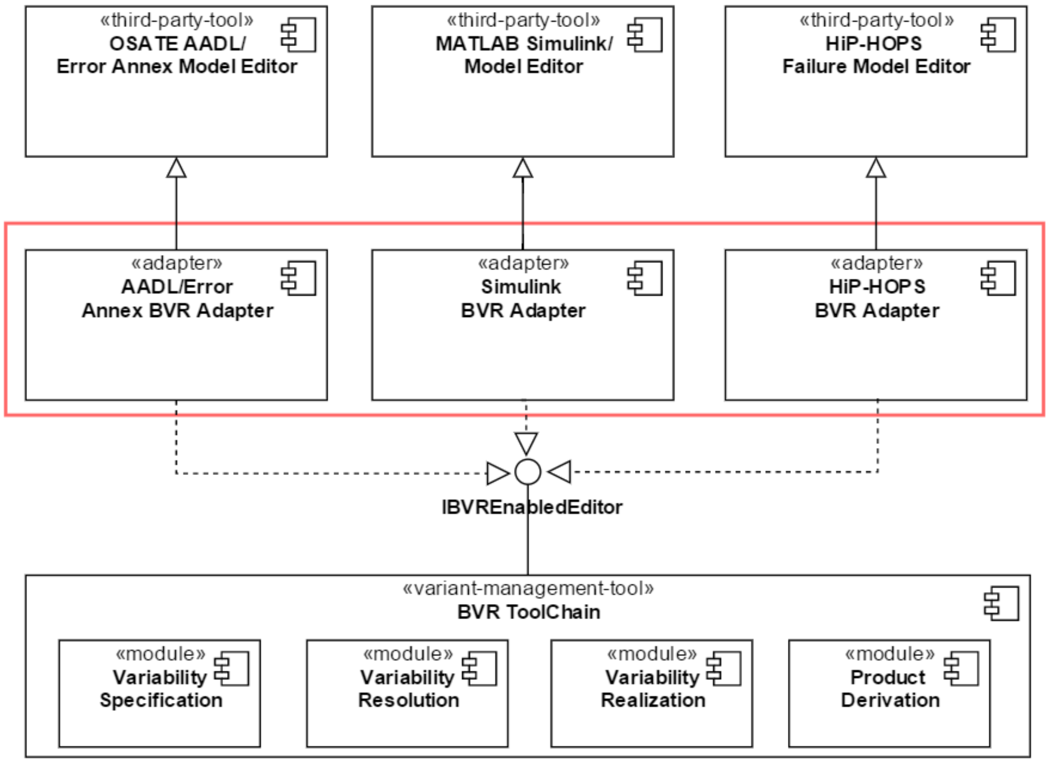


Figure 4.8. BVR adapters to support variability management in compositional safety analysis.

variability in architecture and failure models developed with the support of these editors. Whereas BVR toolset is built upon the EMF platform (ECLIPSE, 2016), the developed BVR adapters were implemented as Eclipse-based plugins.

Figure 4.9 shows the UML class diagram for the HiP-HOPS BVR adapter. "*HiphopsEditor*" extends "*MultipageEditorPart"* abstract class provided by the EMF platform, which represents the tree view editor for the HiP-HOPS failure model. The "*HiphopsBVRAdapter*" extends the HiP-HOPS model editor by implementing the "*IBRVEnabledEditor*" interface to allow the BVR toolset communicating with HiP-HOPS fai-

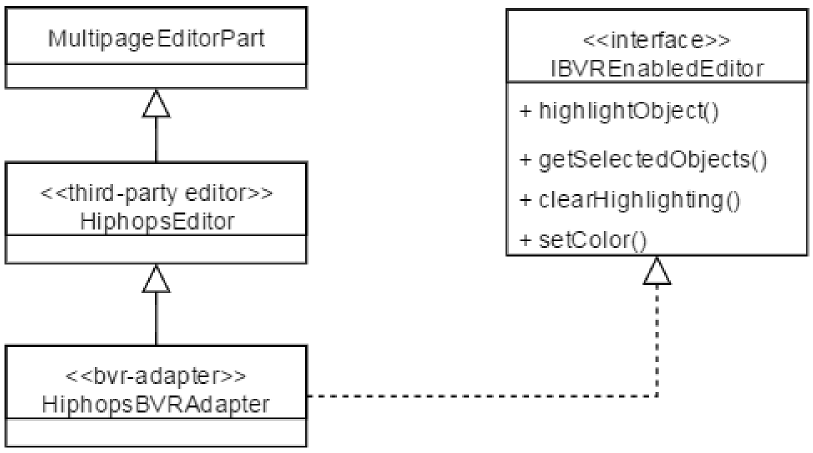


Figure 4.9. HiP-HOPS BVR adapter UML class diagram.

lure model editor. "*HiphopsBVRAdapter"* provides concrete implementations for abstract methods defined in the BVR adapter interface to allow the BVR toolset handling HiP-HOPS model elements during variability modeling and product derivation. The full source code for the "*HiphopsBVRAdapter*" class is available in Appendix A. MATLAB/Simulink and OSATE AADL/Error Annex BVR adapters have the same structure defined in the class diagram shown in Figure 4.9. The detailed description to develop adapters for the BVR toolset can be found elsewhere (VASILEVSKYI *et al.* 2015).

4.3.2 Hephaestus/Simulink HiP-HOPS Extension

The architecture of Hephaestus/Simulink comprises the following modules: "*Core*", "*Feature Modeling*", "*Configuration Knowledge*", "*SPL Assets*" and "*Transformations*" illustrated in Figure 4.10. "*Feature Modeling*" module defines the structure for the product line feature and instance models and also provides a model parser for the feature model. The "*SPL Assets"* module defines the structure for the Simulink model in a data type (metamodel), and it provides a parser for this model. The "*Configuration Knowledge*" module defines the structure for the Hephaestus variability model in a data type, and it provides a parser for this model. The "*Transformations*" module defines models transformations to be applied to a Simulink model when variability is resolved during product derivation. Finally, the "*Core*" module provides mechanisms for managing variability in the Simulink model. The configuration knowledge, feature model, instance model, and the Simulink models are input artefacts to the Hephaestus/Simulink core module resolving variability during the product derivation process.

Since HiP-HOPS compositional safety analysis tool is integrated to the MATLAB/Simulink tool, both system design and safety analysis is stored into the Simulink model. In order to support variability management in the HiP-HOPS failure model, represented by annotations in the Simulink model, the Simulink data type and model parser defined in the Hephaestus/Simulink were extended with the addition of information about the structure of the HiP-HOPS failure model such as identifiers for *hazard* and *component failure data*. Simulink model transformations, detailed in Section 2.5.3.2, have also been changed to support variability management in HiP-HOPS failure model stored into the Simulink model. In the same way as in BVR tool, the adaptation of the Hephaestus/Simulink did not require changing its variability model and variability resolution process.

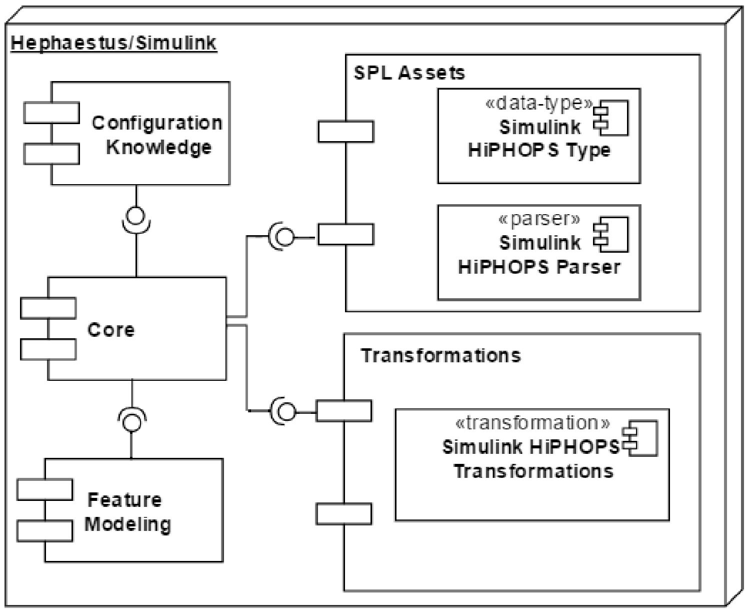

Hephaestus/Simulink
Configuration Knowledge
Core
Feature Modeling
SPL Assets
«data-type» Simulink HiPHOPS Type
«parser» Simulink HiPHOPS Parser
Transformations
«transformation» Simulink HiPHOPS Transformations

Figure 4.10. Hephaestus/Simulink variant management tool.

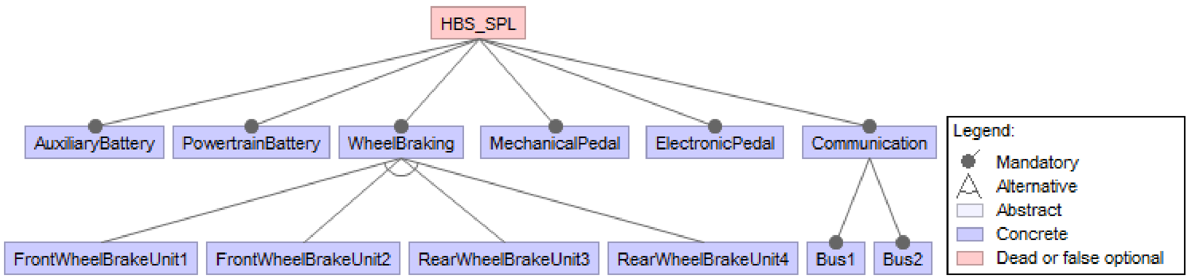
4.4 Variability Management Case Study: Hybrid Braking System SPL

The safety-related variability management approach and tooling presented in this chapter was validated in a Hybrid Braking System (DE CASTRO *et al.* 2010) automotive product line (HBS-SPL) case study. The HBS-SPL was not originally designed as a product line, but it was adapted by adopting an extractive product line development strategy (CLEMENTS and NORTHROP, 2001). Three different HBS-SPL variants were considered in this case study: four wheel braking (4WB), front wheel braking (FWB), and rear wheel braking (RWB). HBS-SPL safety analysis was performed by considering these three variants. HBS-SPL architecture and failure models were specified with the support of MATLAB/Simulink model-based development environment and HiP-HOPS compositional safety analysis tool integrated to MATLAB/Simulink. The BVR variant management toolset, together with BVR adapters for MATLAB/Simulink and HiP-HOPS developed in this thesis, were used to support variability management in the HBS-SPL architecture and failure models. The following sections present the HBS-SPL variability modeling, management and resolution phases.

4.4.1 SC-PLE Phase 1: HBS-SPL Feature Modeling

The Hybrid Braking System (HBS) addresses electric vehicles system integration, in particular propulsion architectures that integrate one electrical motor per wheel (DE CASTRO *et al.* 2010). The term hybrid comes from the fact that braking is achieved through the combined action of the electrical In-Wheel Motors (IWMs) and frictional Electromechanical Brakes (EMBs). One of the most important features of this system is that the integration of IWM in the braking process enables an increase in the vehicle's range: while braking, IWMs work as generators and transform the vehicles kinetic energy into electrical energy that feed the powertrain battery. HBS was developed based on ISO 26262 functional safety for automotive passenger cars safety standard. From the analysis of the HBS (DE CASTRO *et al.* 2011) MATLAB/Simulink architecture model, by adopting an extractive strategy, common and variable wheel braking system functions have been identified. Figure 4.11 shows the HBS-SPL feature model in Feature-Oriented Domain Analysis notation specified using the Feature IDE[17] modeling tool.

The following core features have been identified: "*MechanicalPedal*", a hardware device aimed to capture the driver presses; "*ElectronicPedal*", a hardware device that senses and processes the actions from the mechanical pedal; "*Bus1*" and "*Bus2*", which are software components that send wheel braking forces to the wheel braking units; "*Auxiliary Battery*", a hardware device responsible for feeding, with energy, the electromechanical brake units while braking; and "*Powertrain Battery*", which stores the electrical energy produced by in-wheel motors. Variability has been identified in the wheel braking capability denoted by the "*Wheel-*



¬FrontWheelBrakeUnit1 ∧ (RearWheelBrakeUnit3 ∨ RearWheelBrakeUnit4)

¬FrontWheelBrakeUnit2 ∧ (RearWheelBrakeUnit3 ∨ RearWheelBrakeUnit4)

¬RearWheelBrakeUnit3 ∧ (FrontWheelBrakeUnit1 ∨ FrontWheelBrakeUnit2)

¬RearWheelBrakeUnit4 ∧ (FrontWheelBrakeUnit1 ∨ FrontWheelBrakeUnit2)

Figure 4.11. HBS-SPL feature model.

[17] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

Braking" feature group, which comprises four wheel braking alternative features: "*FrontWheelBrakeUnit1"* and "*FrontWheelBrakeUnit2",* for left and right front wheel braking subsystems, and "*RearWheelBrakeUnit3"* and "*RearWheelBrakeUnit4"*, for left and right rear wheel braking subsystems. Wheel braking features can be composed into different ways, deriving different automotive braking system variants. Constraints are present in the HBS-SPL feature model to show how wheel braking features can be composed in a product variant. For example, ¬ *FrontWheelBrakeUnit1* ^ (*RearWheelBrakeUnit3 v RearWheelBrakeUnit4*) constraint denotes that "*FrontWheelBrakeUnit1"* and "*RearWheelBrakeUnit3"* is an invalid configuration choice, and that "*FrontWheelBraking1", "FrontWheelBraking2"* and "*RearWheelBraking3"* is a valid configuration choice. These constraints delimit the variability scope to a set of product variants. These variants were defined in the product line usage context model.

The definition of product variants and contextual elements that may impact in the safety properties is important to achieve the systematic reuse of both architecture and safety analysis in application engineering. This is done by associating functional variants, i.e., a combination between functional features, to their possible usage defined in context features. Such analysis provides a set of variants to be further considered to support the systematic reuse of components and safety analysis. For HBS-SPL, seven product variants have been identified as shown in Figure 4.12. These variants differ from each other in the number of wheel brake units and the way as they are combined. For example, "*Front Wheel Braking"* (FWB) variant differs from "*Four Wheel Braking"* (4WB) variant by the absence of rear wheel brake units. "*Rear Wheel Braking"* (RWB) variant differs from "*4WB*" by the absence of front wheel brake units. The identified product variants can be deployed in a car, truck, or a military automotive vehicle. Such variation was also defined in the usage context model. The composition of wheel braking and vehicle variants leads to 21 different HBS-SPL product va-

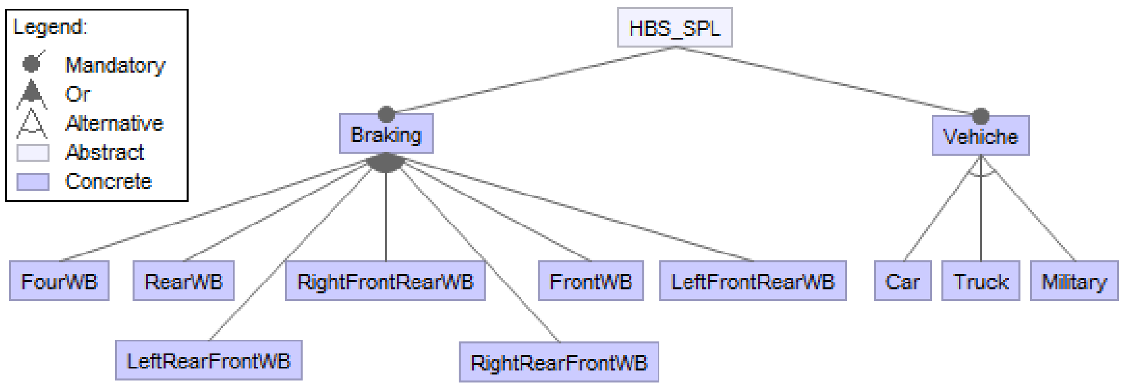


Figure 4.12. HBS-SPL usage context model.

riants, i.e., combinations between system variants and their usage contexts.

4.4.2 SC-PLE Phase 2: HBS-SPL Architectural Design

HBS-SPL was designed in MATLAB/Simulink. Figure 4.13 illustrates the HBS-SPL architecture model in a SysML block diagram. The HBS-SPL architecture comprises 30 model components: 4 wheel braking unit subsystems (one per wheel), which each one comprises 6 components, 1 electronic pedal component, 1 mechanical pedal component, 2 battery components, and 2 communication buses components. "*Communication buses"* represent a duplex bus communication system that sends the wheel braking forces to the wheel brake units. "*Auxiliary Battery"* is a hardware device responsible for feeding the electromechanical brakes while braking, and "*Powertrain Battery"* stores the electrical energy produced by "*In-Wheel Motors"*.

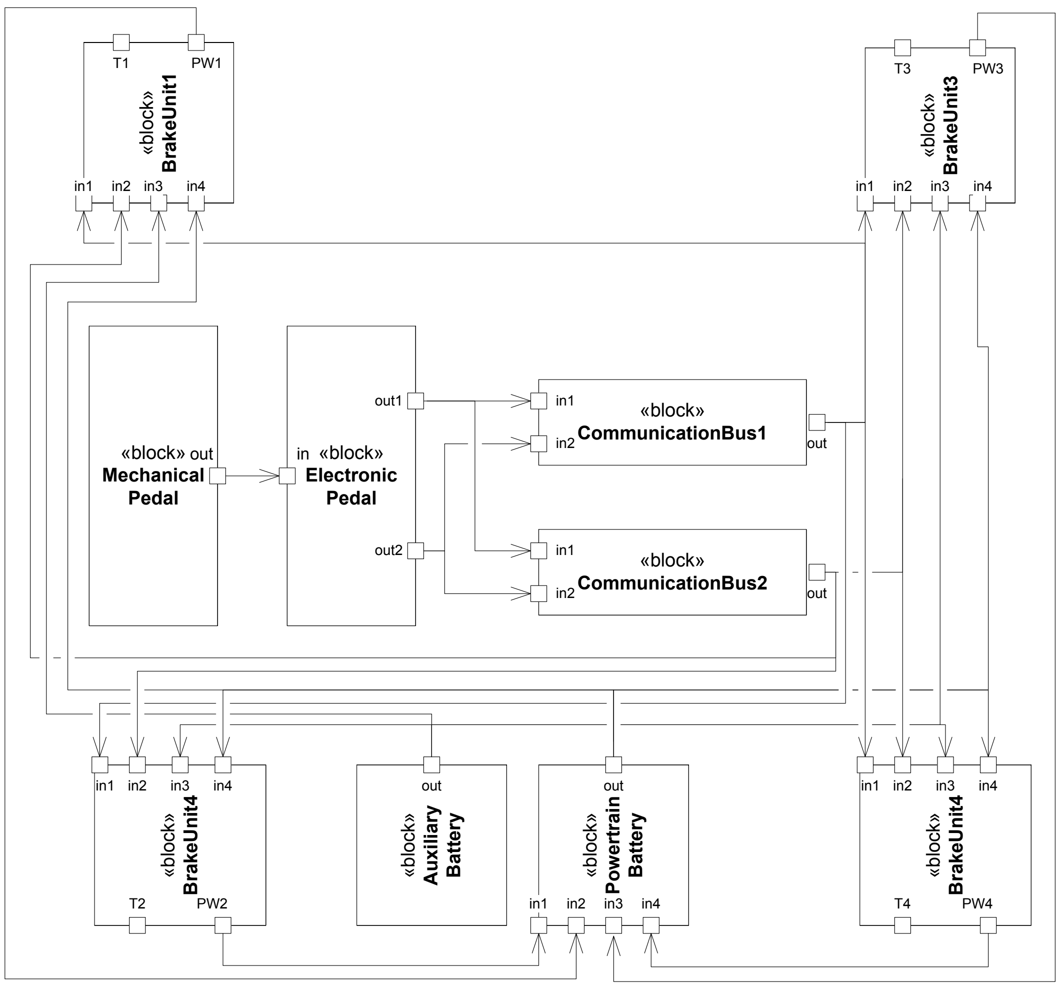


Figure 4.13. HBS-SPL architecture model.

Each wheel "*Brake Unit"* integrates a "*Wheel Node Controller"* (Figure 4.14), which calculates the amount of braking torque to be produced by each wheel braking actuator, and it sends commands to "*EMB"* and "*IWM"* power converters that control "*EMB"* and "*IWM"* braking actuators. "*In-Wheel Motor"* (IWM) actuator decreases the vehicle kinetic energy converting it into electrical energy. IWMs have, however, braking torque availability limitations at high wheel speeds or when the "*Powertrain Battery"* is close to full state of charge. Thus, the "*Electromechanical Braking"* (EMB) actuator is used dynamically with IWMs to provide the required braking torque to address the total braking demand. While braking, the electric power flows from the "*Auxiliary Battery"* to *"EMB"* via "*EMB Power Converter"*; and IWM acts as a power generator providing energy for the "*Powertrain Battery"* via "*IWM Power Converter"*. Finally, "*Add"* component outputs the braking torque and the generated power while braking.

The way in which wheel brake unit subsystems are connected to each other, auxiliary and powertrain batteries components might change according to the selection of wheel braking alternative features specified in the HBS-SPL feature model. Such variation is not expressed in the HBS-SPL architecture model; it was defined outside the model with the support of BVR (VASILEVSKYI *et al.* 2015) variant management tool (see section 4.4.4).

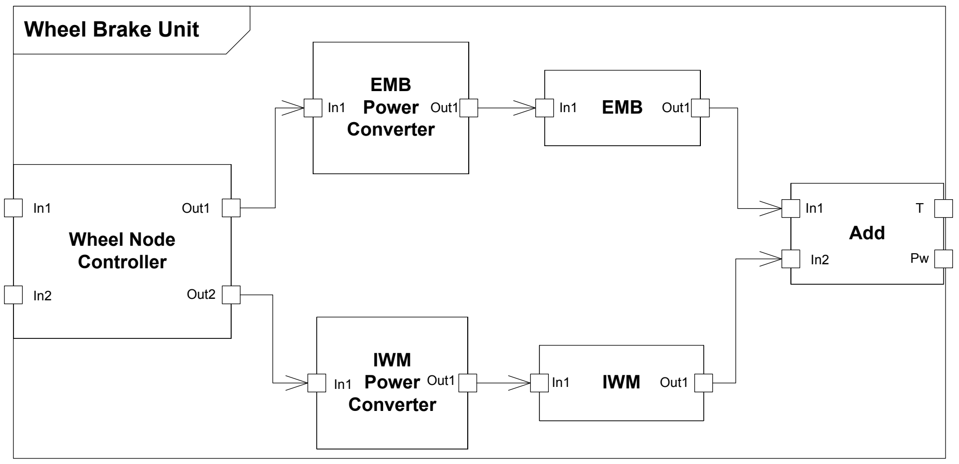


Figure 4.14. Wheel brake unit subsystem.

4.4.3 SC-PLE Phase 3: Product Line Safety Analysis

Functional hazard assessment and component failure analysis were performed by considering the following usage scenarios: "*4WB*", "*FWB*", and "*RWB*". The analysis was constrained to these scenarios to reduce the complexity, since performing safety analysis by considering all HBS-SPL variants would be prohibitive in the real world. HBS-SPL variants were analyzed from the safety perspective by considering the ISO 26262 hazard and risk assessment processes. The extended HAZOP (*HaZard and OPerability*) analysis technique supported by the HiP-HOPS compositional safety analysis tool (PAPADOPOULOS *et al.* 2011), was used to perform the HBS-SPL safety analysis.

Variability in the safety analysis model can be found in functional hazard assessment data, as shown in Table 4.2. Different failure conditions may cause the "*Value braking*" hazard in "*4WB*", "*FWB*" and "*RWB*" system variants. This example shows how architectural variation might directly impact in the safety properties. In the "*4WB*" system variant, "*incorrect values of all brake unit actuator outputs*" can cause this hazard. On the hand, "*incorrect values of front wheel braking actuator outputs*" lead to the occurrence of this hazard in the "*FWB*" variant. The impact of architectural variation in the safety properties is further propagated throughout component deviations that may contribute to the occurrence of system-level failures identified during component failure analysis. Different components may contribute to hazards in different product variants, as illustrated in Table 4.3. "*BrakeUnit3.IWMPowerConverter"* component failures contribute to hazards in "*RWB*" variant, and "*BrakeUnit1.IWM"* failures contribute to hazards in "*FWB*" variant. Product line safety analysis information is stored into the failure model. HBS-SPL safety analysis is further detailed in Chapter 5.

Table 4.2 - Variability in product line functional hazard assessment.

Variant	**Hazard ID**	**Failure Condition**	**ASIL**
Four Wheel Braking (4WB) Deployed into an Automotive Car Vehicle	4WB_Value_braking	Incorrect Value of all brake unit actuator outputs	D
Front Wheel Braking (FWB) Deployed into an Automotive Car Vehicle.	FWB_Value_braking	Incorrect Value of BrakeUnit1 AND BrakeUnit2 actuator outputs.	D

Table 4.3 – Variability in product line component failure logic.

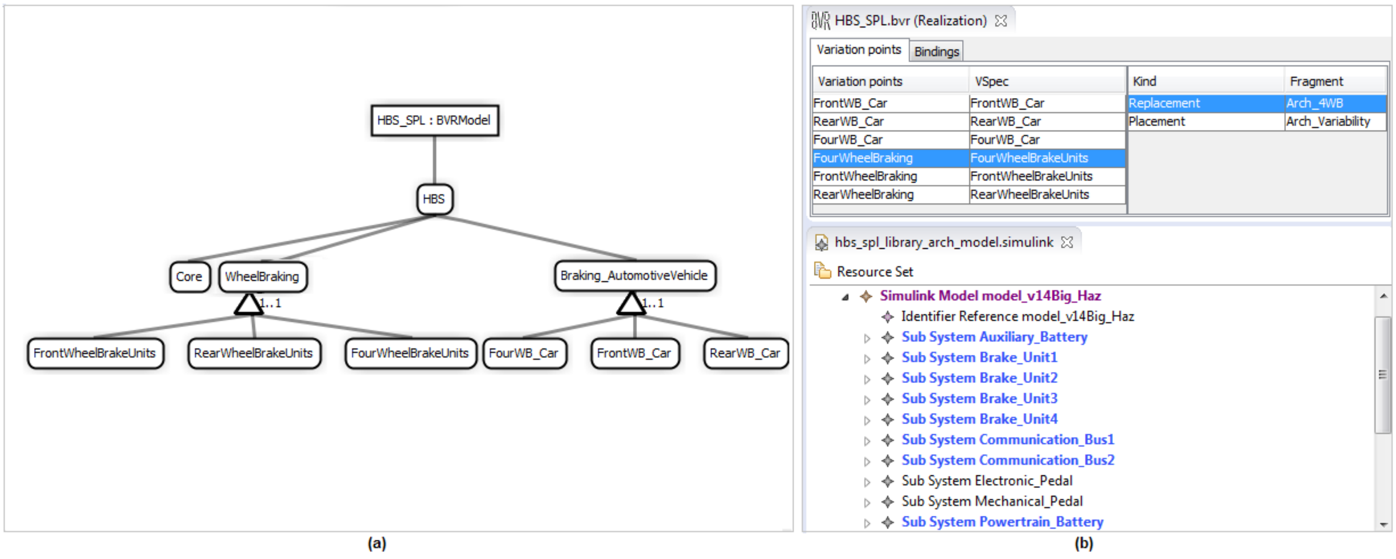
Component	**Impl/Current**	**Output Deviation**	**Failure Expression**
BrakeUnit3.IWMPowerConverter	RWB/No	Omission-Out1	Omission-In1 or OFailure1 or Omission-In2
		Omission-Out2	Omission-In1 or OFailure1
		Value-Out1	Value-In1 or Value-In2 or VFailure1
		Value-Out2	Value-In1 or VFailure1
BrakeUnit1.IWM	FWB/No	Omission-Out1	OFailure1 or Omission-In1 or Omission-In2

4.4.4 SC-PLE Phase 4: HBS-SPL Variability Realization Modeling

BVR variability management tool (VASILEVISKIY *et al.* 2015) and its adapters for MATLAB/Simulink and HiP-HOPS were used to support the specification of the product line variability realization model. BVR toolset provides a set of tools and editors to support standard variability modeling activities: feature modeling, variability realization modeling, and product derivation. A full description of how to configure the variability realization model in the BVR toolset for a given product line can be found in VASILEVISKYI *et al.* (2015) and HAUGEN and OGRAD (2014). In this phase, composition rules describing mapping links between HBS-SPL functional and usage context features and their realization in MATLAB/Simulink architecture and HiP-HOPS failure models were defined. These mappings were defined from the analysis of the HBS-SPL feature, architecture and failure models. From the analysis of the HBS-SPL feature and context models, the following variation points have been identified: "*Wheel Braking*", a functional variation point that specifies different ways that wheel brake units can be combined to derive a product variant; "*Braking*", which defines the product variants that impact in the safety properties, and "*Automotive Vehicle*" variation point, which encapsulates variation on the targeted automotive vehicle where "*Braking*" variants can be deployed. The combination of "*Braking*" and "*Automotive Vehicle*" variation points directly impacts in the definition of the product failure model. For this case study, mapping links between feature and their realization in architectural and safety analysis models were defined by considering: "*4WB*", "*FWB*", and "*RWB*" product variants deployed in a car vehicle.

The specification of the HBS-SPL variability realization model using BVR toolset was performed in two steps. Firstly, the variability specification model was defined based on the product line feature and context models. Finally, the variability realization model was defined by linking variants to placement and replacement fragments referencing architectural and failure model elements, via fragment substitutions. Fragment substitutions define the architectural and failure model elements to be included and excluded when the underlying variant is considered to be bound during product derivation.

Figure 4.15a shows the HBS-SPL variability specification model designed with the support of BVR toolset. Core features represent "*Auxiliary Battery*", "*ComBus1*", "*ComBus2*", "*ElectronicPedal*", "*MechanicalPedal*", and "*PowertrainBattery*" mandatory features shown Figure 4.11. "*FourWheelBrakeUnits*", "*FrontWheelBrakeUnits*", and "*RearWheelBrakeUnits*" represent wheel braking architectural variants, i.e., combinations between wheel brake unit alternative features shown Figure 4.11. "*4WB_Car*", "*FWB_Car*", and "*RWB_Car*" shown in Figure 4.12, respectively represent "*FourWB*", "*FrontWB*", and "*RearWB*" variants deployed in a car vehicle. These variants impact in the resolution of variability in the product line failure model. A fragment substitution was created for each variant defined in the BVR variability specification model (Figure 4.15a), as illustrated in Figure 4.15b. Each fragment substitution contains one placement and one replacement fragment. Placements define model elements to be removed when variability is resolved for a given product variant. Placement elements represent variability mechanisms in the model. Replacements are used to define model elements to be included when variability is resolved for a given product variant (HAUGEN and OGRAD, 2014). Figure 4.15b shows the "*Arch_4WB*" replacement fragment associated with "*FourWheelBrakeUnits*" architectural variant. This fragment indicates that "*Brake_Unit1*", "*Brake_Unit2*", "*Brake_Unit3*", and "*Brake_Unit4*" wheel braking subsystems are included in the product architecture model when variability is resolved for "*FourWheelBrakeUnits*" variant. Therefore, mapping links between functional and usage context features and their realization in the architecture model are obtained in the variability realization model.



HBS_SPL.bvr (Realization)

Variation points | Bindings

Variation points	VSpec
FrontWB_Car	FrontWB_Car
RearWB_Car	RearWB_Car
FourWB_Car	FourWB_Car
FourWheelBraking	FourWheelBrakeUnits
FrontWheelBraking	FrontWheelBrakeUnits
RearWheelBraking	RearWheelBrakeUnits

Kind	Fragment
Replacement	Arch_4WB
Placement	Arch_Variability

hbs_spl_library_arch_model.simulink

Resource Set

- Simulink Model model_v14Big_Haz
  - Identifier Reference model_v14Big_Haz
  - Sub System Auxiliary_Battery
  - Sub System Brake_Unit1
  - Sub System Brake_Unit2
  - Sub System Brake_Unit3
  - Sub System Brake_Unit4
  - Sub System Communication_Bus1
  - Sub System Communication_Bus2
  - Sub System Electronic_Pedal
  - Sub System Mechanical_Pedal
  - Sub System Powertrain_Battery

Figure 4.15. HBS-SPL variability specification and realization models in BVR toolset.

In the HBS-SPL variability modeling, firstly, a replacement fragment is defined for linking "*Core*" feature to the auxiliary battery, communication buses, electronic pedal, mechanical pedal, and powertrain battery MATLAB/Simulink architectural components (see Figure 4.12), as shown in Table 4.4. Fragment substitutions for linking *Wheel Braking* variants to architectural model elements were also specified. For each variant, fragment substitutions composed by placement and replacement fragments were defined. These fragment substitutions define how variability is resolved when "*Wheel Braking*" variants are chosen in the resolution model. Table 4.4 shows placement and replacement fragments associated with other wheel braking variants. "*FrontWheelBraking*" replacement fragment defines that "*Brake Unit1*" and "*Brake_Unit2*" subsystems and their connections with battery components should be included in the resolved architecture model when "*FrontWheelBrakeUnits*" variant is chosen. On the other hand, "*FrontWheelBraking*" placement fragment determines that "*Brake Unit3*" and "*Brake Unit4*" subsystems and their connections with battery components are variability mechanisms that should be removed from the architecture model when variability is resolved for this variant.

Table 4.4 – Wheel braking variants and associated fragments.

Variation Point	Variant	Fragment	Architectural Model Elements
Core	*Core*	Replacement	Auxiliary Battery, Communication Bus1, Communication Bus2, Electronic Pedal, Mechanical Pedal model blocks and their connections.
Wheel Braking	*FrontWheelBraking*	Placement	Brake_Unit1, Brake_Unit2, Brake Unit3, and Brake Unit4 subsystems and their connections.
		Replacement	Brake Unit1 and Brake_Unit2 subsystems and their connections with Auxiliary and Powertrain batteries.
	RearWheelBraking	Placement	Brake Unit1 and Brake Unit2, Brake Unit3, and Brake Unit subsystems and their connections.
		Replacement	Brake Unit3 and Brake_Unit4 subsystems and their connections with Auxiliary and Powertrain batteries.
	FourWheelBraking	Placement	Brake_Unit1, Brake_Unit2, Brake Unit3 and Brake_Unit4 subsystems and their connections.
		Replacement	Brake_Unit1, Brake_Unit2, Brake Unit3, and Brake Unit4 subsystems and their connections.

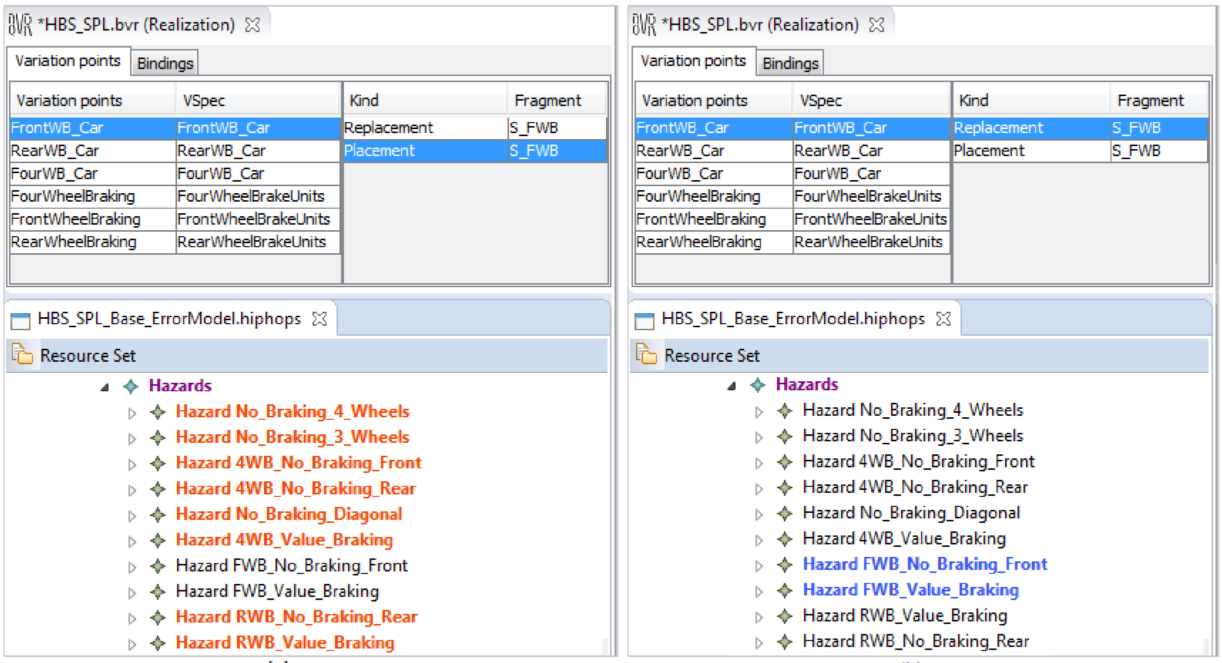
After specifying fragment substitutions for HBS-SPL architectural components, fragment substitutions were specified to define how variability in the product line failure model is resolved when *"4WB_Car"*, *"FWB_Car"*, and *"RWB_Car" safety-related* variants, i.e., variants in the product line failure model, are chosen in the resolution model. Table 4.5 shows an excerpt of the mappings linking safety variants to their realization in the product line safety analysis (failure model). When *"4WB_Car"* safety variant is chosen, hazards and component failure data associated with *"FWB_Car"* and *"RWB_Car"* variants, specified in a placement fragment, are removed from the failure model, and variant-specific safety information, specified in the replacement fragment, are included in the resolved model.

Table 4.5 – Braking/Automotive Vehicle variants and their associated fragments.

Variation Point	**Variant**	**Fragment**	**Failure Model Elements**	
			Hazard and Risk Analysis Data	**Failure Data**
Braking/ Automotive Vehicle	*4WB_Car*	Placement	No_braking_four_wheels, No_braking_three_wheels, No_braking_diagonal, 4WB_No_braking_front, 4WB_No_braking_rear, 4WB_Value_braking, RWB_No_braking_rear, RWB_Value_braking	RW-BU3WNCImpl, RW-U4WNCImpl, …
		Replacement	No_braking_four_wheels, No_braking_three_wheels, No_braking_diagonal, 4WB_No_braking_front, 4WB_No_braking_rear, 4WB_Value_braking	BU1WNCImpl, BU2WNCImpl, BU3WNCImpl, BU4WNCImpl, …
	FWB_Car	Placement	No_braking_four_wheels, No_braking_three_wheels, No_braking_diagonal, 4WB_No_braking_front, 4WB_No_braking_rear, 4WB_Value_braking, RWB_No_braking_rear, RWB_Value_braking	RW-BU3WNCImpl, RW-BU4WNCImpl, BU3WNCImpl, BU4WNCImpl, …
		Replacement	FWB_No_braking_front, FWB_Value_braking	BU1WNCImpl, BU2WNCImpl, …

Figure 4.16 illustrates the relationships between HiP-HOPS failure model elements and *"FWB_Car"* safety-related variant in BVR and HiP-HOPS model editors. When placement and replacement fragments are selected in the variability realization model, the corresponding HiP-HOPS model elements are highlighted in the HiP-HOPS model EMF-based editor. Placement fragments are highlighted in red, and replacements in blue.

Figure 4.16a shows that when “*S_FWB*” placement fragment is selected, failure model elements representing variability mechanisms are highlighted in red. Figure 4.16b also illustrates that when “*S_FWB*” replacement fragment is selected, failure model elements to be included in the product failure model when variability is resolved for the “*FrontWB*” variant are highlighted in blue. The graphical user interface provided by BRV toolset, and the developed adapter plugins reduced the complexity to defining mappings linking features to their realization in architectural components and safety analysis assets in the variability realization model. The delivered HBS-SPL variability realization model comprises 6 fragment substitutions, 2 placements and 6 replacement fragments. 3 fragments substitutions are associated with architectural model elements, and 3 fragment substitutions are associated with failure model elements.


(a)

*HBS_SPL.bvr (Realization)

Variation points | Bindings

Variation points	VSpec
FrontWB_Car	FrontWB_Car
RearWB_Car	RearWB_Car
FourWB_Car	FourWB_Car
FourWheelBraking	FourWheelBrakeUnits
FrontWheelBraking	FrontWheelBrakeUnits
RearWheelBraking	RearWheelBrakeUnits

Kind	Fragment
Replacement	S_FWB
Placement	S_FWB

HBS_SPL_Base_ErrorModel.hiphops

Resource Set

- Hazards
  - Hazard No_Braking_4_Wheels
  - Hazard No_Braking_3_Wheels
  - Hazard 4WB_No_Braking_Front
  - Hazard 4WB_No_Braking_Rear
  - Hazard No_Braking_Diagonal
  - Hazard 4WB_Value_Braking
  - Hazard FWB_No_Braking_Front
  - Hazard FWB_Value_Braking
  - Hazard RWB_No_Braking_Rear
  - Hazard RWB_Value_Braking

(b)

*HBS_SPL.bvr (Realization)

Variation points | Bindings

Variation points	VSpec
FrontWB_Car	FrontWB_Car
RearWB_Car	RearWB_Car
FourWB_Car	FourWB_Car
FourWheelBraking	FourWheelBrakeUnits
FrontWheelBraking	FrontWheelBrakeUnits
RearWheelBraking	RearWheelBrakeUnits

Kind	Fragment
Replacement	S_FWB
Placement	S_FWB

HBS_SPL_Base_ErrorModel.hiphops

Resource Set

- Hazards
  - Hazard No_Braking_4_Wheels
  - Hazard No_Braking_3_Wheels
  - Hazard 4WB_No_Braking_Front
  - Hazard 4WB_No_Braking_Rear
  - Hazard No_Braking_Diagonal
  - Hazard 4WB_Value_Braking
  - Hazard FWB_No_Braking_Front
  - Hazard FWB_Value_Braking
  - Hazard RWB_Value_Braking
  - Hazard RWB_No_Braking_Rear

Figure 4.16. HBS-SPL variability realization and failure models in BVR tool.

4.4.5 SC-PLE Phases 5 and 6: Resolution Modeling and Product Derivation Phases

In product line application engineering, the resolution model is defined, with the support of the BVR resolution model editor, according to variant-specific requirements. Thus, functional and safety-related variants defined in the product line feature model are chosen. For this case study, resolution models were defined for: HBS-4WB, HBS-FWB, and HBS-RWB variants show in Figure 4.17. These variants differ from each other in the number of architectural and failure model elements, and in the way as they are combined when variability in architecture and failure models is resolved. HBS-4WB variant contains four wheel brake units deployed in a car vehicle, and variant-specific failure data denoted by "*FourWB*" usage context feature in Figure 4.17a. HBS-FWB variant comprises front wheel brake units and "*FrontWB*" variant-specific failure data (Figure 4.17b). Finally, HBS-RWB comprises rear wheel brake units and "*RearWB*" variant-specific failure data (Figure 4.17c).

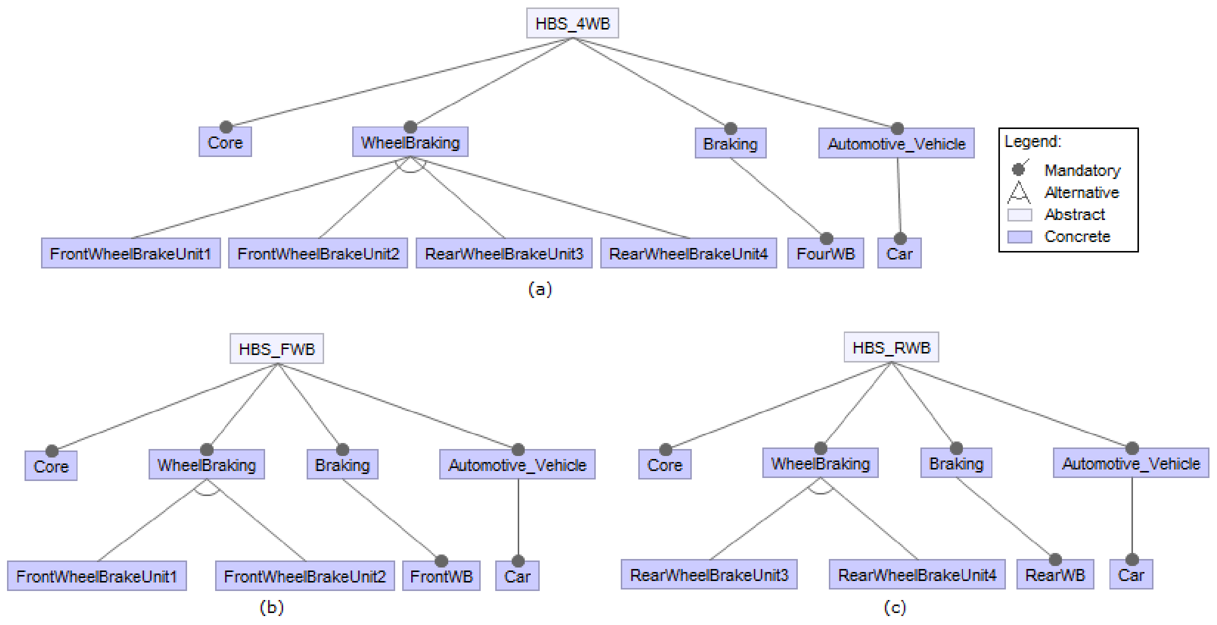


Figure 4.17. HBS-SPL resolution models.

For each HBS-SPL system variant, the following models were input to the BVR toolset resolving variability in product line architecture and failure models during "*SC-PLE-6: Product Derivation*" phase: variability specification, resolution, and realization models, and HBS-SPL MATLAB/Simulink architecture and HiP-HOPS failure models. Finally, for each variant, BVR was executed to generate variant-specific MATLAB/Simulink architecture and failure models. The HBS-4WB architecture model contains both mechanical pedal, and electronic pedal that sends the braking outputs to the communications buses. *Communication* buses send braking commands to "*Brake_Unit1*", "*Brake_Unit2*", "*Brake_Unit3*", and "*Brake_Unit4*" components (see Figure 4.13).

The HBS-FWB system variant differs from HBS-4WB by the absence of rear wheel braking units. The HBS-RWB variant differs from HBS-4WB by the absence of front wheel brake units. Hazards, their causes, and component failure logic are different in each one of these variants. For each HBS-SPL system variant, hazard and risk analysis, and component failure data is stored in the failure model together with Simulink architecture model. Therefore, the systematic reuse of both architectural and safety analysis is achieved, and product safety assessment can be performed, with the support of compositional safety analysis tools, from a reused architecture model enhanced with failure data. The variability management approach has also been further validated in the Tiriba Flight Control aerospace product line (BRANCO *et al.* 2011) case study presented in Chapter 8 and detailed in Appendix D.

4.5 Discussion and Limitations

The variability management approach presented in this chapter provides systematic steps to support variability modeling and management in architecture and safety analysis models. Such approach supports the traceability of context and design variation throughout architectural and safety assets across product line development and safety processes. Therefore, variability in functional and usage context features is traced to architectural design. Whereas safety is context-dependent, usage context and architectural variation are further traced to safety analysis assets, i.e., functional hazard assessment and component failure analysis.

Achieving the systematic reuse of safety analysis data in safety-critical product line engineering requires extending existing variant management tools to support the specification of mappings linking product line features to their realization in safety analysis in the variability model. This chapter has presented a method to support the adaptation of variant management tools to support variability management in safety analysis models. Thus, we have extended the BVR variant management tool, by creating adapters to support variability management in architecture and failure models specified with the support of MATLAB/Simulink and HiP-HOPS, and OSATE AADL/Error Annex model-based development and compositional safety analysis tools. We also have adapted Hephaestus/Simulink to support variability management in HiP-HOPS failure models. The complexity of performing such adaptations is dependent upon the characteristics of the targeted variant management tool and the available documentation.

In addition, the systematic reuse of safety analysis in safety-critical product line engineering requires the delimitation of the scope of variability in the safety model, and the definition of mapping links between features and their realization in the safety model into the variability model. The success of applying such approach is dependent upon the domain knowledge of product line engineers and safety analysts in scoping safety variability and establishing mapping links between features and their realization in the failure model. Since architectural variants directly impacts in safety properties, mappings between features and failure model elements should be performed aware of architectural and contextual variants. The complexity of the specification of placement and replacement fragments on the variability model may increase as the number of architectural and context variants increases. More conflicts in the specification of placement and replacement fragments may arise and their resolution could become a time consuming task. The variability realization model directly impacts in the variability resolution process, whose errors could lead to the derivation of wrong variant-specific architecture and failure models. Although variability realization modeling is an error-prone task, graphical editors, as provided by BVR toolset, contribute to minimize the errors in this phase.

Adapting a variant management tool is limited to whether they provide adapter interfaces that allow extending them to support variability management in the safety model. In addition, it is necessary to be aware of how variability is managed and resolved in the targeted tool. When adapter interfaces are provided, e.g., which is the case of BVR, CVL, Hephaestus/Simulink, and pure::variants, and variability modeling is orthogonal, i.e., product line variability is specified in a dedicated model, the adaptation consists in extending this interface to create an adapter plugin to support the management and resolution of variability in safety analysis models. When adapter interfaces are not provided, the analysis of the available documentation is required to identify how variability management and product derivation processes were implemented in the tool, to be able to further extend the target tool. The guidance to extend variant management tools presented section 4.2 can be used to support the identification of basic elements of a variant management tool.

4.6 Summary

This chapter presented a variability management approach and tool support for the systematic reuse architecture and safety models in safety-critical product line engineering. The approach is applicable to different variant management and compositional safety analysis techniques than used in this thesis. A method to support the adaptation of existing variant management tools to support variability management in safety models has also been proposed. This method was used to adapt BVR and Hephaestus/Simulink variant management tools to support variability management in safety analysis models developed with the support of HiP-HOPS and OSATE AADL/Error Annex. The proposed approach and tooling provides seamless integration between variant management and compositional safety analysis techniques.

BVR toolset and the developed adapters for MATLAB/Simulink and HiP-HOPS were used to support the management and resolution of variability in architectural and safety analysis models from the Hybrid Braking System automotive product line. Thus, context and design variation were linked to their realization in MATLAB/Simulink architecture and HiP-HOPS failure models, and then, multiple variant-specific Simulink and failure models were automatically generated during the product derivation. The reuse of safety analysis data is achieved early on safety assessment process, contributing to reduce the effort of safety analysts in performing safety analysis for a specific product variant, as illustrated in automotive and aerospace product line compositional safety analysis case studies presented in Chapters 5 and 8. With the support of existing compositional safety analysis techniques and tools, e.g., HiP-HOPS and OSATE AADL, fault trees and FMEAs results can be generated from the reused variant-specific architecture and failure models. The following chapter presents the product line compositional safety analysis and design optimization approach.

Chapter 5

Product Line Compositional Safety Analysis and Design Optimization

5.1 Introduction

In safety-critical software product line engineering, variation in design and context may impact in safety properties identified during hazard and risk analysis, and component failure analysis. Therefore, safety analysis in safety-critical product lines should be performed aware of context and design variation earlier in domain engineering in order to support the systematic reuse of architectural components in a range of different variants. In a safety-critical product line, safety properties may change according to the selection of product variants. Different variants and context lead to different hazards, associated risks to the overall system safety, allocated safety requirements, and contributing component failures. Variation in safety analysis directly impact in fault trees, FMEA, and safety integrity requirements decomposition assessment artefacts that are built upon safety analysis artefacts.

With the advance of research in the field of system safety engineering, the benefits related to traceability and automation provided by model-based safety assessment techniques have been recognized by both industry and safety standards (CMU, 2013; LINDEN *et al.* 2007). With evolution of model-based development techniques, system safety analysis can be performed integrated with system design with the support of compositional safety analysis techniques. Compositional safety analysis integrates system and failure modeling in a single model, and automates part of the system safety analysis with the provision of automatic synthesis of FTA and FMEA from a system model enhanced with failure behavior information. Thus, compositional safety analysis techniques contribute to reduce the complexity and supporting the efficient and consistent evolution of design and failure models, reducing the costs and improving the quality of the system safety analysis (DOMIS and TRAPP, 2008; LISAGOR *et al.* 2006; JOSHI *et al.* 2005).

Compositional safety analysis is effective in gathering safety and reliability information of the system, which can be further used to support safety analysts in taking architectural decisions. However, since the evaluation of multiple design choices against optimization objectives is time consuming, design optimization techniques can be used to automate the analysis of candidate solutions against a range of optimization objectives such as reliability and cost. In safety-critical systems development, design optimization can be used to support the automatic decomposition of safety integrity requirements throughout contributing component failures in order to achieve compliance with safety standards without being stringent or expensive. In this thesis we have used design optimization techniques to support the automatic decomposition of safety integrity requirements in safety-critical product lines and their instances (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014; BIEBER *et al.* 2011). To take the benefits of integrating compositional safety analysis and design optimization into product line processes it is necessary to consider to impact of context and design variation in performing safety analysis. This is necessary to enable the systematic reuse of safety assets in domain engineering, and generation of assessment assets in application engineering.

This chapter presents a model-based approach to support the systematic integration of compositional safety analysis and design optimization techniques into safety-critical software product line engineering processes. The approach has been instantiated with the support of HiP-HOPS and OSATE AADL/Error Annex compositional safety analysis techniques, HiP-HOPS design optimization techniques, and extensions of variant management tools to support variability management in safety analysis models (detailed in Chapter 4). The proposed product line compositional safety analysis and design optimization approach has been built in compliance with system safety assessment processes defined in ARP 4754A and ISO 26262 safety standards. In domain engineering, the approach provides support to perform safety analysis aware of variation in design and context to enable the systematic reuse of safety assets. In application engineering, the approach supports the automatic generation of safety artefacts, e.g., fault trees and FMEA, and safety integrity requirements decomposition, using compositional safety analysis and design optimization techniques. In this thesis, we have used the BVR toolset to support variant management, HiP-HOPS and OSATE AADL/Error Annex to support compositional safety analysis, and HiP-HOPS Tabu Search algorithms

(AZEVEDO *et al.* 2013) to support the product line design optimization. The integration of variant management, compositional safety analysis and design optimization techniques provide automated support for the proposed product line compositional safety analysis approach.

The approach presented in this chapter differs from existing work on product line safety analysis (SCHULZE *et al.* 2013; GOMEZ *et al.* 2010; HABLI, 2009; LIU *et al.* 2007; DEHLINGER and LUTZ, 2006; DEHLINGER and LUTZ, 2004; FENG and LUTZ, 2005; DINGDING and LUTZ, 2002) by making an explicit distinction between reusable safety artefacts, in which variability should be managed, and those that should be generated from the reused safety artefacts. Such distinction contributes to reduce the complexity of variability management in system safety engineering artefacts, and enables the automated traceability of context and design variation throughout safety artefacts produced in domain engineering and application engineering processes, e.g., safety analysis, fault trees and FMEA results. In domain engineering, hazard and component failure analysis are considered reusable safety assets for which variability should be managed. Fault trees, FMEA, safety integrity requirements decomposition, and assurance case artefacts, which are dependent upon the safety analysis, should be generated in application engineering. The approach defines a process to perform product line safety analysis in domain engineering to support the systematic reuse and generation of safety artefacts in application engineering. This chapter presents a systematic approach to integrate compositional safety analysis and design optimization into safety-critical software product line engineering, and a method and tool built as an extension to HiP-HOPS design optimization technique to support automatic analysis and allocation of safety integrity requirements to product line components.

Section 5.2 presents the phases, activities, tasks, and their input and output artefacts of the proposed approach to support: context aware and reuse-driven safety analysis in domain engineering, and automatic generation of safety assessment artefacts, with the support of compositional safety analysis and design optimization techniques, in application engineering. Section 5.3 presents the proposed method and tool to support the automatic allocation of safety integrity requirements to safety-critical product line components, addressing automotive and aerospace domains. Section 5.4 presents the evaluation of the proposed compositional safety analysis approach and product line SIL decomposition tooling based on a hybrid braking system automotive case study. Section 5.5 discusses the results and limitations. Section 5.6 presents a summary of this chapter.

5.2 Product Line Compositional Safety Analysis and Design Optimization Approach

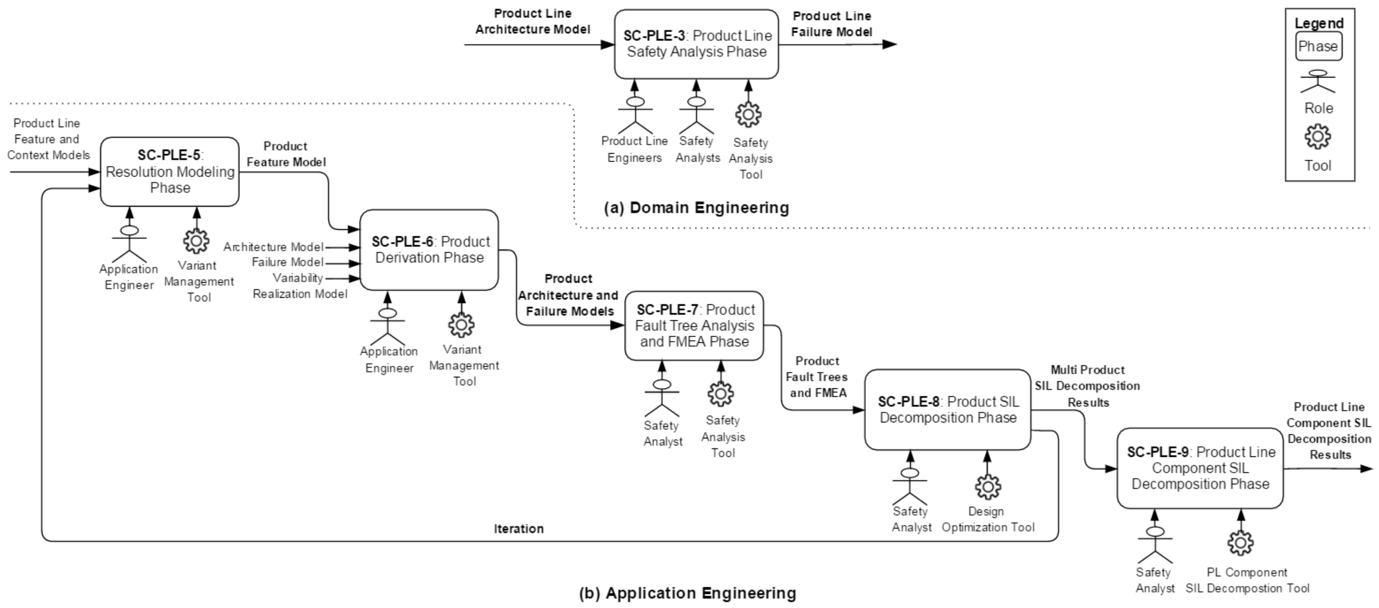
In safety-critical software product line engineering, safety analysis should be considered from earlier stages in the domain engineering to support the systematic reuse of safety-critical components. The integration of model-based development and compositional safety analysis into software product line engineering allows safety analysis to be performed in parallel to the system design. Thus, compositional safety analysis allows safety analysts to specify the system failure behavior incrementally as the design progresses from the system architecture to detailed component level. Compositional safety analysis reduces the complexity of safety analysis by adopting a “divide-and-conquer” approach, by breaking down the analysis into the characterization of the failure behavior of individual components. Therefore, the component failure behavior relates to design components in a clear fashion, enabling traceability and systematic reuse of both components and associated failure behavior. However, since safety is context dependent, to achieve the systematic reuse of design and component failure logic[18], product line compositional safety analysis should be performed aware of the impact context and design variation in system safety properties as proposed in this thesis. Such analysis supports the systematic reuse of product line design and safety analysis, and automatic synthesis of fault trees and FMEA in application engineering. In addition, the integration of design optimization into product line processes supports safety analysts in achieving cost effective process-based certification of both product line and their instances. In domain engineering, design optimization supports the automatic allocation of safety integrity requirements to product line components. In application engineering, design optimization supports the automatic decomposition of safety integrity requirements throughout component failure modes of a particular system variant.

Figure 5.1 illustrates the phases of the proposed for product line component safety analysis and design optimization approach, which are divided in domain engineering and application engineering. The approach was defined in compliance with safety assessment processes defined in ARP 4754A and ISO 26262 safety standards, which were adapted to address safety-critical product line engineering (OLIVEIRA *et al.* 2014). For example, the

[18]Component failure logic: output and input deviations, internal failures of a component, and their causal relationships (LISAGOR *et al.* 2006).

"*SC-PLE-3: Product Line Safety Analysis*" phase in domain engineering corresponds to Functional Hazard Assessment (FHA) and Preliminary System Safety Assessment (PSSA) processes defined in ARP 4754A/ARP 4761A. In addition, the "*SC-PLE-3*" phase is equivalent to Hazard and Risk Analysis tasks defined in ISO 26262 safety lifecycle. In this thesis, HiP-HOPS and OSATE AADL/Error Annex compositional safety analysis techniques were considered to provide automated support for safety analysis in product line domain engineering and application engineering.

The approach starts from a preliminary product line architecture model, which comprises core, optional and alternative systems, subsystems and components. Architecture models can be specified using data-flow oriented languages supported by tools such as MATLAB/Simulink, SimulationX, SCADE, or using an architectural description language such as AADL (SAE, 2012). During safety analysis, the architecture model is annotated with information about potential hazards arising in the class of systems potentially derived from the product line and its context, and potential failure conditions in architectural components leading the these hazards. Up to this point, the SPL architecture model contains substantial safety-related information that can be reused for variant-specific safety analysis.



Product Line Architecture Model
SC-PLE-3: Product Line Safety Analysis Phase
Product Line Failure Model
Product Line Engineers
Safety Analysts
Safety Analysis Tool
(a) Domain Engineering
Legend
Phase
Role
Tool
Product Line Feature and Context Models
SC-PLE-5: Resolution Modeling Phase
Product Feature Model
Application Engineer
Variant Management Tool
Architecture Model
Failure Model
Variability Realization Model
SC-PLE-6: Product Derivation Phase
Application Engineer
Variant Management Tool
Product Architecture and Failure Models
SC-PLE-7: Product Fault Tree Analysis and FMEA Phase
Safety Analyst
Safety Analysis Tool
Product Fault Trees and FMEA
SC-PLE-8: Product SIL Decomposition Phase
Safety Analyst
Design Optimization Tool
Multi Product SIL Decomposition Results
SC-PLE-9: Product Line Component SIL Decomposition Phase
Product Line Component SIL Decomposition Results
Safety Analyst
PL Component SIL Decompostion Tool
Iteration
(b) Application Engineering

Figure 5.1. Product line compositional safety analysis and design optimization approach.

In domain engineering, once the product line architecture model has been annotated with hazards and component failure logic, with the support of compositional safety analysis techniques, variability in architecture and safety analysis models is managed using variant management techniques as detailed in Chapter 4. Thus, mapping links between context and design variation and their realization into architecture and safety analysis are specified in the variability model. In application engineering, a product variant ("*SC-PLE-5*") is defined by selecting product line features, and then, derived ("*SC-PLE-6"*), with the support of a variant management tool. In the following, if specific functions/features are added to a variant-specific architecture model, variant-specific safety analysis should be performed, with the support of compositional safety analysis, in the same way as safety analysis in domain engineering ("SC-PLE-3"). This is required to assess the impact of the added functions on the overall safety of the system architecture by identifying variant-specific hazards and component failure modes. Such information provides feedback to the product line safety analysis in domain engineering, thus variant-specific functions and their safety analysis can be added to the product line asset base for further reuse. Therefore, the product line feature and context, architecture, safety analysis, and variability models should be updated.

Further on the analysis, compositional safety analysis is used to perform the automatic synthesis of fault trees and FMEA results for a given product variant from the reused architecture and failure models ("SC-PLE-7" in Figure 5.1). Fault trees are generated for each variant-specific hazard. A fault tree shows how component failures can be logically combined and propagated throughout the architecture leading to the occurrence of a hazard. Later, variant-specific fault trees are combined to create a FMEA showing the contributions of each component and associated failure modes to the occurrence of system hazards. Fault trees and FMEA for an individual variant allows safety analysts to identify whether product requirements are met. In addition, fault tree analysis and FMEA for multiple product variants allow safety analysts to draw conclusions about the safety of the product line architecture. For example, FMEAs can show the component failure modes that contribute to severe system hazards across the product line architecture. The respective components creating these failure modes must be developed to the appropriate level of integrity. The analysis of fault trees and FMEA results for multiple product variants support safety analysts to take decisions to ensure that architectural components meet the safety requirements across different usage scenarios covered by the analysis.

The generated variant-specific fault trees and FMEA results are input artefacts for design optimization performing the automatic decomposition of safety integrity requirements allocated to variant-specific hazards, in terms of SILs, throughout the contributing component failure modes ("SC-PLE-8"). Design optimization provides near optimal SIL allocations allowing safety analysts to achieve process-based certification for a given system variant without being stringent or expensive, since stringent SILs demands more effort in terms of verification, validation, and testing required to address certification requirements. Design optimization techniques are detailed in Section 2.2.4.2, and SIL decomposition process in Section 2.2. In this thesis we have used HiP-HOPS Tabu Search design optimization algorithms to support the automatic decomposition of automotive SILs and aerospace Development Assurance Levels (DALs) (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014). After generating variant-specific fault trees, FMEA, and SIL decomposition results another variant can be derived ("SC-PLE-5" and SC-PLE-6"), and further analyzed by performing FTA, FMEA, and SIL decomposition.

Whereas variation in design and context directly impact the SIL decomposition results, each product variant might have different SIL decompositions. It implies that establishing safety integrity requirements for product line components require the analysis of the SILs allocated to those components in a range of different product variants (scenarios). If different SILs are allocated to a component in different variants, then the highest SIL must be met for that component being safely used across the analyzed variants. This type of allocation would allow developers to meet their responsibilities in order to ensure the safety of the product line architecture, and achieving compliance with safety standards, without incurring unnecessary costs of complete reanalysis and reallocation of SILs as traditionally demanded for each variant. In this thesis, a method and tool have been development to support the analysis and allocation of SILs to product line components (OLIVEIRA *et al.* 2015a).

The tool named PL-SILDec (*Product Line SIL Decomposition*) was developed as an extension of HiP-HOPS Tabu Search ASIL/DAL (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014) optimization tools to address the product line design. The input for the tool is a set of variant-specific SIL decomposition results provided by HiP-HOPS SIL decomposition tool. PL-SILDec outputs the SILs to be allocated to product line components addressing safety across the SPL. Therefore, when the required number of product variants was already analyzed, the product line component SIL decomposition ("*SC-PLE-9*") is performed, with the support of PL-SILDec tool, to support product line process-based certification. The following subsections detail the product line compositional safety analysis phases, activities, tasks, and their input and output artefacts.

5.2.1 SC-PLE-3: Product Line Safety Analysis

The "*Product Line Safety Analysis*" phase examines the product line architecture from the perspective of different product variants and their context to identify potential hazards, failure conditions, and contributing component failures, associated risk classification and allocated safety requirements that can arise in different product variants and context. Product line safety analysis differs from traditional safety analysis for a single system by considering the impact of context and design variation in safety properties. Therefore, safety analysis is performed by considering a range of product variants and their context. Combinations between functional and usage context features defined in the feature model provide a set of scenarios that could be considered in performing product line compositional safety analysis. The product line compositional safety analysis phase ("*SC-PLE-3*") comprises the following activities: identification of usage scenarios ("*SC-PLE-3.1*"), functional hazard assessment (*"SC-PLE-3.3"*), and component failure analysis ("*SC-PLE-3.4*") illustrated in Figure 5.2.

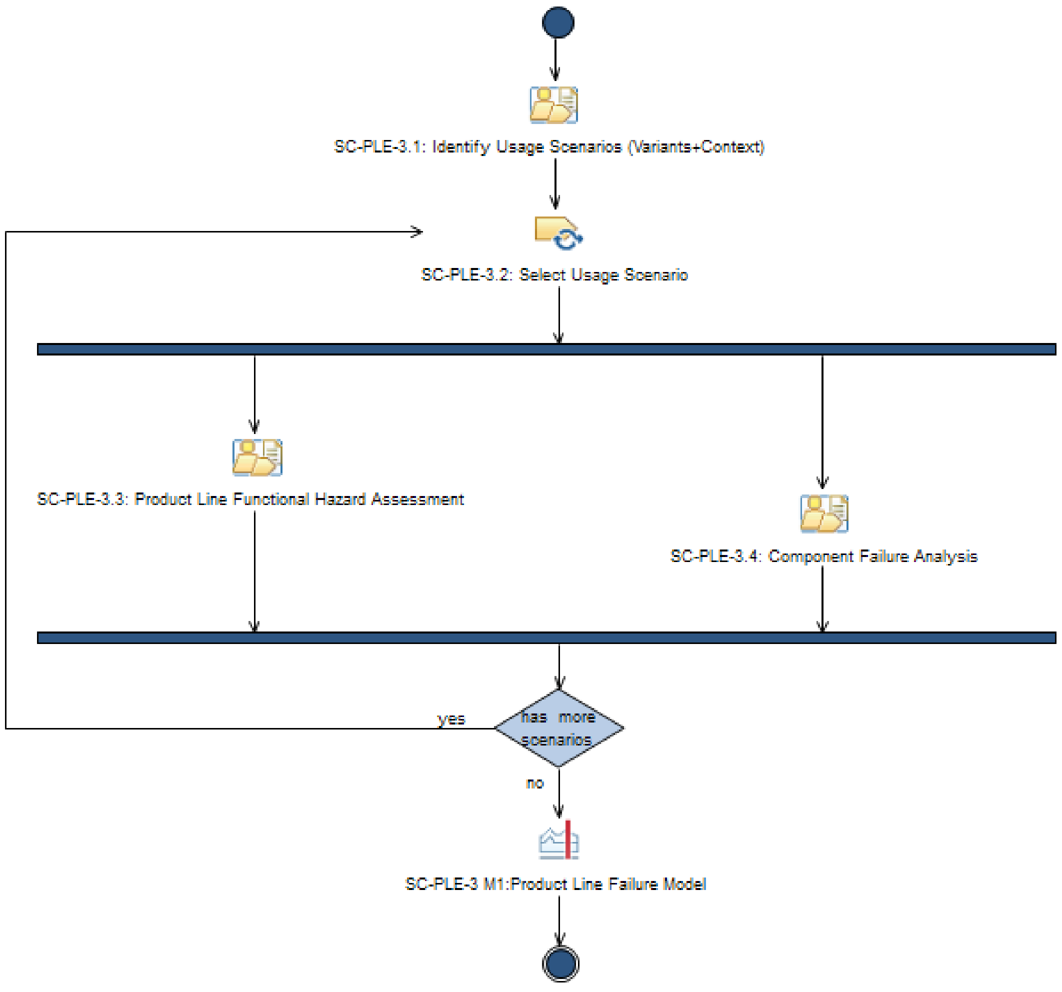

SC-PLE-3.1: Identify Usage Scenarios (Variants+Context)
SC-PLE-3.2: Select Usage Scenario
SC-PLE-3.3: Product Line Functional Hazard Assessment
SC-PLE-3.4: Component Failure Analysis
yes
has more scenarios
no
SC-PLE-3 M1:Product Line Failure Model

Figure 5.2. SC-PLE-3: Product line safety analysis activities and tasks.

5.2.1.1 Identify Usage Scenarios

The "*SC-PLE-3.1: Identify Usage Scenarios*" aims to identify product variants and their respective context (scenarios) to be considered during product line safety analysis. The product line feature, context, and architecture models are input artefacts to identify candidate usage scenarios for product line safety analysis. Feature and context models allow safety analysts capturing design choices and constraints that should be considered when performing product line safety analysis. The product line architecture model shows the interactions between components, and data transformations in terms of handling potential variations between product variants (FENG and LUTZ, 2005). "SC-PLE-3.1" comprises the following tasks: identify functional and contextual variants, and combining functional and contextual variants as illustrated in Figure 5.3. Firstly, safety analysts identify combinations between functional features, which represent system functions and their interactions, to derive functional variants. In the following, for each functional variant, safety analysts identify combinations between context features, which define the characteristics of potential contexts that the given functional variant can operate. Finally, by combining the identified functional variants and their respective contextual variants (functional variant + context variant), different usage scenarios can be derived. For example, by considering the Hybrid Braking System product line feature and context models presented in Chapter 4 (Section 4.4.1), seven braking system variants that could be deployed in a car, truck or military vehicle (context variants) have been identified. Therefore, by combining functional and context variants, 21 usage scenarios can be derived. Scenarios are useful to guide the safety analysis of safety-critical product lines.

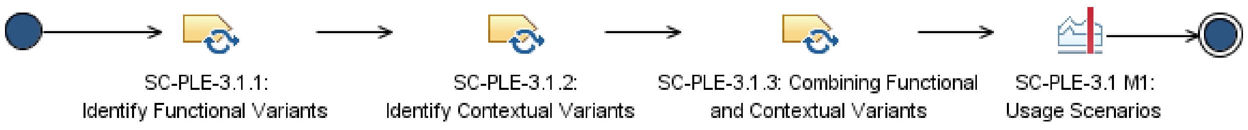

SC-PLE-3.1.1:
Identify Functional Variants
SC-PLE-3.1.2:
Identify Contextual Variants
SC-PLE-3.1.3: Combining Functional
and Contextual Variants
SC-PLE-3.1 M1:
Usage Scenarios

Figure 5.3. SC-PLE-3.1: Identify usage scenarios tasks.

A scenario-based approach for product line compositional safety analysis, as presented in this thesis, allows achieving the balance between the reuse of safety and reliability attributes identified from the analysis of feature interactions that may violate safety properties, and the management of features and their interactions to avoid the feature explosion problem (LIU *et al.* 2007; FENG and LUTZ, 2005). A scenario-based approach for compositional safety analysis allows safety analysts in extracting the required domain knowledge to perform safety analysis for safety-critical product line architectural components, since it would be prohibitive performing such analysis covering all product variants. In this thesis, we have established that the criteria to access the relevant product variants and context (scenarios) to be considered when performing product line safety analysis are dependent upon: safety analyst's domain knowledge, and relevant product variants from the "stakeholders" viewpoint.

Whereas safety-related scenarios involve sequencing of interactions between architectural components that contribute for the realization of product line features, the UML Sequence Diagram is useful to capture the dynamic view of the system execution required for product line safety analysis (FENG and LUTZ, 2005). In addition to feature, context, and architecture models, UML sequence diagrams and Use Case-based analysis techniques (ALLENBY and KELLY, 2001) can also be used to guide safety analysts in identifying potential product variants to be considered in performing product line safety analysis. Feature model combinatorial analysis algorithms (JOHANSEEN *et al.* 2012) used in software product line testing can also be used to support the identification of candidate usage scenarios to be considered during product line safety analysis. In this thesis, the analysis of feature, context, and architecture models has been used to identify relevant usage scenarios to be considered during product line safety analysis. At the end of the "*SC-PLE-3.1*", a list of relevant usage scenarios for product line safety analysis is delivered.

5.2.1.2 Functional Hazard Assessment

After scoping the product line safety analysis to a set of usage scenarios, functional hazard and risk assessment, and component failure analysis are performed with the support of compositional safety analysis techniques. In this thesis, HiP-HOPS and OSATE AADL/Error Annex techniques have been considered to support the product line compositional safety analysis. In the proposed approach, functional hazard and risk assessment, and component failure analysis are performed incrementally and iteratively by considering each usage scenario earlier identified in "*SC-PLE-3.1*". After selecting a usage scenario, functional hazard and risk assessment are performed to identify potential hazards that can arise in the given scenario, the risk they pose to the overall system safety, and allocated safety requirements, as illustrated in Figure 5.4. The usage scenario and the product line architecture model are input artefacts for the "*SC-PLE-3.2: Functional Hazard Assessment*" phase, which comprises four tasks detailed in the following.

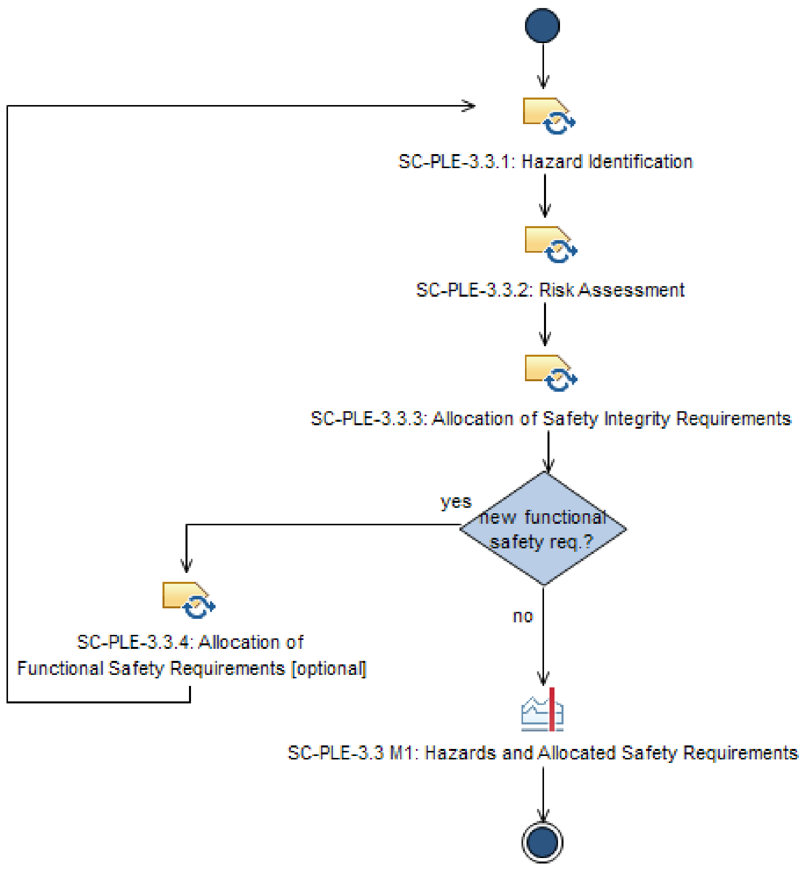

SC-PLE-3.3.1: Hazard Identification
SC-PLE-3.3.2: Risk Assessment
SC-PLE-3.3.3: Allocation of Safety Integrity Requirements
new functional safety req.?
yes
no
SC-PLE-3.3.4: Allocation of Functional Safety Requirements [optional]
SC-PLE-3.3 M1: Hazards and Allocated Safety Requirements

Figure 5.4. SC-PLE-3.3: Functional hazard assessment tasks.

SC-PLE-3.2.1: Hazard Identification: is intended to identify combinations of component failures leading to system-level failures. Hazards are specified by means of logical expressions involving potential failures in product line architectural components. These failures are generally stated in terms of failure types that typically include: *omission*, *commission*, *value*, *early* and *late* failure modes. Firstly, interactions between core architectural components are analyzed in order to identify potential hazards. Later, architectural components representing variation defined in the selected usage scenario are then, analyzed to identify potential hazards that can arise in a specific usage scenario. For each usage scenario, this task outputs a list of identified hazards in the targeted scenario.

SC-PLE-3.2.2: Risk Assessment: this task aims to estimate, based on probabilistic risk tolerability criteria defined in the targeted safety standard, the risk posed by each identified hazard. Different contexts may change the risk posed by a given hazard in different usage scenarios. Thus, risk assessment is performed aware of the impact of context and design variation. At the end of this task, the risk posed by each identified hazard is determined.

SC-PLE-3.2.3: Allocation of Safety Integrity Requirements: is performed from the analysis of the outputs provided by hazard identification and risk assessment. Safety Integrity

Levels are allocated to each identified system hazard according to their risk classification. At the end this task, safety integrity requirements are allocated to each identified hazard.

SC-PLE-3.2.4: Allocation of Functional Safety Requirements: is an optional task intended to identify potential functional safety requirements that can arise to eliminate or minimize the effects of the identified hazards on the overall system safety. A functional safety requirement represents a functional requirement intended to mitigate the effects of system failures. Redundancy is an example of functional safety requirement. If a functional safety requirement is added to the product line architecture, hazard identification, risk assessment, and allocation of safety integrity requirements should be performed to evaluate the impact of the newer added functionality on the overall safety of a particular usage scenario. At the end of the "*SC-PLE-3.2: Functional Hazard Assessment*" phase, hazards and allocated safety requirements are then, delivered.

5.2.1.3 Component Failure Analysis

After the identification of the potential hazards that can arise in a particular usage scenario, safety analysts identify how architectural components can fail and contributing to the occurrence of each identified hazard during component failure analysis ("*SC-PLE-3.4*"). Safety analysts specify the failure behavior associated with each product line architectural component by stating what can go wrong with such component and how it responds to failures elsewhere in the architecture. Such information is called component failure logic. With the support of compositional safety analysis, the component failure behavior is specified by means of annotations in the product line architecture model, which comprises a set of failure expressions showing how deviations in component output ports can be caused either by internal failures in the component or corresponding deviations in the component inputs. Component deviations may include unexpected omission of an output or unintended commission of output, or incorrect output values, or the output being sent too early or late (PAPADOPOULOS *et al.*, 2011). Compositional safety analysis techniques store the component failure logic in a library, so that other components of the same type can reuse it. Whereas the component failure logic may change according to different product line usage scenarios, such variation can be encapsulated into these libraries. Each library contains a failure logic implementation that comprises the failure modes of a component addressing a specific usage scenario.

The component failure analysis is broken down into four tasks as shown in Figure 5.5. The product line architecture model and the identified system hazards for a particular usage scenario are input artefacts to perform the component failure analysis. Firstly, safety analysts select a particular component to be analyzed ("*SC-PLE-3.4.1*"), followed by the identification of potential component output deviations that can contribute to the occurrence of each identified hazard in the given scenario ("*SC-PLE-3.4.2*"). For each identified output deviation, safety analysts identify potential combinations between component internal failures and input deviations that may contribute to the occurrence of each component output deviation ("*SC-PLE-3.4.3*" and "*SC-PLE-3.4.4*"). The analysis continues whilst there are architectural components to be analyzed. At the end, the component failure data for a particular usage scenario is delivered.

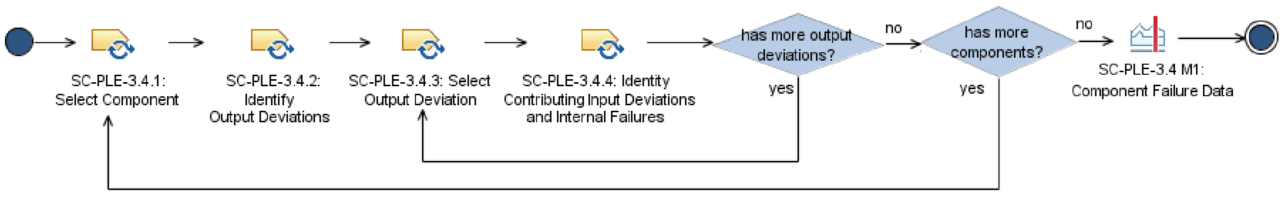

SC-PLE-3.4.1: Select Component
SC-PLE-3.4.2: Identify Output Deviations
SC-PLE-3.4.3: Select Output Deviation
SC-PLE-3.4.4: Identity Contributing Input Deviations and Internal Failures
has more output deviations?
no
yes
has more components?
no
yes
SC-PLE-3.4 M1: Component Failure Data

Figure 5.5. SC-PLE-3.4: Component failure analysis tasks.

Table 5.1 shows an example of component failure analysis for "*C1*" component addressing three different usage scenarios. Each component failure logic implementation contains different output and input deviations, and internal failures, i.e., defined in failure expressions, that contribute to the occurrence of hazards in these scenarios. For example, the causes for "*Omission-Out1*" output deviation are different in "*Scenario1*" and "*Scenario2*". In the example from Table 5.1, the "*Scenario1*" is the current failure logic implementation for component "*C1*". The failure logic inherent to each product line component is stored into the product line failure model together with functional hazard assessment information.

It is important to highlight that the local component failure analysis can reflect either real characteristics or simply the design intention for the analyzed architectural components. In both cases the analysis is useful. For example, at early stages when components are under

Table 5.1 – Variation in component failure logic.

Component	**Impl/Current**	**Output Deviation**	**Failure Expression**
C1	Scenario1/Yes	Omission-Out1	OFailure1 or (Omission-In1 and Omission-In2)
	Scenario2/No	Omission-Out1	OFailure1 or Omission-In2
	Scenario3/No	Value-Out1	VFailure1 or Value-In1 or (Omission-In1 and Value-In2)

design and only design intentions are encoded, it is still possible, using compositional safety analysis, to evaluate the suitability of the proposed design under these encoded intentions on the failure logic, fault propagation, fault mitigation and fault tolerance of various architectural components. Such analysis can support safety analysts in the identification of weaknesses and to decide how to improve the product line architectural design e.g., by introducing components with improved characteristics or fault tolerant features. At the end of the product line safety analysis phase, the product line failure model with hazards and component failure data associated with each usage scenario considered in the analysis is then, delivered.

After the product derivation, variant-specific safety analysis in product line application engineering can be performed in the same way as defined in domain engineering, with the focus in identifying the impact of variant-specific system functions added to the reused product architecture model on the overall system safety.

5.2.2 SC-PLE-7: Product Fault Tree Analysis and FMEA

The capability of automatic synthesis of fault trees and FMEA provided by compositional safety analysis techniques may reduce the effort in performing variant-specific safety assessment by reusing not all, but a set of variant-specific hazards and component failure logic previously identified in domain engineering. If applicable, after performing variant-specific safety analysis to identify variant-specific hazards and component failure data, compositional safety analysis techniques are used to automatically generating fault trees and FMEA results from a variant-specific architecture model annotated with safety-related information. Compositional safety analysis techniques output variant-specific fault trees and FMEA in a graphical representation or in XML files. For example, HiP-HOPS outputs variant-specific fault trees and FMEA results in an XML file and presents it in the form of hyperlinked HTML pages. OSATE AADL/Error Annex outputs graphical representations of fault trees. The accuracy of the generated variant-specific fault trees and FMEA is dependent upon whether functional hazard assessment and component failure analysis were performed aware of the impact context and design variation.

5.2.3 SC-PLE-8: Product SIL Decomposition

Variant-specific fault trees and FMEA results generated in the previous phase are input artefacts for SIL decomposition optimization algorithms (SOKOROS *et al.* 2015;

AZEVEDO *et al.* 2014, DHOUIBI *et al.* 2014, AZEVEDO *et al.* 2013, PARKER et al. 2013, MADER *et al.* 2012, BIEBER et al. 2011, PAPADOPOULOS et al. 2010, LEE et al. 2009). In this phase, SIL decomposition optimization algorithms are used to support the automatic decomposition of safety integrity requirements allocated to variant-specific hazards throughout the contributing component failure modes. In this thesis, we have used HiP-HOPS Tabu Search ASIL/DAL (SOKOROS *et al.* 2015; AZEVEDO *et al.* 2014) metaheuristic optimization algorithms to support the automatic decomposition of safety integrity requirements in automotive and aerospace systems. Metaheuristic algorithms do not guarantee finding optimal allocation solutions; however, they are capable of providing near optimal allocations within acceptable time spans. In the same way as fault trees and FMEA results, the accuracy of variant-specific SIL decomposition results is dependent upon whether product line safety analysis was performed aware of the impact of context and design variation. SIL decomposition results define the safety objectives for a specific system variant achieving process-based certification.

5.2.4 SC-PLE-9: Product Line Component SIL Decomposition

After generating SIL decomposition results for a range of product variants considered during product line safety analysis, product line component SIL decomposition can performed to support product line process-based certification. Multi-variant SIL decomposition results are input artefacts for this phase. These artefacts are then analyzed in order to identify the safety integrity requirements to be allocated to product line components address certification requirements. The output of this phase is a list of product line component SILs. This phase is supported by a method and tool, developed in this thesis as an extension to HiP-HOPS design optimization tool, to support the automated analysis and allocation of safety integrity requirements to product line components. The proposed method and tool are detailed in the following section.

5.3 Product Line Component SIL Allocation Method and Tool

This section presents the proposed method and tooling support for automatic allocation of safety integrity requirements to components of a safety-critical product line

(OLIVEIRA *et al.* 2015a). SIL decomposition in product line design contributes to define safety objectives to achieve product line process-based certification without being stringent or expensive. The method extends the capabilities of the HiP-HOPS design optimization method and tool to address product line design. Whereas product line components and their failure modes may receive different safety integrity requirements in different product variants, the ability to use an architectural component safely across a range of different variants means selecting the highest SIL given by multiple SIL decomposition results. The method describes how to perform the analysis of multiple SIL decomposition results to extract the safety integrity requirements that should be allocated to architectural components. The method was implemented in a prototype tool developed as an extension of HiP-HOPS Tabu Search SIL allocation tool. The capability of automatic allocation of safety integrity requirements to product line components can also be further incorporated into existing design optimization tools other than HiP-HOPS. Section 5.3.1 presents the proposed method, and Section 5.3.2 presents the implementation of the method in an automated tool support.

5.3.1 Product Line Component SIL Decomposition Method

Figure 5.6 shows the steps of the proposed method to support the product line component SIL decomposition process. The input artefacts for the proposed method are SIL decomposition results for multiple product variants (Figure 5.6a). HiP-HOPS Tabu Search SIL decomposition tool outputs SIL decomposition results for an individual product variant in a structured XML file. Firstly, each variant-specific SIL decomposition result is parsed to allow the manipulation of the information stored into these artefacts (Step 1 in Figure 5.6b). In the following, variant-specific SIL decomposition files are analyzed one by one, to obtain the SILs allocated to product line components in each product variant (Step 2 in Figure 5.6c). This is done for each product line component in the given variant by analyzing the SILs allocated to failure modes associated with the given component in an individual product variant. Therefore, the most stringent SIL allocated to a failure mode associated with that component is, then, the assumed SIL for that component in the analyzed variant. For each individual product variant, this is repeated for all components. After obtaining the SILs allocated to product line components in each individual product variant, for each product line component, the analysis is performed as follows: the SILs allocated to a particular component in different product variants are analyzed in order to verify the most stringent SIL allocated to that component across the analyzed product variants (Step 3 in Figure 5.6d). Finally, the SIL

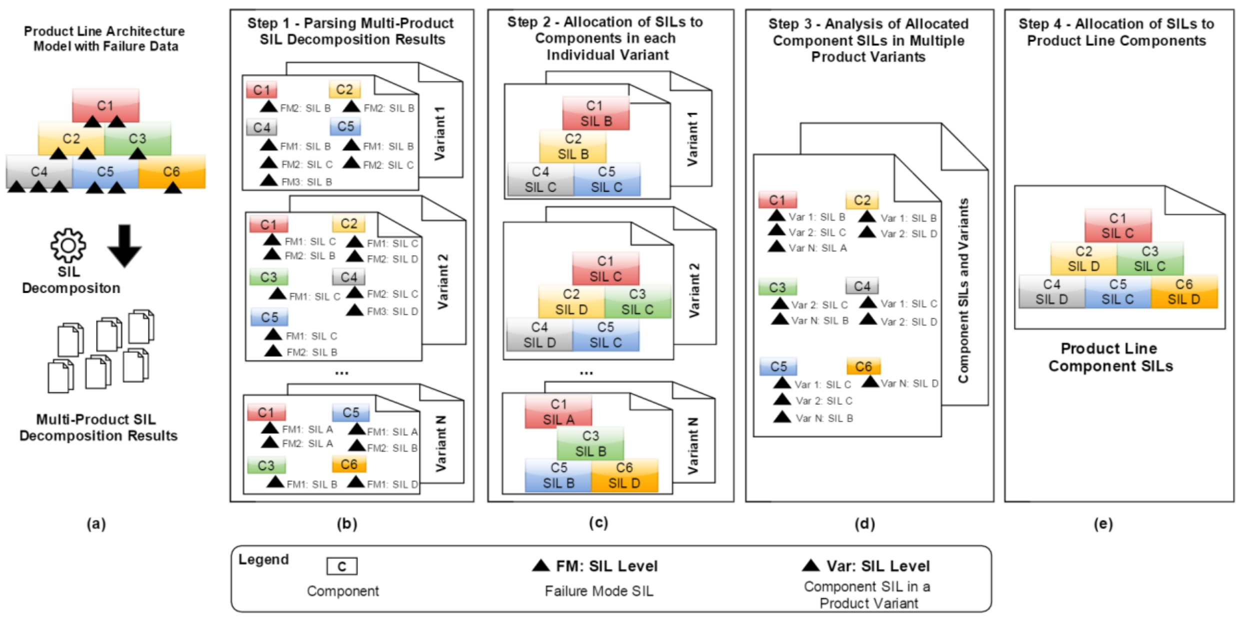

Product Line Architecture Model with Failure Data
SIL Decompositon
Multi-Product SIL Decomposition Results
Step 1 - Parsing Multi-Product SIL Decomposition Results
Step 2 - Allocation of SILs to Components in each Individual Variant
Step 3 - Analysis of Allocated Component SILs in Multiple Product Variants
Step 4 - Allocation of SILs to Product Line Components
Variant 1
Variant 2
Variant N
Component SILs and Variants
Product Line Component SILs
(a)
(b)
(c)
(d)
(e)
Legend
Component
FM: SIL Level
Failure Mode SIL
Var: SIL Level
Component SIL in a Product Variant

Figure 5.6. Method for automatic decomposition of SILs to product line components.

that each product line component should meet to address process-based certification is obtained, and the results, i.e., the SILs allocated to all product line components, are then exported in a file format such as XML (Figure 5.6e).

"*SIL Decomposition*" and "*Product Line Component SIL Decomposition*" metamodels were created to support the method illustrated in Figure 5.6. The "*SIL Decomposition*" metamodel defines the structure for variant-specific SIL decomposition results input artefacts provided by design optimization tools. The "*Product Line Component SIL Decomposition*" metamodel defines the structure of the standard output for the product line component SIL decomposition results. In safety-critical product line development, SIL decomposition has an important role in identifying, early on the design, safety integrity requirements necessary to comply with safety standards in order to achieve product line process-based certification. Both metamodels were used in the implementation of the automated tool support for product line component SIL decomposition that extends the HiP-HOPS capabilities to address product lines. These metamodels can be further used to define standard input and output SIL decomposition artefacts in extending other existing SIL decomposition tools to support product line design.

Figure 5.7 shows the "*SIL Decomposition*" and "*Product Line Component SIL Decomposition*" metamodels. The "*SIL Decomposition*" metamodel has been defined from the analysis of the output SIL decomposition artefacts provided by different design optimization techniques and tools. Particularly, the "*SIL Decomposition*" metamodel presented in Figure

5.7a was built upon the SIL decomposition output format provided by HiP-HOPS SIL decomposition optimization tool. A standard SIL decomposition output model comprises a root element that stores the reference to the system model from which the SIL decomposition results were generated. The "*TreeSILDecompositionResults*" root element might contain one or more "*Components"*. A component may be associated with zero or more failure modes ("*BasicEvents"*), and a component may contain subcomponents. Finally, a "*BasicEvent*" element refers to a component failure mode and its respective SIL. Component SILs can be derived from the information stored into "*BasicEvent*" elements.

Figure 5.7b shows the "*Product Line Component SIL Decomposition*" metamodel. "*PLSILDecompositionResults*" is the root the element of a product line component SIL decomposition model that contains references to "*PLComponentSIL*" elements. A "*PLComponentSIL*" model element stores the information about the required SIL to be allocated to a given component addressing process-based certification. A "*PLComponentSIL*" contains one or more "*VariantSpecComponentSIL*" allocation results, and it may also contain subcomponent SIL allocation results. "*VariantSpecComponentSIL*" element stores a SIL allocation result for a product line component in a specific product variant. A "*VariantSpecComponentSIL*" element contains references to one or more failure mode SIL al-

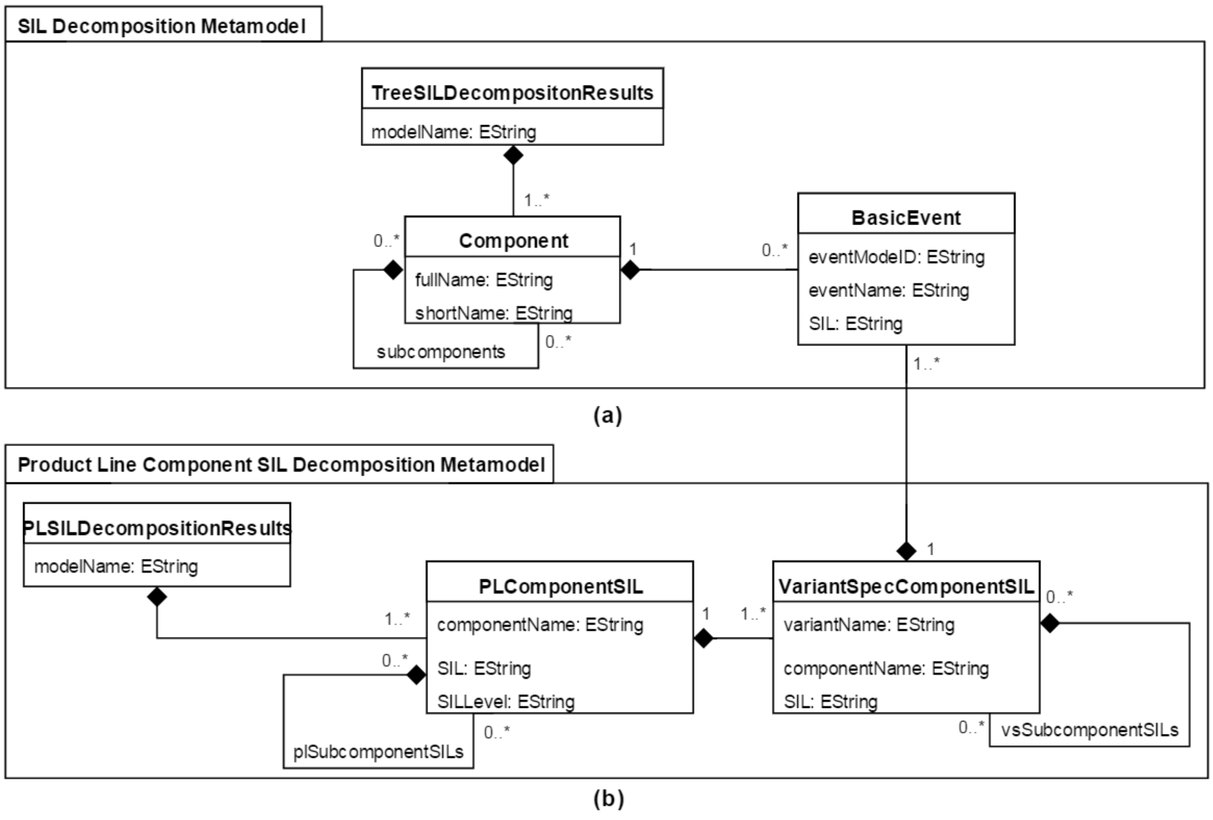


Figure 5.7. SIL decomposition metamodels.

location results (i.e., "*BasicEvents*") associated with a particular component in a given product variant. Finally, a "*VariantSpecComponentSIL*" may contain zero or more subcomponent SIL allocation results.

5.3.2 Product Line component SIL Decomposition (PL-SILDec) Tool

Figure 5.8 illustrates the PL-SILDec tool architecture. The tool was developed in Java language as an extension of HiP-HOPS Tabu Search ASIL/DAL decomposition optimization tooling. The PL-SILDec tool supports the analysis of ASILs for automotive, and DALs for avionics safety-critical product lines. The HiP-HOPS ASIL/DAL decomposition tooling (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014) outputs SIL decomposition results for a particular product variant in an XML file. XML files related to SIL decomposition results for multiple product variants are input artefacts for PL-SILDec tool performing the analysis and allocation of safety integrity requirements to components of a safety-critical product line. The PL-SILDec is mainly composed by the following components:

- ***User Interface***: it provides the mean for the user input SIL allocation files associated with multiple product variants of a safety-critical product line. Currently, the user informs the directories of the SIL allocation files required for the analysis via command prompt;
- ***Parser:*** it performs the parsing of each SIL allocation XML file to allow the manipulation of information stored into these files as Java objects in compliance with the "*SIL Decomposition*" metamodel. Therefore, classes representing the structure of the "*SIL Decomposition*" metamodel were created to manipulate variant-specific SIL decomposition information stored into the XML file. XStream[19] Java library was used to manipulate SIL decomposition XML files as objects. This library provides mechanisms to serialize and de-serialize objects from/to XML;
- ***Configuration SIL Allocation Analyzer:*** it performs the analysis of a parsed SIL decomposition XML file of a particular product variant to allocate SILs to components in such variant. For each variant-specific SPL component the analysis is performed as follows: the SILs allocated to each failure mode associated to the given component are analyzed. The most stringent SIL allocated to a failure mode is, then, the SIL to be allocated for that component in the analyzed product variant. Such analysis outputs a

[19] http://x-stream.github.io/

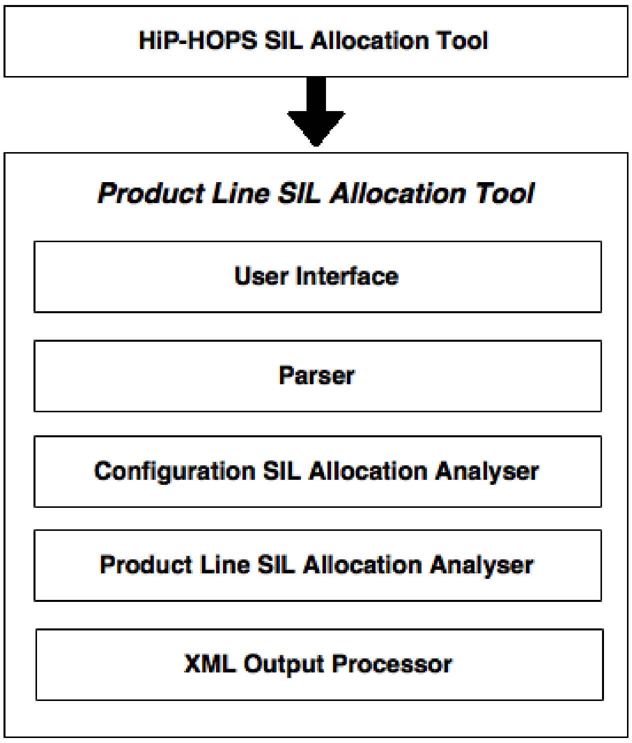

HiP-HOPS SIL Allocation Tool
Product Line SIL Allocation Tool
User Interface
Parser
Configuration SIL Allocation Analyser
Product Line SIL Allocation Analyser
XML Output Processor

Figure 5.8. Product line SIL decomposition tool architecture.

list of SILs allocated to components in a particular product variant. This module performs such analysis for each SIL decomposition XML file under analysis. Therefore, a list of SILs to be allocated to product line architectural components in each product variant is obtained. Such information is stored as objects as defined in "*Product Line Component SIL Decomposition*" metamodel;

- ***Product Line SIL Allocation Analyzer:*** it performs the analysis of the SILs allocated to product line components in multiple product variants to extract the SILs that should be allocated to components in order to address certification requirements across different product variants. For each component, the analysis is performed as follows: the SILs allocated to the given component in each product variant are analyzed, and the most stringent SIL is allocated to that component. The analysis outputs a list of product line components and their allocated SILs;
- ***XML Output Processor:*** it transforms product line component SIL allocation results into a structured output XML file based on the "*Product Line Component SIL Decomposition*" metamodel. XStream library is also used to serialize the object structure of the product line component SIL allocation results to output an XML file.

5.4 Compositional Safety Analysis Case Study: Hybrid Braking System Product Line

The proposed product line compositional safety analysis and design optimization approach was evaluated against safety assessment processes defined in ARP 4754A and ISO 26262 safety standards. The evaluation is focused on Hazard and Risk Analysis, Preliminary System Safety Assessment (PSSA), System Safety Assessment (SSA), and safety integrity requirements decomposition processes. The case study presented in this section is based on a prototype automotive Hybrid Braking System product line (HBS-SPL) earlier introduced in the variability management case study presented in Chapter 4 (Section 4.4).

HBS-SPL is a prototype automotive braking system product line designed in MATLAB/Simulink. It addresses electric vehicles system integration, in particular for propulsion architectures that integrate one electrical motor per wheel (DE CASTRO *et al.* 2010). From the analysis of the HBS-SPL core features and wheel braking variation point, seven different hybrid braking system variants can be derived from HBS-SPL as detailed in Chapter 4 (Section 4.4.1). The seven HBS-SPL system variants differ from each other in the number of wheel brake units and the way that they are composed. For example, "*Front Wheel Braking*" (FWB) variant differs from "*Four Wheel Braking*" (4WB) by the absence of rear wheel braking units. The "*Rear Wheel Braking*" (RWB) variant differs from 4WB by the absence of front wheel brake units. In addition, these seven HBS-SPL variant can be assumed to operate under different usage contexts. For example, a specific product variant can be deployed in a car, truck, or a military vehicle. Therefore, the composition between HBS-SPL functional features and their usage context generates 21 potential usage scenarios for HBS-SPL. Whereas the HBS-SPL functional features and their usage context are different in each product variant, hazard and risk assessment, and component failure analysis may change according to the usage scenario. The product line compositional safety analysis and design optimization case study has been performed by considering "*4WB*", "*FWB*" and "*RWB*" braking system variants. The HiP-HOPS compositional safety analysis and design optimization tools have been used to support product line safety assessment in both domain engineering and application engineering processes. The following sections present the results of compositional safety analysis and design optimization phases in domain engineering and application engineering.

5.4.1 SC-PLE Phase 3: HBS-SPL Safety Analysis

Hazards can arise from interactions between HBS-SPL components in a range of usage scenarios. Different hazards can arise according to how HBS-SPL functional and usage context features can be combined in a usage scenario. Although performing the product line safety analysis by considering all usage scenarios would be prohibitive, scoping such analysis to a set of scenarios has brought some degree of reuse for safety analysis process (OLIVEIRA *et al.* 2014). In order to illustrate the product line safety analysis process, such analysis was performed by considering three different HBS-SPL usage scenarios: "*4WB*", "*FWB*", and "*RWB*" braking system variants deployed in a car vehicle ("*SC-PLE-3.1: Identify Usage Scenarios*"). Each one of these scenarios was analyzed from the safety perspective by considering ISO 26262 hazard and risk assessment processes. The following subsections show the results of performing functional hazard assessment and component failure analysis by considering "*4WB*", "*FWB*", and "*RWB*" system variants.

5.4.1.1 Functional Hazard Assessment

The product line Functional Hazard Assessment (FHA) was performed based on the extended HAZOP (*HaZard and OPerability*) analysis technique (KLETZ, 1992) where *omission, commission, value, early,* and *late* guidewords were used to describe system hazards. The HiP-HOPS compositional safety analysis tool was used to support functional hazard assessment in the HBS-SPL Simulink model. HiP-HOPS tool was chosen as it provides integration with MATLAB/Simulink, the target development platform used for developing the HBS-SPL.

For each HBS-SPL usage scenario (product variant + context), functional hazard assessment started from the analysis of potential interactions between components in the HBS-SPL architecture model that may lead to system-level hazards. Such analysis is performed manually by safety analysts based on their domain knowledge. In this case, interactions between wheel braking components that can lead to hazardous events have been identified. Therefore, hazards and their associated failure conditions have been identified. Further on the analysis, risk assessment was performed where each identified hazard was classified regarding to its severity. Safety integrity requirements, in terms of Automotive Safety Integrity Levels (ASILs) were allocated to eliminate or minimize the hazard effects. Functional safety requirements have not been identified for the HBS-SPL.

Table 5.2 shows the identified hazards, their associated failure conditions, and allocated safety integrity requirements. The *hazard* term used throughout this case study refers to *potential source of harm* in ISO 26262. Table 5.2 also shows the association between the identified hazards and product line usage scenarios by means of the column "*Usage Scenario*". In order to simplify the case study, ASILs allocated to HBS-SPL hazards were not assigned on the basis of the full ISO 26262 (ISO, 2011) risk assessment process. The ASILs were derived by only considering the hazard's severity. Six hazards have been identified from the analysis of the "*4WB*" usage scenario, two hazards have been identified from the analysis of the "*FWB*" scenario, and other two hazards have been identified from the analysis of the RWB usage scenario. Common and variable hazards were identified by analyzing each HBS-SPL usage scenario. An example of common hazard is "*Value_braking*". Although the commonality, different failure conditions contribute to the occurrence of this hazard and different ASILs were allocated to this hazard in each usage scenario. The failure conditions that directly cause the *"Value_braking"* hazard in the *four wheel braking* scenario is "*incorrect value of all brake unit actuators output*", and "*incorrect value of rear wheel braking actuators output*" in the *rear wheel braking* scenario. In order to differentiate common hazards identified across the usage scenarios, each hazard was annotated with a prefix, identifying the corresponding usage scenario (see the "*Hazard ID*" column in Table 5.2). For example, the "*4WB*" prefix in "*Value_braking*" denotes that this hazard was identified from the analysis of *four wheel braking* scenario. The "*RWB*" denotes that *"Value_Braking"* was identified from the analysis of the *rear wheel braking* scenario.

Table 5.2 – Functional hazard assessment for the hybrid braking system product line architecture.

Usage Scenario	Hazard ID	Failure Conditions	ASIL
Four Wheel Braking (4WB) Deployed into a Car Vehicle.	No_braking_four_wheels	Omission of all brake unit actuators outputs.	D
	No_braking_three_wheels	Omission of BrakeUnit1 AND BrakeUnit2, AND BrakeUnit3 actuators outputs.	D
	4WB_No_braking_front	Omission of BrakeUnit1AND BrakeUnit2 actuators outputs.	D
	4WB_No_braking_rear	Omission of BrakeUnit3 AND BrakeUnit4 actuators outputs.	C
	No_braking_diagonal	Omission of BrakeUnit1 AND BrakeUnit4 actuators outputs OR Omission of BrakeUnit2 AND BrakeUnit4 actuators outputs.	C
	4WB_Value_braking	Incorrect Value of all brake unit actuators outputs.	D
Front Wheel Braking (FWB) Deployed into a Car Vehicle.	FWB_No_braking_front	Omission of BrakeUnit1 AND BrakeUnit2 actuators outputs.	D
	FWB_Value_braking	Incorrect Value of BrakeUnit1 AND BrakeUnit2 actuators outputs.	D
Rear Wheel Braking (RWB) Deployed into a Car Vehicle.	RWB_No_braking_rear	Omission of BrakeUnit3 AND BrakeUnit4 actuators outputs.	D
	RWB_Value_braking	Incorrect Value of BrakeUnit3 AND BrakeUnit4 actuators outputs.	D

In the hybrid braking system product line FHA process, different ASILs were allocated to the same hazard by considering different usage scenarios for HBS-SPL components. For example, ASIL C was allocated to "*No braking rear*" hazard in the "*4WB*" scenario, and a most stringent ASIL D was allocated to the same hazard in the "*RWB*" scenario. Variable hazards have also been identified by analyzing the HBS-SPL usage scenarios. For example, "*No_braking_four_wheels*", "*No_braking_three_wheels*", and "*No_braking_diagonal*" hazards are specific to the *four wheel braking* scenario. After identifying the potential hazards that can arise in different usage scenarios, component failure analysis is performed to identify the potential component failure modes that contribute to the occurrence of each identified hazard in the given usage scenario.

5.4.1.2 Component Failure Analysis

From the analysis of HBS-SPL functional hazard assessment results for each usage scenario, 77 component failure logic expressions inherent to 30 HBS-SPL architectural components were added to the product line failure model via HiP-HOPS failure editor. Table 5.3 shows fragments of the failure logic for six HBS-SPL components considering the "*4WB*", "*FWB*", and "*RWB*" usage scenarios. The "*Usage Scenario/Current*" column deno-

Table 5.3 - Failure logic for HBS-SPL architectural components.

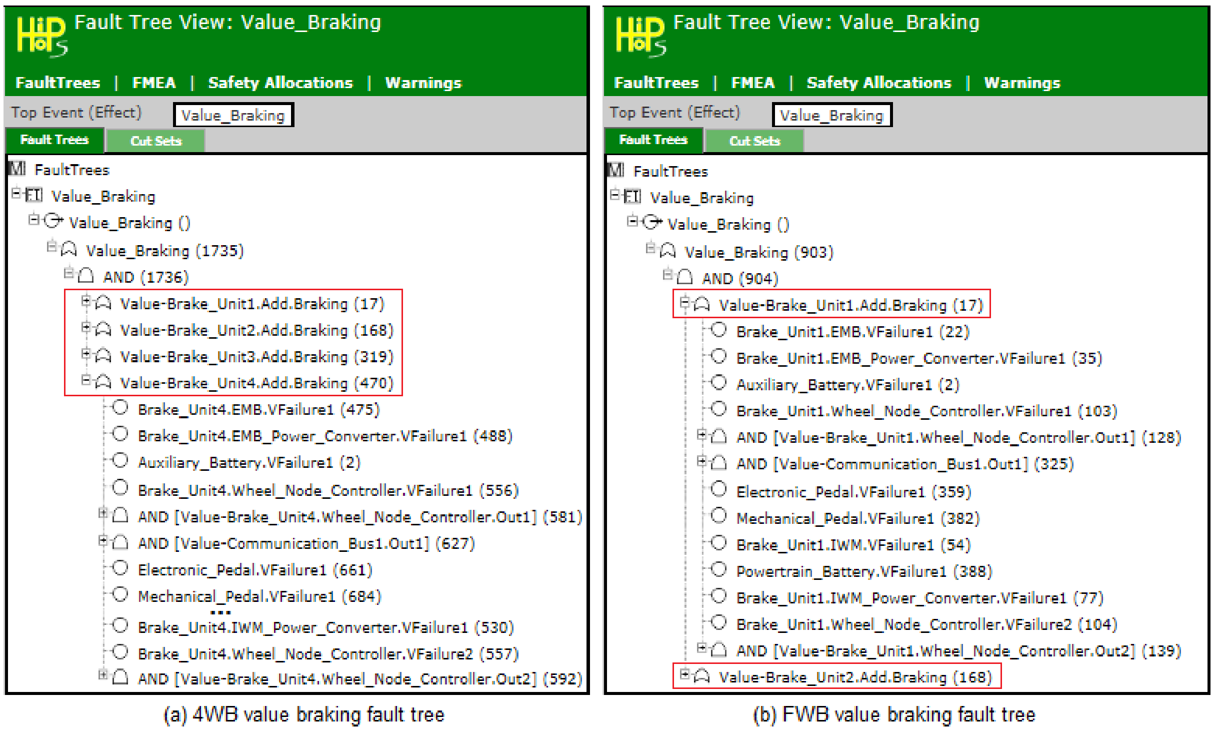
Component	Usage Scenario/Current	Output Deviation	Failure Expression
BrakeUnit3.WheelNodeController	4WB/Yes	Omission-Out1	OFailure1 or (Omission-In1 and Omission-In2)
		Value-Out1	VFailure1 or Value-In1 OR (Omission-In1 and Value-In2)
		Omission-Out2	OFailure2 or (Omission-In1 and Omission-In2)
		Value-out2	VFailure2 or Value-In1 or (Omission-In1 and Value-In2)
BrakeUnit3.EMBPowerConverter	4WB/Yes	Omission-Out1	OFailure1 or Omission-In1 or Omission-In2
		Value-Out1	VFailure1 or Value-In1 or Value-In2
BrakeUnit3.IWMPowerConverter	RWB/No	Omission-Out1	Omission-In1 or OFailure1 or Omission-In2
		Omission-Out2	Omission-In1 or OFailure1
		Value-Out1	Value-In1 or Value-In2 or VFailure1
		Value-Out2	Value-In1 or VFailure1
BrakeUnit4.EMB	RWB/No	Omission-Out1	OFailure1 or Omission-In1
		Value-Out1	VFailure1 or Value-In1
BrakeUnit1.IWM	FWB/No	Omission-Out1	OFailure1 or Omission-In1 or Omission-In2
		Omission-Out2	OFailure1 or Omission-In1 or Omission-In2
		Value-Out1	VFailure1 or Value-In1 or Value-In2
		Value-Out2	VFailure1 or Value-In1 or Value-In2
BrakeUnit1.EMB	FWB/No	Omission-Out1	OFailure1 or Omission-In1
		Value-Out1	VFailure1 or Value-In1

tes the usage scenario associated with the failure logic implementation and whether it is assumed as the current one for the given component. The failure logic for "*WheelNodeController"* (WNC) subcomponent from "*BrakeUnit3"* subsystem addresses the *4WB* usage scenario. Two omission and two value failure modes in WNC outputs were considered. The omission of the WNC outputs can be caused by an internal failure in WNC or omission of its inputs. WNC value output failures can be caused by WNC internal failure or incorrect value in one of the WNC inputs. The failure logic for "*BrakeUnit3.IWMPowerConverter"* and "*BrakeUnit4.EMB"* components are specific to the *RWB* variant. The failure logic for "*BrakeUnit1.IWM"* and "*BrakeUnit1.EMB"* components are specific to the *FWB* variant.

5.4.2 SC-PLE Phase 7: Product Fault Tree Analysis and FMEA

After deriving HBS-SPL variant-specific architecture models with safety-related information, with the support of a variant management tool (Chapter 4), each variant-specific model was input to HiP-HOPS compositional safety analysis tool performing the synthesis of fault trees and FMEA results. As a result, fault trees, failure cut sets, and FMEA results were generated for each HBS-SPL system variant. Fault trees were generated for each variant-specific hazard shown in Table 5.2. Therefore, 6 fault trees were generated for "*4WB"* product variant, 2 fault trees were generated for the "*FWB"* variant, and 2 fault trees were generated for the "*RWB"* variant, and FMEA tables were generated for each HBS-SPL system variant. Failure rates were not calculated.

Figure 5.9 shows excerpts of fault trees and failure cut sets generated for *Value_braking* hazard in "*4WB"* and "*FWB"* braking system variants. Other fault trees generated for the "*4WB*" system variant are available in Appendix B. The analysis of the generated fault trees shows the impact of variation in HBS-SPL architectural components on the hazard causes. Therefore, failure conditions that can directly cause the hazard are "*incorrect value of all brake unit outputs*" in the "*4WB"* fault tree (Figure 5.9a). On the other hand, the causes for the same hazard in the "*FWB"* fault tree (Figure 5.9b) are "*incorrect value of Brake_Unit1 and Brake_Unit2 outputs*". There are also differences in the component failure modes that indirectly may lead to the occurrence of *"Value_braking"* hazard in these system variants. For example, failures modes from "*Brake_Unit4"* subcomponents such as "*Brake_Unit4.EMB.VFailure1"* and "*Brake_ Unit4.EMB.PowerConverter.VFailure1"* may lead to the occurrence of "*Value braking*"


Fault Tree View: Value_Braking
FaultTrees | FMEA | Safety Allocations | Warnings
Top Event (Effect)
Value_Braking
Fault Trees
Cut Sets

```
FaultTrees
  Value_Braking
    Value_Braking ()
      Value_Braking (1735)
        AND (1736)
          Value-Brake_Unit1.Add.Braking (17)
          Value-Brake_Unit2.Add.Braking (168)
          Value-Brake_Unit3.Add.Braking (319)
          Value-Brake_Unit4.Add.Braking (470)
            Brake_Unit4.EMB.VFailure1 (475)
            Brake_Unit4.EMB_Power_Converter.VFailure1 (488)
            Auxiliary_Battery.VFailure1 (2)
            Brake_Unit4.Wheel_Node_Controller.VFailure1 (556)
            AND [Value-Brake_Unit4.Wheel_Node_Controller.Out1] (581)
            AND [Value-Communication_Bus1.Out1] (627)
            Electronic_Pedal.VFailure1 (661)
            Mechanical_Pedal.VFailure1 (684)
            ...
            Brake_Unit4.IWM_Power_Converter.VFailure1 (530)
            Brake_Unit4.Wheel_Node_Controller.VFailure2 (557)
            AND [Value-Brake_Unit4.Wheel_Node_Controller.Out2] (592)
```

(a) 4WB value braking fault tree

```
FaultTrees
  Value_Braking
    Value_Braking ()
      Value_Braking (903)
        AND (904)
          Value-Brake_Unit1.Add.Braking (17)
            Brake_Unit1.EMB.VFailure1 (22)
            Brake_Unit1.EMB_Power_Converter.VFailure1 (35)
            Auxiliary_Battery.VFailure1 (2)
            Brake_Unit1.Wheel_Node_Controller.VFailure1 (103)
            AND [Value-Brake_Unit1.Wheel_Node_Controller.Out1] (128)
            AND [Value-Communication_Bus1.Out1] (325)
            Electronic_Pedal.VFailure1 (359)
            Mechanical_Pedal.VFailure1 (382)
            Brake_Unit1.IWM.VFailure1 (54)
            Powertrain_Battery.VFailure1 (388)
            Brake_Unit1.IWM_Power_Converter.VFailure1 (77)
            Brake_Unit1.Wheel_Node_Controller.VFailure2 (104)
            AND [Value-Brake_Unit1.Wheel_Node_Controller.Out2] (139)
          Value-Brake_Unit2.Add.Braking (168)
```

(b) FWB value braking fault tree

Figure 5.9. Value braking fault trees for HBS-SPL system variants.

hazard in the "*4WB*" but not in the "*FWB*" variant. Multiple variant-specific fault trees provide the variability analysis of the causal chain of system-level hazards showing how component choices (i.e., feature selection) may change the component failure modes that can contribute to the occurrence of hazards in different product variants.

FMEA results describe the relationships between direct and further effects of component failure modes and the occurrence of system hazards. Tables 5.4 and 5.5 show fragments of FMEA results for "*4WB*" and "*FWB*" system variants considering the "*Brake_Unit1.EMB*" and "*Auxiliary_Battery*" component failure modes. These tables show the effects of the occurrence of failure modes and whether the failure mode is a single point of failure or not. For example, the occurrence of "*Brake_Unit1. EMB.OFailure1*" omission failure mode in the "*4WB*" variant, as illustrated in Table 5.4, indirectly contributes to the occurrence of "*No_braking_four_wheels*", "*No_braking_3_wheels*", "*No_braking_diagonal*", "*No_braking_front*", and "*No_bra-king_rear*" hazards in conjunction with other failure modes. On the other hand, the FMEA table for the "*FWB*" system variant from Table 5.5 shows that the occurrence of "*Brake_Unit1.EMB.OFailure1*" omission failure mode indirectly contributes to "*No_braking_front*" only in conjunction with other failure modes. The occurrence of an

Table 5.4 – Excerpt of FMEA results for 4WB system variant.

Component	**Failure Mode**	**System Effect**	**Single Point of Failure**
Auxiliary_Battery	VFailure1	Value_braking	True
Brake_Unit1.EMB	OFailure1 (21)	No_braking_4_wheels	False
		No_braking_3_wheels	False
		No_braking_diagonal	False
		No_braking_front	False
		No_braking_rear	False
	VFailure1 (22)	Value_braking	False

Table 5.5 – Excerpt of FMEA results for FWB system variant.

Component	**Failure Mode**	**System Effect**	**Single Point of Failure**
Auxiliary_Battery	VFailure1	Value_braking	True
Brake_Unit1.EMB	OFailure1 (21)	No_braking_front	False
	VFailure1 (22)	Value_braking	False

incorrect value in the "*Auxiliary_Battery*" component output directly contributes to the occurrence of "*Value_braking*" hazard in both "*4WB*" and "*FWB*" system variants, denoted by the "*true*" value in the "*Single Point Failure*" column in tables 5.4 and 5.5. The analysis of the FMEA results for both HBS-SPL variants has shown the impact of design variation on the cause-effect relationships between the contributing component failure modes and the occurrence of system-level hazards.

From the analysis of the complete FMEA results for "*4WB*", "*RWB*", and "*FWB*" braking system variants, 9 single-point and 39 multi-point failure modes were identified for the "*4WB*" variant, 5 single-point and 33 multi-point failure modes were identified for the "*RWB"* variant, and 5 single-point and 33 multi-point failure modes were identified for the "*FWB*" system variant. Such analysis has given insights into the design of the HBS-SPL, for instance common points of failure that affect different product variants and contribute to significant system hazards that arise across the product line, but also more subtle findings on the failures of individual system variants. It should be noted that existing design optimization techniques integrated to compositional safety analysis can also be used to support the automatic decomposition of safety integrity requirements allocated to variant-specific system hazards throughout the contributing failure modes. Therefore, from the ASILs allocated to variant-specific hazards, it is possible to automatically allocate ASILs to component failure modes, and then determine the safety integrity requirements that must be met for those components achieving process-based certification requirements for a particular system variant.

5.4.3 SC-PLE Phase 8: SIL Decomposition for HBS-SPL System Variants

The Tabu Search ASIL decomposition optimization algorithm provided by HiP-HOPS compositional safety analysis tooling was chosen to support the automatic ASIL decomposition in HBS-SPL braking system variants. The HiP-HOPS Tabu Search ASIL decomposition optimization algorithm and other existing design optimization techniques for SIL decomposition are detailed in Chapter 2 (Section 2.2.4.2). The failure cut sets, i.e., the fault trees and FMEA results, generated for a HBS-SPL system variant are input artefacts for the HiP-HOPS ASIL decomposition algorithm. The SIL decomposition was performed for "*4WB*", "*FWB*", and "*RWB*" braking system variants based on the following cost heuristic that expresses the relative cost jumps of developing a component according to different ASILs: 0 (ASIL QM), 10 (ASIL A), 20 (ASIL B), 40 (ASIL C), and 50 (ASIL D). This cost heuristic was used for illustrative purposes, but any other that the safety analyst finds more suitable can be used instead. Additional information on ASIL decomposition process can be found in ISO 26262 – Road Vehicles Functional Safety standard (ISO, 2011).

The HiP-HOPS Tabu Search ASIL decomposition algorithm was executed for each HBS-SPL system variant using the same cost heuristic. Table 5.6 shows an excerpt of the generated ASIL allocation results for "*4WB*", "*FWB*", and "*RWB*" system variants. These results correspond to the best ASIL decomposition solution found for each HBS-SPL system variant and their total ASIL cost. ASILs were allocated to 60 failure modes associated with 30 HBS-SPL components into three different product variants. The failure modes are stated in terms of omission (*OFailureID*) and value (*VFailureID*) failure types as shown in Table 5.6.

Table 5.6 – ASILs allocated to component failure modes across HBS-SPL system variants.

Component	Failure Mode	4WB ASIL	FWB ASIL	RWB ASIL
Auxiliary_Battery	OFailure1	QM(0)	B(2)	B(2)
	VFailure1	D(4)	D(4)	D(4)
Brake_Unit1.EMB	OFailure1	QM(0)	B(2)	-
	VFailure1	QM(0)	B(2)	-
Brake_Unit2.IWM	OFailure1	A(1)	QM(0)	-
	VFailure1	QM(0)	B(2)	-
…	…	…	…	…
Brake_Unit3.IWM_Power_Converter	OFailure1	QM(0)	-	B(2)
	VFailure1	B(2)	-	QM(0)
Brake_Unit4.Wheel_Node_Controller	OFailure1	B(2)	-	A(1)
	OFailure2	A(1)	-	B(2)
	VFailure1	B(2)	-	B(2)
	VFailure2	B(2)	-	B(2)
Mechanical_Pedal	OFailure1	D(4)	D(4)	D(4)
	VFailure1	D(4)	D(4)	D(4)
	ASIL Cost:	**830**	**740**	**740**

From the analysis of multiple variant-specific ASIL decomposition results, it has been found that the ASILs allocated to a particular component failure mode may change according to the HBS-SPL product variant. For example, the ASIL allocated to an "*OFailure1*" omission failure mode into the "*Auxiliary_Battery*" component is QM (0) in the "*4WB*" variant; and B (2) in both "*FWB*" and "*RWB*" variants. Such difference emphasizes that depending on the different ways in which HBS-SPL architectural components are composed in a given product variant, it may change the safety integrity of component failures, and the integrity of the component itself. It also impacts in changes in the total ASIL cost. The HiP-HOPS Tabu Search ASIL decomposition algorithm has calculated the total ASIL cost for each HBS-SPL product variant according to the cost heuristic previously explained earlier on this section. The ASIL cost for ASIL decomposition solutions were respectively 830 for "*4WB*", 740 for "*FWB*", and 740 for "*RWB*" product variant, as shown in Table 5.6. Therefore, from the analysis of multiple variant-specific ASIL decomposition results, it is possible to automatically allocate ASILs to components, and thereby determine the safety integrity requirements to be allocated for those components in order to fulfill the safety requirements to achieve product line process-based certification.

5.4.4 SC-PLE Phase 9: HBS-SPL Component SIL Decomposition

The set of variant-specific ASIL decomposition results generated with the support of HiP-HOPS ASIL decomposition optimization algorithm are input artefacts for the product line component SIL decomposition tool (PL-SILDec), which automatically allocates ASILs to HBS-SPL architectural components early on the design. The tool performs the allocation of safety integrity requirements to product line component based on the following principle: the most stringent ASIL allocated to a failure mode of a component across multiple product variants is the required ASIL to guarantee the safely usage of that component across the SPL (i.e., usage scenarios considered during product line FHA), in order to achieve product line process-based certification.

Firstly, the PL-SILDec tool derives the component ASILs in each HBS-SPL product variant from the analysis of the ASILs allocated to failure modes of components. For each component in the given variant, an analysis is performed to identify the most stringent ASIL allocated to a failure mode associated with that component. For example, considering the "*4WB*" system variant, the analysis of the ASILs allocated to

"*Brake_Unit3.IWM_Power_Converter*" component failure modes, (i.e., ASIL QM for "*OFailure1*" and ASIL B for "*VFailure1*") shows that the component ASIL for that variant should be ASIL B. Finally, for each HBS-SPL component, the tool analyses the ASILs allocated for that component in each product variant to extract the highest ASIL allocated to such component across the different product variants. ASILs were allocated to 30 HBS-SPL components.

Table 5.7 presents the ASILs allocated to 16 HBS-SPL components into three different variants considered to be deployed in a car vehicle. In the same way as ASILs allocated to component failure modes, the HBS-SPL component ASILs may change according to the product variant. For example, the ASILs allocated to "*Brake_Unit1.EMB*" and "*Brake_Unit1.EMB_Power_Converter*" components are respectively "QM", and "A" in *4WB* product variant, and "B" and "B" in *FWB* variant. From the analysis of the component ASILs allocated in multiple product variants, the PL-SILDec tool has derived the ASILs that should be allocated to HBS-SPL components, shown in the column "MAX ASIL" in Table 5.7, in order to achieve product line process-based certification. Thus, the HBS-SPL architectural components can be safely used across different product variants considered during product line safety analysis.

The analysis of the results about implications on safety integrity requirements of possible usage of safety-critical components early on the design provides useful feedback to the product line development process, helping to meet safety requirements, achieving process-based certification without incurring unnecessary costs. It may guide safety engineers to ta-

Table 5.7 – HBS-SPL component ASILs.

HBS-SPL Component	**4WB ASIL**	**FWB ASIL**	**RWB ASIL**	**MAX ASIL**
Auxiliary_Battery	D (4)	D (4)	D (4)	D (4)
Brake_Unit1.EMB	QM (0)	B (2)	-	B (2)
Brake_Unit1.EMB_Power_Converter	A (1)	B (2)	-	B (2)
Brake_Unit1.IWM	A (1)	B (2)	-	B (2)
Brake_Unit1.IWM_Power_Converter	A (1)	B (2)	-	B (2)
Brake_Unit1.Wheel_Node_Controller	A (1)	B (2)	-	B (2)
…	…	…	…	..
Brake_Unit4.EMB	B (2)	-	B (2)	B (2)
Brake_Unit4.EMB_Power_Converter	B (2)	-	B (2)	B (2)
Brake_Unit4.IWM	B (2)	-	B (2)	B (2)
Brake_Unit4.IWM_Power_Converter	B (2)	-	B (2)	B (2)
Brake_Unit4.Wheel_Node_Controller	B (2)	-	B (2)	B (2)
Communication_Bus1	B (2)	B (2)	B (2)	B (2)
Communication_Bus2	B (2)	B (2)	B (2)	B (2)
Electronic_Pedal	D (4)	D (4)	D (4)	D (4)
Mechanical_Pedal	D (4)	D (4)	D (4)	D (4)
Powertrain_Battery	D (4)	B (2)	B (2)	D (4)

ke design decisions in order to achieve process-based certification, ensuring that product line architectural components can be safely used across the usage scenarios. Product line component SILs may also guide safety engineers in establishing safety objectives for less and higher critical product line components in order to achieve process-based certification without being stringent or expressive. In traditional safety standards, different safety objectives are defined to achieve process-based certification according to the component integrity level. Safety objectives demand development and safety assessment guidance/activities and output artefacts that should be considered in the development process to address certification requirements of a given SIL level. By considering "*Auxiliary_Battery*" and "*Brake_Unit4.Whee-lNodeController*" components and their ASIL levels, Table 5.8 shows a subset of hardware and software development activities/guidance required to achieve process-based certification in compliance with ISO 26262 safety standard. Whereas the "*Auxiliary_Battery*" is a high integrity hardware and software component (i.e., ASIL D), according to "*ISO 26262-5: Product Development at the Hardware Level*" and "*ISO 26262-6: Product Development at the Software Level*", the following development and safety assessment activities/guidance are prescribed for "*Auxiliary_Battery*" component achieving process-based certification: deductive analysis (e.g., Fault Tree Analysis) and inductive analysis (e.g., FMEA) are highly recommended, hierarchical design and development by hardware prototyping are recommended, hardware design walk-through is optional (i.e., there is no recommendation for or against its usage for the identified ASIL), and the usage of established design principles is highly recommended.

For less-critical components, such as "*Brake_Unit4.WheelNodeController*" software component, less stringent safety objectives and development guidance were allocated, as shown in Table 5.8. According to ISO 26262, examples of activities that should be allocated to the "*Brake_Unit4.WheelNodeController*" component are: inspection of the design is highly recommended, and semi-formal verification and data-flow analysis are recommended. Therefore, instead of allocating highly integrity development and assessment activities/guidance to all product line components, such type of allocation allows safety analysts allocating development and assessment activities to a component or a subset of components according to their level of integrity. Less stringent activities allocated to low integrity components means a reduction of development and safety assessment effort and costs. This type of allocation contributes to address the safety requirements to address process-based certification without compromising safety.

Table 5.8 – Safety objectives allocated to HBS-SPL components per ASIL level.

HBS-SPL Component	ASIL	Activity/Guidance	Section
Auxiliary_Battery	D	Hierarchical hardware design (recommended)	ISO-26262-5: Section 7.4.1.6
		…	…
		Deductive Analysis (highly recommended)	ISO 26262-5: Section 7.4.3.1
		Inductive Analysis (highly recommended)	ISO 26262-5: Section 7.4.3.2
		Development by hardware prototyping (recommended)	ISO 26262-5: Section 7.4.4.1
		Hardware design walk-through (optional)	ISO 26262-5: Section 7.4.4.1
Brake_Unit4.WheelNodeController	B	Inspection of the design (highly recommended)	ISO 26262-6: Section 7.4.18
		…	…
		Semi-formal verification (recommended)	ISO 26262-6: Section 8.4.5
		Data-flow analysis (recommended)	ISO 26262-6: Section 8.4.5

5.5 Discussion and Limitations

The product line compositional safety analysis and design optimization approach proposed in this thesis was built upon safety assessment processes defined in safety standards from automotive and aerospace domains to address the product line safety assessment. By applying the proposed approach in automotive and aerospace safety-critical product lines it has been shown evidence of compliance with ISO 26262 Functional Safety Road Vehicles and ED-79A/ SAE ARP 4754A and ARP 4761 safety standards. The product line safety analysis in both domain and application engineering delivers as outputs hazard and risk assessment, and component failure analysis in compliance with Hazard Analysis and Risk Assessment processes defined in ISO 26262 – Part 3, Sections 7 and 8, and Functional Hazard Assessment phase defined in ARP 4754A item 5.1.1.

In product line application engineering, the automatic synthesis of product fault trees and FMEA results from the reused failure model in the proposed approach addresses Deductive (e.g., fault trees, Dependence Diagrams, Markov analysis) and Inductive (e.g., FMEA, Event Tree Analysis - ETA) analyses techniques prescribed by Safety Analysis process defined in ISO 26262 – Part 9, Section 8. In ARP 4754A, the synthesis of product fault trees and FMEA results complies with Preliminary System Safety Assessment (PSSA), and System Safety Assessment (SSA) phases defined in items 5.1.2 and 5.1.3 from ARP 4754A. The outputs of variant-specific SIL decomposition results in product line application engineering comply with ISO 26262 – Part 9, Section 5 – "*Requirements decomposition with respect to ASIL tailoring*", and ARP 4754A item 5.2 – "*Development assurance level assignment*". In domain engineering, the output of product line component SIL decomposition also complies with ISO 26262 ASIL decomposition and ARP 4754A DAL assignment processes. The product line FHA and component failure data, and product line component SILs generated in domain engineering, and variant-specific FHA and component failure data, fault trees, FMEA, and SIL decomposition generated in application engineering comply with ISO 26262 – Part 9 Sections 5, 6 and 8. These safety assessment artefacts also comply with ARP 4754A Functional Hazard Assessment (FHA), Preliminary System Safety Assessment (PSSA), and System Safety Assessment (SSA) processes.

Whereas a product line approach provides a range of potential product variants, it would be prohibitive performing functional hazard assessment and component failure analysis covering all possible scenarios for a target product line. Therefore, product line safety analysis cannot be fully automated because it is dependent upon the domain knowledge from analysts. To achieve the effective reuse of safety assets in safety-critical product line engineering processes it is feasible to consider a set of usage scenarios relevant to the target domain addressed by the product line and their stakeholders. The precision in the selection of relevant usage scenarios to be considered in product line safety analysis is subjective and dependent upon the safety analyst's domain knowledge. This is one limitation of the product line compositional safety analysis and design optimization approach that can be minimized with safety analyst knowledge about previous safety analysis performed in other systems from the same domain and via interactions with stakeholders. Metaheuristics optimization algorithms can be a solution to support safety analysts to take decisions in the selection of relevant usage scenarios to be considered during product line safety analysis, but it is outside the scope of this thesis and subject of future research in product line safety assessment.

5.6 Summary

Achieving the systematic reuse of safety assets in safety-critical product line engineering requires performing safety analysis aware of interactions between architectural components in a range of usage scenarios and the impact of variation in design and context in safety properties. The novelty of the safety analysis approach presented in this chapter is the provision of an integrated and systematic method to integrate compositional safety analysis and design optimization into product line processes. In domain engineering, functional hazard assessment and component failure analysis is performed from the perspective of different ways in which common and variable architectural components can be combined in a range of scenarios, and variability in the product line failure model is managed. In application engineering, product line architecture and failure models are then reused, with the support of variant management techniques, and compositional safety analysis supports the automatic synthesis of variant-specific fault trees, FMEA, and SIL decomposition safety assets. Therefore, context and design variation is traced throughout design and safety assessment assets. The approach can be adapted to work with a range of model-based development and compositional safety analysis tools, e.g., OSATE AADL/Error Annex system modeling and compositional safety analysis tool. The approach was validated in safety-critical product lines from automotive and aerospace (see Chapter 8) domains, but it can be generalized to address other safety-critical product lines from different domains such as medical and railway. The benefits of the proposed approach are the systematic reuse of the failure model and automated generation of fault trees and FMEA safety assets provided by seamless integration between model-based development, compositional safety analysis and variant management techniques. It contributes to reduce the effort and costs of performing variant-specific safety analysis, since such analysis is performed from the reused failure model, and automated with the support of compositional safety analysis techniques.

With regard to product line design optimization, a method and tool for automated analysis and decomposition of safety integrity requirements in product line design was developed in the course of this thesis. The PL-SILDec is a prototype tool built as an extension of the HiP-HOPS SIL decomposition tool (AZEVEDO *et al.* 2014). The tool supports the automatic allocation of SILs to SPL components taking into account their possible usage across the product line. It has been discussed both in theory and through automotive and aerospace (Chapter 8) case studies, how such a method and tool can potentially reduce the cost of safety-critical product line development by allocating less stringent safety integrity requirements to architectural components whilst meeting process-based certification safety objectives. Through this method, it is possible to establish safety integrity requirements for product line components anticipating their potential usage in a number of product variants. The product line component SIL decomposition method extends and automates the SIL decomposition principles prescribed by safety standards to address process-based certification in the product line design. Variant-specific safety assets such as hazard analysis, fault trees, and FMEA generated with the support of the proposed approach can be further used as certification evidence referenced in the assurance case produced later in application engineering to address goal-based certification. Chapter 6 presents the integration of model-based assurance cases approach into software product line engineering processes to support the automatic generation of variant-specific assurance cases from a diverse set of design and safety assessment models.

Chapter 6

MODEL-BASED ASSURANCE CASES IN SOFTWARE PRODUCT LINE ENGINEERING

6.1 Introduction

The research contributions presented in the previous two chapters (Chapter 4 – Product Line Safety Variability Management, and Chapter 5 – Product Line Compositional Safety Assessment) supports establishing the main causal models for hazards in a product line. It allows identifying the failure modes of interest for individual components, and determining the safety properties of components, e.g., failure rates, and SILs. However, safety analysis is not the only form of evidence required for assurance. Other forms of evidence comprise: testing, formal verification, and field data. At the core of safety-critical product line engineering is the idea that product line assets can be stored in a repository – these are not simply the software components themselves, they can be information about the components, e.g., requirements document, architecture, safety analysis, test cases. When it comes to assurance, we want to store assurance artefacts – e.g., testing results. To show the relationship between analysis artefacts generated for product line instances and the required assurance evidence, it is necessary to understand how the analysis can be used to shape an assurance case for a product line instance that can be linked to product line assurance assets such as hazard analysis and risk assessment, fault tree analysis, and FMEA results.

If we are able to package up and reuse product line components, then, from the assurance perspective, we must show how the assurance evidence that can be generated for components can adequately support the assurance case for any given product line instance. Therefore, the Model-Based Assurance Cases (MBAC) (HAWKINS *et al.* 2015) approach can be integrated into product line processes to support the automatic generation of assurance cases for product line instances from a diverse set of system models. Thus, automated traceability between system models and the assurance case can be achieved. This chapter presents a systematic approach to integrate the MBAC approach into product line engineering processes to support the automatic generation of assurance cases for product line instances (Product line Model-Based Assurance Cases in Chapter 3). The approach was validated to generate assurance cases for instances of a hybrid braking system product line (HBS-SPL) with the support of the MBAC tool.

Section 6.2 presents a systematic approach to integrate MBAC into product line engineering processes. Section 6.3 presents the instantiation of the proposed approach in an automotive case study. Section 6.4 presents the results of an experimental study conducted to evaluate the feasibility of the MBAC approach in generating assurance cases for product line instances. Section 6.5 presents the limitations of the MBAC approach. Section 6.6 presents the summary of this chapter.

6.2 Product Line Model-Based Assurance Cases

This section presents a systematic approach to integrate model-based assurance cases techniques (See Section 2.4.3), specifically techniques based on the model-weaving approach (DEL FABRO *et al.* 2005), into product line engineering processes. The phases, activities, and tasks of the proposed approach were defined from the analysis of the required steps to prepare the MBAC environment to support the generation of assurance cases for instances of automotive braking system and flight control aerospace safety-critical product lines. The proposed approach comprises three phases in domain engineering, and two phases in application engineering. The proposed product line MBAC approach is not tied to: specific assurance case modeling notations, e.g., GSN and CAE, technologies to define system models and their mappings to assurance case patterns, e.g., Eclipse Modeling Framework (EMF) or Meta Object Facility (MOF), and model-based assurance case tool, e.g., MBAC (HAWKINS *et al.* 2015). In addition, the approach can be applied in an iterative and incremental fashion.

In domain engineering, the approach defines phases to configure the model-based assurance case environment as illustrated in the SADT (ROSS, 1977) diagram in Figure 6.1a.

Firstly, assurance case patterns should be defined (SC-PLE-10), followed by the definition of the structure of each SPL asset model that provides information to instantiate these patterns in a metamodel (SC-PLE-11), and finally, mapping links between pattern elements and information elements provided by a diverse set of SPL asset models, required to instantiate these patterns, are specified in the weaving model (SC-PLE-12). Assurance case patterns, SPL artefact metamodels, and the weaving model created in domain engineering are reusable assets used in application engineering, together with input variant-specific design and safety assessment models, to support the weaving of assurance case patterns and system models, generating assurance cases for product line instances. Figure 6.1b illustrates the approach phase (SC-PLE-13) defined in the application engineering to: configure the targeted MBAC instantiation program with assurance case patterns, SPL asset metamodels, the weaving model and variant-specific input artefacts for automatically generating assurance cases for a particular product line instance. The following sections detail each one of these phases.

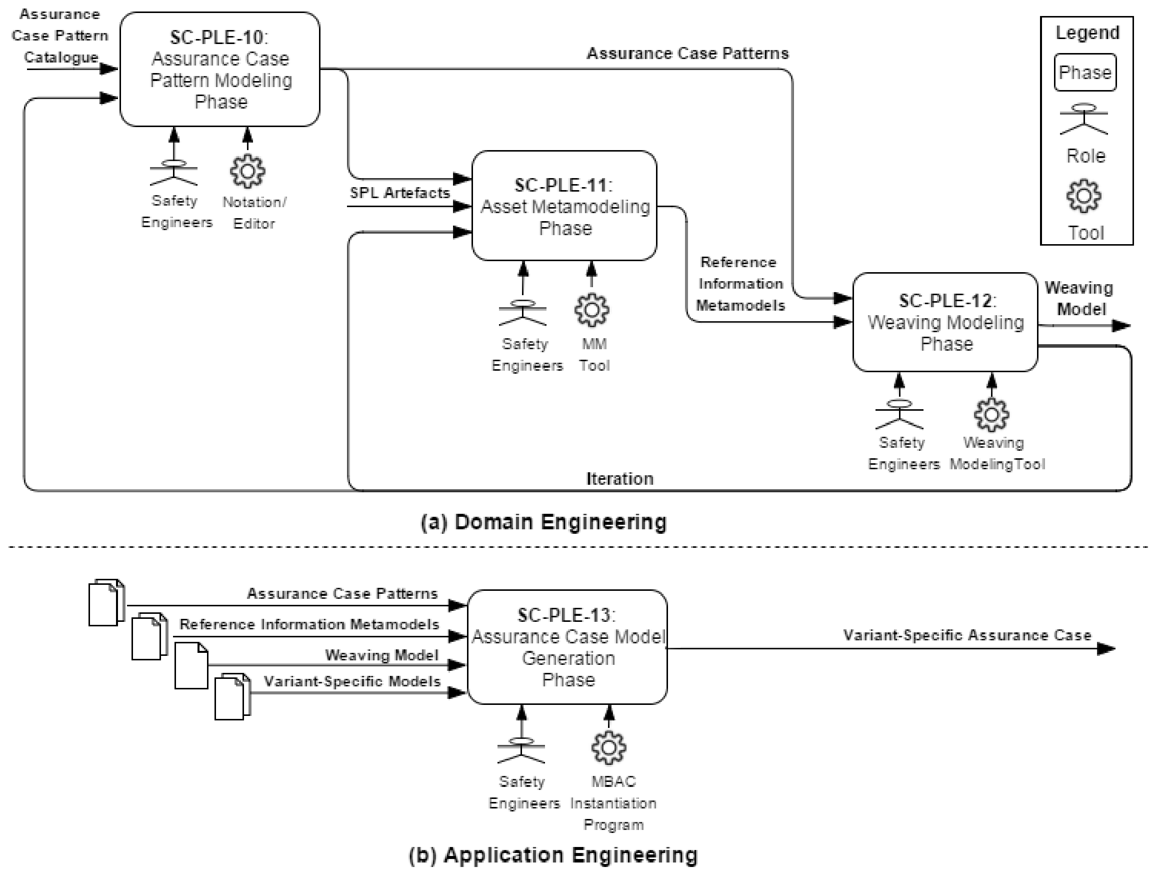

Assurance Case Pattern Catalogue
SC-PLE-10: Assurance Case Pattern Modeling Phase
Safety Engineers
Notation/ Editor
Assurance Case Patterns
SPL Artefacts
SC-PLE-11: Asset Metamodeling Phase
Safety Engineers
MM Tool
Reference Information Metamodels
SC-PLE-12: Weaving Modeling Phase
Safety Engineers
Weaving ModelingTool
Weaving Model
Iteration
Legend
Phase
Role
Tool
(a) Domain Engineering
Assurance Case Patterns
Reference Information Metamodels
Weaving Model
Variant-Specific Models
SC-PLE-13: Assurance Case Model Generation Phase
Variant-Specific Assurance Case
Safety Engineers
MBAC Instantiation Program
(b) Application Engineering

Figure 6.1. Model-based assurance cases in product line engineering processes.

6.2.1 SC-PLE-10: Assurance Case Pattern Modeling

The *Assurance Case Pattern Modeling* phase comprises a set of tasks shown in Figure 6.2. In this phase, safety engineers specify assurance case patterns with the support of an assurance case modeling notation and its graphical editor, e.g. GSN editor. Therefore, from the analysis of existing assurance case pattern catalogues (OLIVEIRA *et al.* 2014a; HABLI, 2009; HABLI and KELLY, 2006; YE, 2005; WEAVER, 2003; KELLY and McDERMID, 1997) safety engineers define the assurance case pattern architecture for the targeted system domain ("*SC-PLE-10.1*"). The established architecture comprises a hierarch of assurance case patterns that defines the structure of variant-specific assurance cases. Such architecture defines the assurance case pattern elements, e.g., goals and context elements in GSN, which require further instantiation. It defines pattern elements that are optional or that require multiple instances when an abstract *term* associated with the given multiplicity relation is bound during pattern instantiation. Section 2.4.2.2 provides a detailed description of multiplicity relations in GSN assurance case patterns.

After establishing the assurance case pattern architecture, safety engineers identify abstract *terms* in assurance case expression patterns, i.e., *terms* that require further instantiation (SC-PLE-10.2). Abstract *terms* in assurance case expression patterns are references to information elements from a diverse set of system models. Thus, abstract *terms*

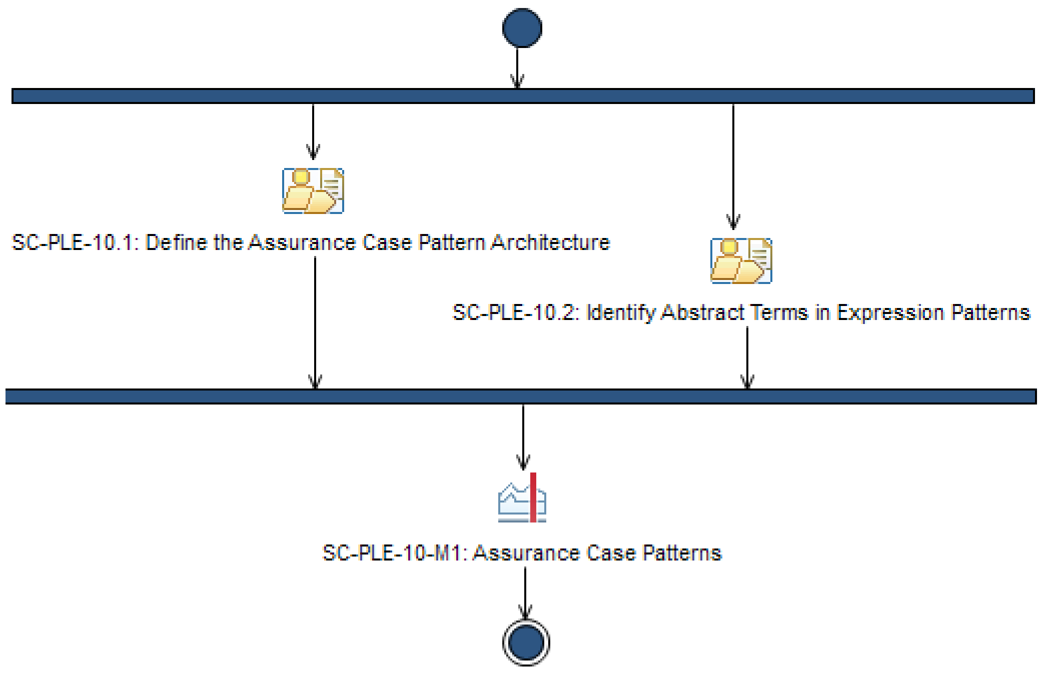

SC-PLE-10.1: Define the Assurance Case Pattern Architecture
SC-PLE-10.2: Identify Abstract Terms in Expression Patterns
SC-PLE-10-M1: Assurance Case Patterns

Figure 6.2. Assurance case pattern modeling phase.

are instantiated from the information provided by different system models. For example, "*systemX*" is an abstract *term* in the assurance case expression pattern shown in Figure 6.3, which refers to the "*system name*" information element stored into the system architecture model. Therefore, in SC-PLE-10.2, assurance case expression patterns are analyzed in order to identify *terms* that require further instantiation. This information is important for further linking abstract terms to elements from the system model in the weaving model. The end of this phase outputs a structured assurance case pattern hierarch and its associated abstract *terms*. The derived assurance case pattern architecture defines the structure of variant-specific assurance cases.

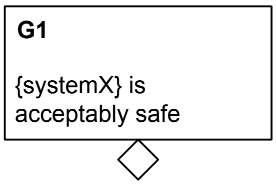


Figure 6.3 Example of an abstract term.

6.2.2 SC-PLE-11: Asset Metamodeling

As defined in the MBAC approach (Section 2.4.3.2), a reference information model represents a system model that contain information required to instantiate assurance case patterns. These models represent different views of a particular system, e.g., design, safety assessment and process (evidence) models, referenced in an assurance case. In this phase, with the support of a model-based development environment, e.g., EMF, safety engineers define the structure of product line assets in metamodels, for each asset intended to be input to the MBAC assurance case generation process. The end of this phase delivers a set of product line asset metamodels and graphical editors, in the form of plugins, generated from these metamodels. This is required to allow the underlying MBAC tool to handle the target product line asset models of a particular product line instance to extract the required information to generate a variant-specific assurance cases. The EMF platform can be used to support the design of these metamodels. Details about how to create metamodels and generate editors for domain-specific languages with the support of EMF can be found elsewhere (ECLIPSE, 2016).

6.2.3 SC-PLE-12: Weaving Modeling

In this phase, abstract *terms* defined in assurance case patterns are linked to system model elements, which contain the required information to instantiate these *terms* in the weaving model. The weaving model concept is detailed in Section 2.4.3.3. The weaving model can be specified manually or with the support of graphical editors, e.g., ATLAS Model Weaver[20] (DEL FABRO *et al.* 2005a), and yEd Graph Editor (YWORKS, 2016). Figure 6.4 shows the tasks to be performed during the weaving modeling phase. Assurance case patterns and product line asset metamodels are input artefacts for this phase. Firstly, elements referencing each abstract "*term*" defined in assurance case patterns should be specified in the weaving model (SC-PLE-12.1). Each element should have the same label of the corresponding abstract "*term*" defined in assurance case patterns. In the following, elements referencing each "*system model element*" that provides information to instantiate abstract "*terms*" should be specified in the weaving model (SC-PLE-12.2). Each "*system model*" referencing element comprises the pair *"element type, model name"*. Referencing elements for system model elements should also have the same label of the corresponding element defined into the system metamodel. Finally, mapping links between elements referencing abs-

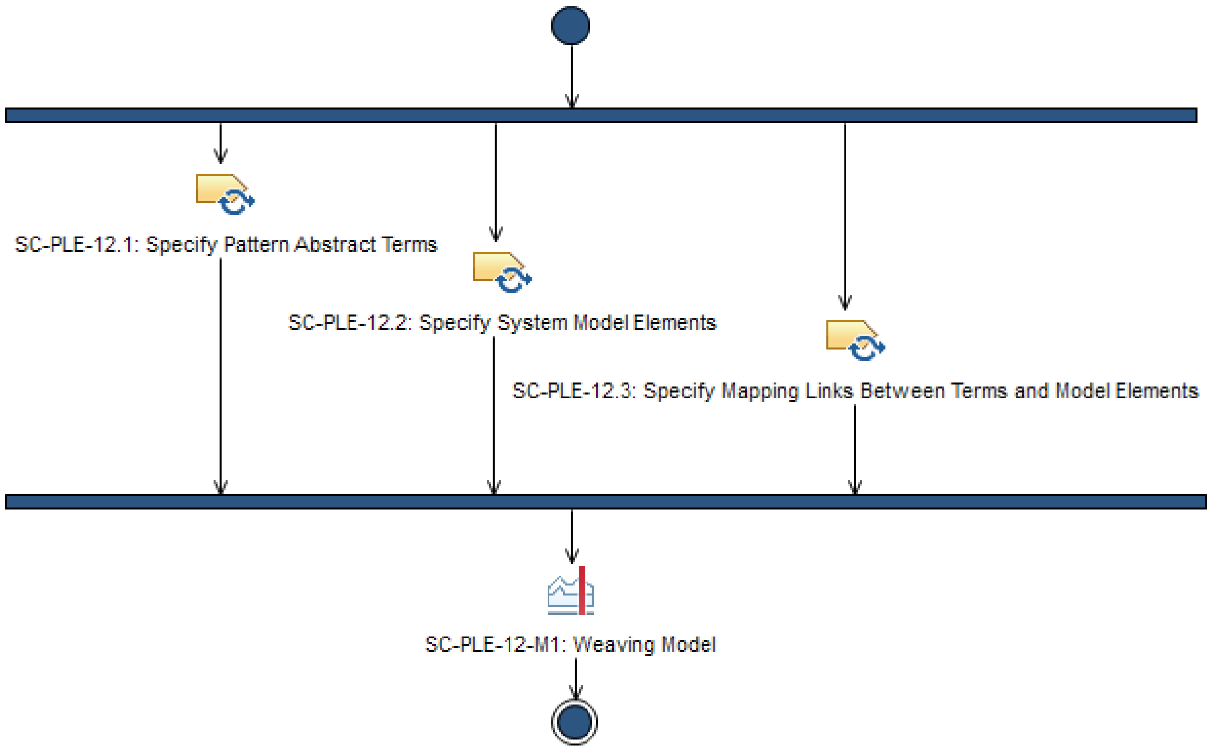

SC-PLE-12.1: Specify Pattern Abstract Terms
SC-PLE-12.2: Specify System Model Elements
SC-PLE-12.3: Specify Mapping Links Between Terms and Model Elements
SC-PLE-12-M1: Weaving Model

Figure 6.4. Weaving modeling phase.

[20] https://projects.eclipse.org/projects/modeling.gmt.amw

tract "*terms*" and "*system model*" referencing elements are specified (SC-PLE-12.3). A mapping link may optionally include two properties: "*targetProperty*", used when it is necessary referencing an attribute of the given "*system model element*", e.g., the attribute "*name*" from "*System*" model element; and *"isMulti"*, a Boolean property used control the amount of information required to instantiate a given abstract "*term*". A "*true*" value indicates that multiple occurrences of a given "*system model element*" are required to instantiate one occurrence of the given abstract "*term*", whereas a "*false*" value indicates that each occurrence of a given "*system model element*" results in a different instance for the given abstract "*term*". Figure 6.5 illustrates an example of a mapping link between "*systemX*" abstract "*term*" and "*system*" model element in the weaving model. This mapping link contains a property indicating that "*systemX*" requires the "*name*" attribute from *"system"* model element to be instantiated.

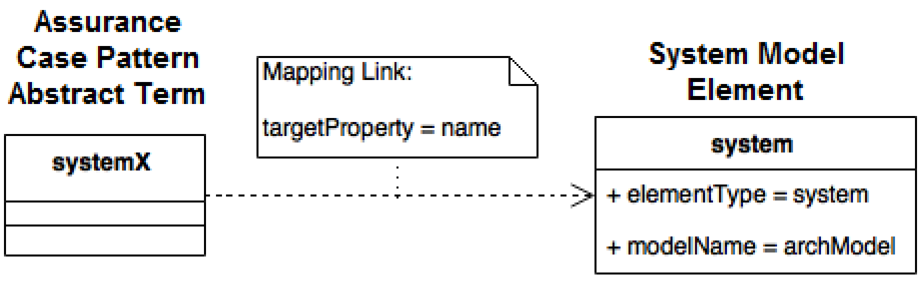


Figure 6.5. Linking pattern abstract term and system model element.

At end of this phase the weaving model is delivered. This model defines a pattern describing how assurance case pattern elements should be instantiated from the available evidence information provided by a diverse set of system models, by linking pattern elements to the evidence. Therefore, traceability between the assurance case and the referenced safety evidence is achieved, and changes in variant-specific system models are automatically propagated throughout the assurance case.

6.2.4 SC-PLE-13: Assurance Case Model Generation

In product line application engineering, assurance case patterns, reference information metamodels, and the weaving model defined in domain engineering, together with variant-specific design, assessment and process models are input artefacts to configure the targeted MBAC tool to generate the assurance case. The output of this phase is an assurance case model for a particular product line instance. The assurance case patterns, reference information metamodels, and the weaving model are reusable assets that can be used to generate assurance cases for multiple product line instances.

6.3 MBAC Case Study: Hybrid Braking System Product Line

This section presents a case study in which the proposed product line MBAC approach was applied, using the Goal Structuring Notation and its editor, MBAC approach and tooling (HAWKINS *et al.* 2015), to support the automatic generation of assurance cases for instances of the HBS-SPL automotive product line presented earlier in Chapter 4 (Section 4.4). The MBAC approach and tooling are detailed in Section 2.4.3. The following sections present the results obtained in each phase of the proposed approach.

6.3.1 SC-PLE Phase 10: HBS-SPL Assurance Case Pattern Modeling

From the analysis of existing assurance case patterns (HABLI 2009; WEAVER, 2003; KELLY and McDERMID, 1997), the assurance case pattern architecture was defined based on "*Hazard Avoidance*" (KELLY and McDERMID, 1997), "*Risk Argument*" (HABLI, 2009), and "*Absence of Hazardous Software Failure Mode*" (WEAVER, 2003) patterns, with the support of an EMF-based GSN editor. The assurance case pattern architecture is illustrated in Figure 6.6. Each assurance case pattern defined in this architecture is detailed in the following:

- An adapted version of *Hazard Avoidance* pattern that decomposes the claim over the safety of a given product line instance into sub-claims arguing that the risk posed by each hazard associated with the given instance is acceptable;
- The "*Risk Argument*" pattern was extracted from the analysis of assurance cases for an engine turbine avionics product line (HABLI, 2009). It decomposes the claim arguing the risk posed by a given system hazard into claims arguing the absence of contributing failure modes; and
- *Absence of Hazardous Software Failure Mode* argument pattern, which decomposes the claim arguing the "*absence of a given contributing component failure mode*" into sub-claims arguing the absence of primary, secondary, and control failure modes.

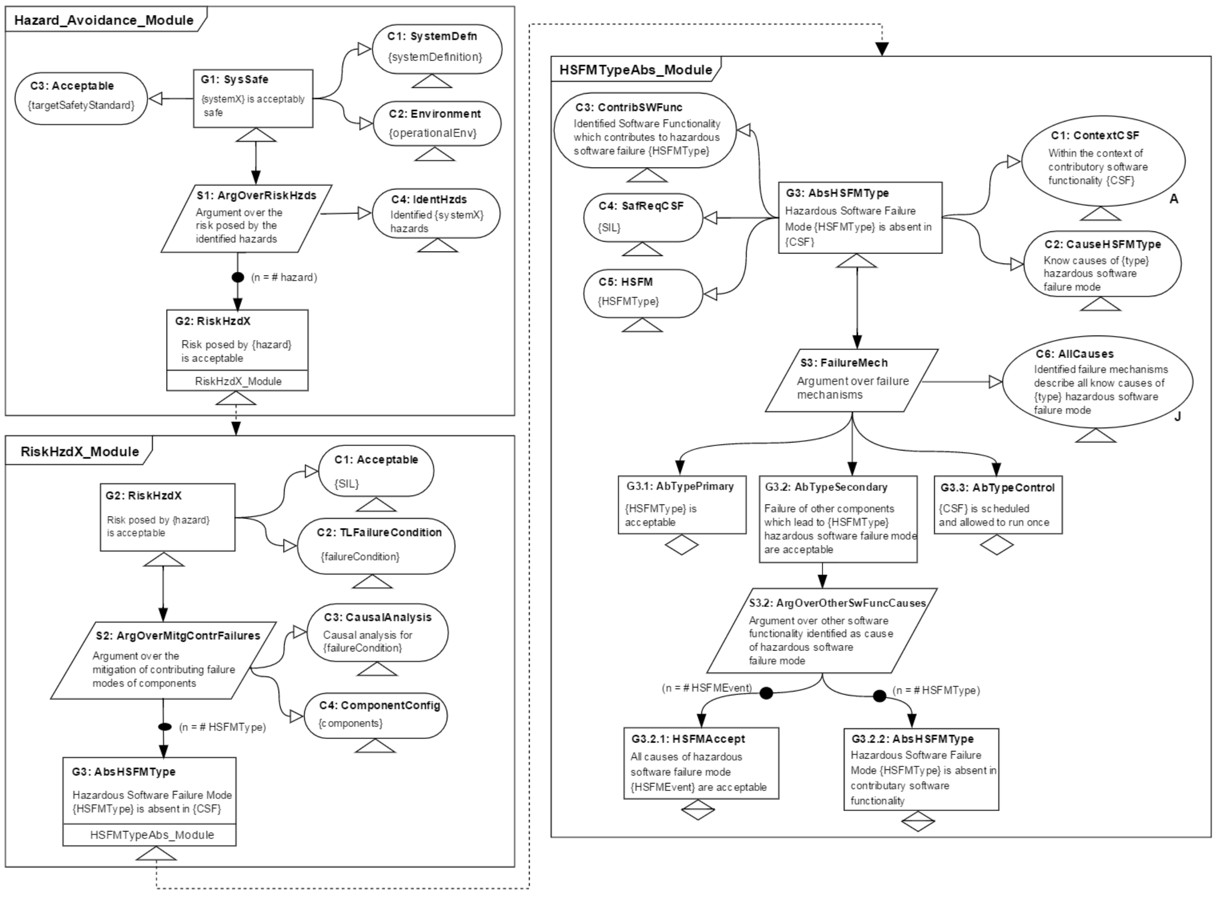

Hazard_Avoidance_Module
C1: SystemDefn
{systemDefinition}
C3: Acceptable
{targetSafetyStandard}
G1: SysSafe
{systemX} is acceptably safe
C2: Environment
{operationalEnv}
S1: ArgOverRiskHzds
Argument over the risk posed by the identified hazards
C4: IdentHzds
Identified {systemX} hazards
(n = # hazard)
G2: RiskHzdX
Risk posed by {hazard} is acceptable
RiskHzdX_Module
RiskHzdX_Module
C1: Acceptable
{SIL}
G2: RiskHzdX
Risk posed by {hazard} is acceptable
C2: TLFailureCondition
{failureCondition}
S2: ArgOverMitgContrFailures
Argument over the mitigation of contributing failure modes of components
C3: CausalAnalysis
Causal analysis for {failureCondition}
C4: ComponentConfig
{components}
(n = # HSFMType)
G3: AbsHSFMType
Hazardous Software Failure Mode {HSFMType} is absent in {CSF}
HSFMTypeAbs_Module
HSFMTypeAbs_Module
C3: ContribSWFunc
Identified Software Functionality which contributes to hazardous software failure {HSFMType}
C1: ContextCSF
Within the context of contributory software functionality {CSF}
A
C4: SafReqCSF
{SIL}
G3: AbsHSFMType
Hazardous Software Failure Mode {HSFMType} is absent in {CSF}
C2: CauseHSFMType
Know causes of {type} hazardous software failure mode
C5: HSFM
{HSFMType}
S3: FailureMech
Argument over failure mechanisms
C6: AllCauses
Identified failure mechanisms describe all know causes of {type} hazardous software failure mode
J
G3.1: AbTypePrimary
{HSFMType} is acceptable
G3.2: AbTypeSecondary
Failure of other components which lead to {HSFMType} hazardous software failure mode are acceptable
G3.3: AbTypeControl
{CSF} is scheduled and allowed to run once
S3.2: ArgOverOtherSwFuncCauses
Argument over other software functionality identified as cause of hazardous software failure mode
(n = # HSFMEvent)
(n = # HSFMType)
G3.2.1: HSFMAccept
All causes of hazardous software failure mode {HSFMEvent} are acceptable
G3.2.2: AbsHSFMType
Hazardous Software Failure Mode {HSFMType} is absent in contributary software functionality

Figure 6.6. Assurance case pattern architecture.

"Hazard Avoidance", "*Risk Argument"*, and "*Absence of Hazardous Software Failure Mode"* assurance case patterns define a structure for product-based and hazard directed argumentation. In addition, other assurance case patterns can also be added to this structure to support the automatic generation of process-based arguments (HABLI and KELLY, 2006) for a particular product line instance. A process-based argument pattern defines structure to argue the safety of the development processes considered during the development of safety-critical systems, in terms of quality of personnel and tools and compliance with safety standards. Figure 6.7 illustrates the "*Risk Argument"* pattern modeled using a GMF-based GSN graphical editor. This pattern defines a set of abstract "*terms*" that require further instantiation, denoted by a red rectangle. For example, "*hazard*", "*SIL*", and "*failure condition*" abstract *"terms"* require information provided by the Failure Model of the targeted product line instance. This pattern also defines that the "*G1*" claim arguing that the risk posed by a given hazard is acceptable is decomposed into multiple sub-claims arguing the absence of each contributing component failure mode (i.e., each node of a fault tree). This is indicated by "*n = # (hazard causes)"* expression supported by the relation between "*S1*" strategy and

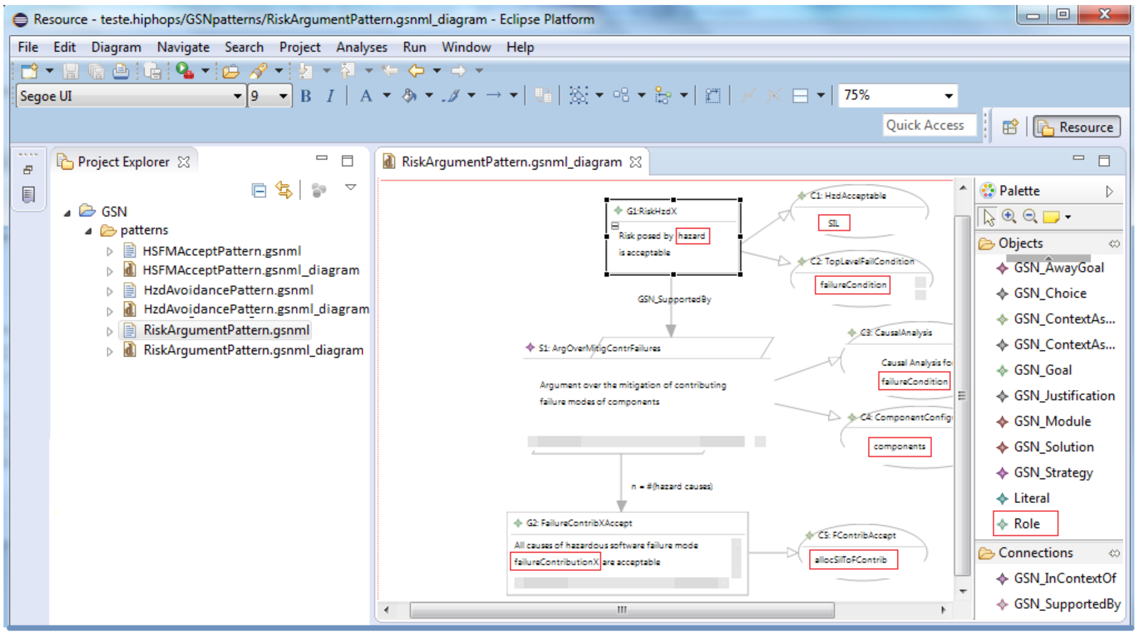

Resource - teste.hiphops/GSNpatterns/RiskArgumentPattern.gsnml_diagram - Eclipse Platform
File Edit Diagram Navigate Search Project Analyses Run Window Help
Segoe UI
9
75%
Quick Access
Resource
Project Explorer
GSN
patterns
HSFMAcceptPattern.gsnml
HSFMAcceptPattern.gsnml_diagram
HzdAvoidancePattern.gsnml
HzdAvoidancePattern.gsnml_diagram
RiskArgumentPattern.gsnml
RiskArgumentPattern.gsnml_diagram
RiskArgumentPattern.gsnml_diagram
G1:RiskHzdX
Risk posed by hazard is acceptable
C1: HzdAcceptable
SIL
C2: TopLevelFailCondition
failureCondition
GSN_SupportedBy
S1: ArgOverMitigContrFailures
Argument over the mitigation of contributing failure modes of components
C3: CausalAnalysis
Causal Analysis fo
failureCondition
C4: ComponentConfig
components
n = #(hazard causes)
G2: FailureContribXAccept
All causes of hazardous software failure mode failureContributionX are acceptable
C5: FContribAccept
allocSilToFContrib
Palette
Objects
GSN_AwayGoal
GSN_Choice
GSN_ContextAs...
GSN_ContextAs...
GSN_Goal
GSN_Justification
GSN_Module
GSN_Solution
GSN_Strategy
Literal
Role
Connections
GSN_InContextOf
GSN_SupportedBy

Figure 6.7. Risk argument pattern in the GSN editor.

"*G2*" claim.

The GSN editor (HAWKINS *et al.* 2015) generates both graphical and *".gsnml"* representations of assurance case patterns. The *".gsnml"* format provide the required information by MBAC tool instantiating GSN assurance case pattern elements and associated abstract *"terms"*. Figure 6.8 shows an excerpt of the *".gsnml"* representation of the "*Risk Argument*" pattern. Line 05 indicates that the "*G1*" claim should be instantiated, and line 07 indicates that the "*hazard*" abstract *term*, referenced in the expression pattern from that claim, should be instantiated with the information provided by the hazard analysis asset.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <gsnmetamodel:Case xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
03  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:gsnmetamodel="http://gsnmetamodel/1.0">
04  <contains xsi:type="gsnmetamodel:GSN_Module" id="RiskHzdX_Module">
05   <ArgumentElements xsi:type="gsnmetamodel:GSN_Goal" id="G1:RiskHzdX" tobeInstantiated="true">
06     <contents xsi:type="gsnmetamodel:Literal" literal="Risk posed by"/>
07     <contents xsi:type="gsnmetamodel:Role " id="hazard" role="hazard"/>
08     <contents xsi:type="gsnmetamodel:Literal" literal="is acceptable" />
09   </ArgumentElements>
10              ...
11  </contains>
12 </gsnmetamodel:Case>
```

Figure 6.8. Excerpt of *.gsnml* representation of the risk argument pattern.

6.3.2 SC-PLE Phase 11: HBS-SPL Asset Metamodeling

Considering that both HBS-SPL and Tiriba Flight Control product line case studies used throughout this thesis provide the same types of development and assessment assets, the following asset metamodels were adopted, and other were created with the support of Eclipse Modeling Framework:

- *AADL metamodel* (SAE, 2012) and *Simulink metamodel*[21]: represent the architectural view of the system;
- *Feature metamodel:* it was created from the analysis of the *"Product Line Metamodel"* (HABLI 2009), and feature metamodels provided by existing Eclipse-based feature modeling tools, e.g., FeatureIDE, and Feature Modeling Plugin (CZARNECKI *et al.* 2004). The feature model represents the functional view of the system;
- *Context metamodel:* it was created in the same way as the feature metamodel. A context model defines the characteristic of the environment where a given product line instance can operate;
- *Failure metamodel:* created based on the HiP-HOPS failure metamodel. It represents the failure analysis perspective of the system; and
- *Fault Tree-FMEA* metamodel: developed from the "*Architectural*" and "*Component*" failure metamodels (HABLI, 2009). It represents the structure of fault tree and FMEA results, i.e., the cause-effect view of the system failure behavior;

Instances of product line asset metamodels provide the required information for the MBAC tool instantiating assurance case patterns. Figure 6.9 illustrates a screenshot of the "*Failure Metamodel*" in EMF platform. EMF provides both graphical and tree view editors to support the specification of models and metamodels. The diagram view illustrates that "*Failure Metamodel*" instances can contain zero or more "*Hazards*", and the occurrence of a "*Hazard*" can be associated with one or more "*Causes*".

[21]https://github.com/FTSRG/massif/tree/master/plugins/hu.bme.mit.massif.simulink/model

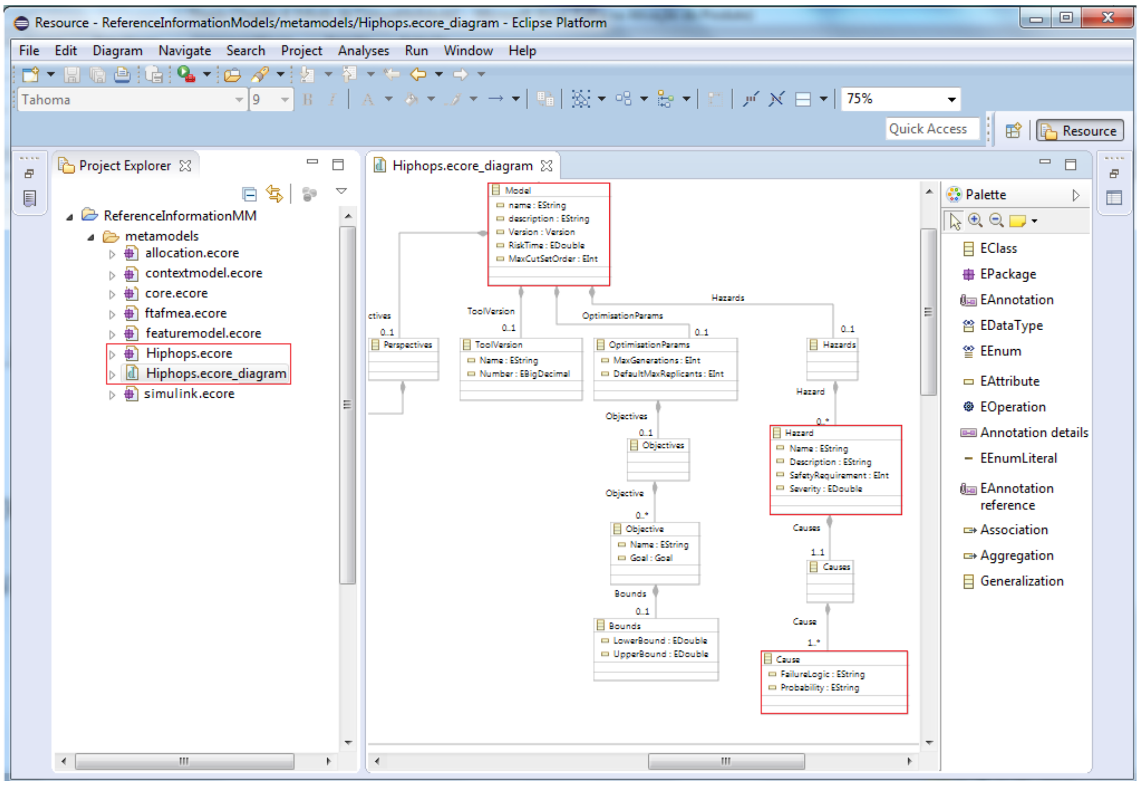


Figure 6.9. Failure metamodel in the EMF platform.

6.3.3 SC-PLE Phase 12: HBS-SPL Weaving Modeling

Assurance case patterns (Section 6.3.1), and product line asset metamodels (Section 6.3.2) are input artefacts to define mapping links between assurance case pattern abstract "*terms*" and "*system model elements*" that provide the required information to instantiate them. Assuming the automotive and aerospace product lines used throughout this thesis, the following asset metamodels were considered: AADL or Simulink, feature, context, failure, and fault tree and FMEA metamodels.

Figure 6.10 shows an excerpt of the created weaving model to support the instantiation of the assurance case patterns defined in Section 6.3.1. A UML class diagram is used to illustrate the weaving model. This model was created with the support of yEd graph modeling tool (YWORKS, 2016), and a weaving metamodel that extends the standard weaving metamodel (DEL FABRO et al. 2005). Firstly, abstract *"terms"* from assurance case patterns are defined in the left-hand side of the model. The abstract terms to be instantiated are: "*operationalEnvironment*", "*systemDefinition*", "*system*", "*components*", "*CSF*", "*targetSafetyStandard*", "*hazard*", "*SIL*", "*silAllocatedToHzds*", "*failureCondition*", and "*HSFMType*". In the following, from the analysis of the created "*terms*", and the information

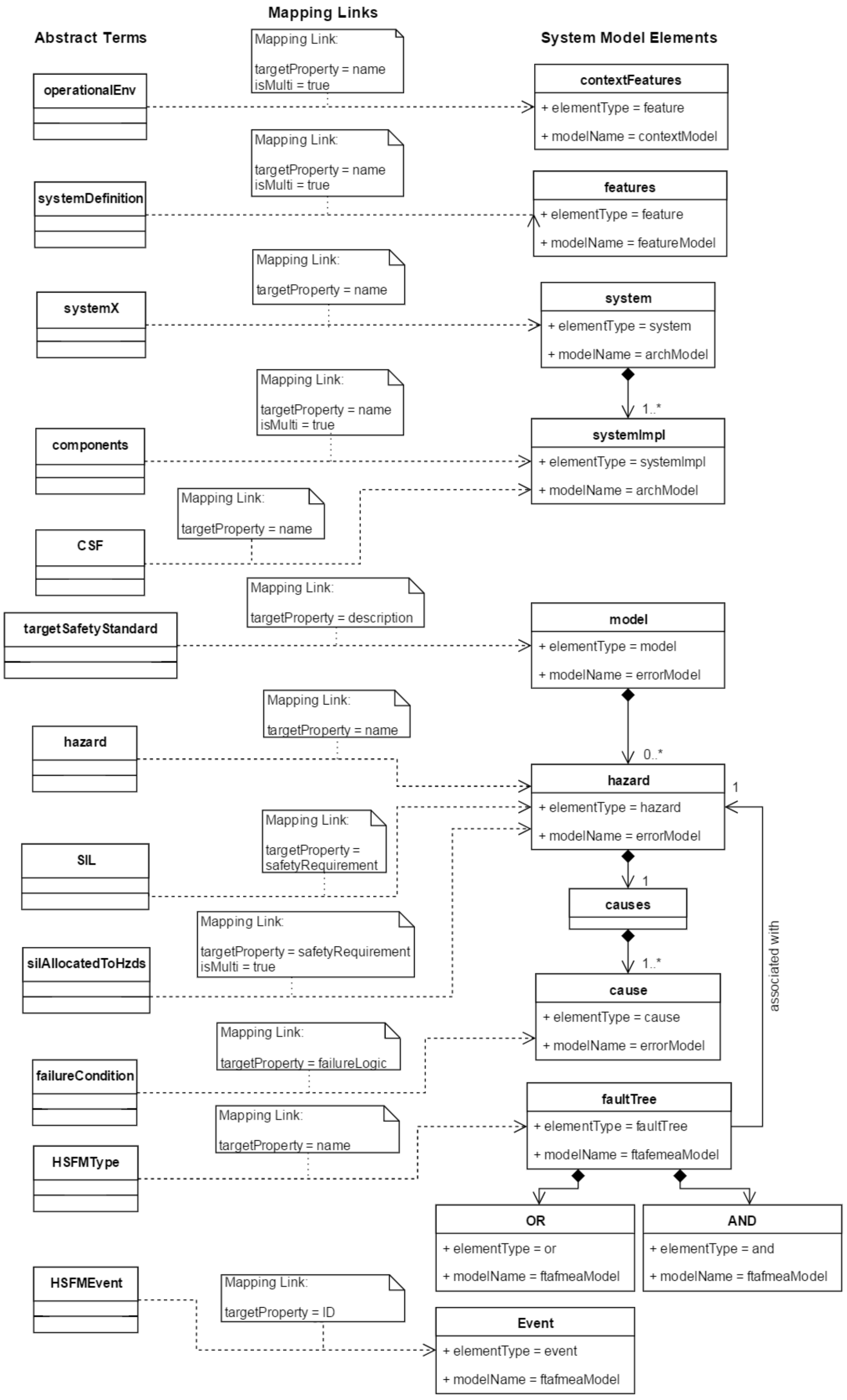

Mapping Links
Abstract Terms
System Model Elements
operationalEnv
systemDefinition
systemX
components
CSF
targetSafetyStandard
hazard
SIL
silAllocatedToHzds
failureCondition
HSFMType
HSFMEvent
Mapping Link: targetProperty = name isMulti = true
Mapping Link: targetProperty = name isMulti = true
Mapping Link: targetProperty = name
Mapping Link: targetProperty = name isMulti = true
Mapping Link: targetProperty = name
Mapping Link: targetProperty = description
Mapping Link: targetProperty = name
Mapping Link: targetProperty = safetyRequirement
Mapping Link: targetProperty = safetyRequirement isMulti = true
Mapping Link: targetProperty = failureLogic
Mapping Link: targetProperty = name
Mapping Link: targetProperty = ID
contextFeatures + elementType = feature + modelName = contextModel
features + elementType = feature + modelName = featureModel
system + elementType = system + modelName = archModel
1..*
systemImpl + elementType = systemImpl + modelName = archModel
model + elementType = model + modelName = errorModel
0..*
hazard + elementType = hazard + modelName = errorModel
1
1
causes
1..*
cause + elementType = cause + modelName = errorModel
associated with
faultTree + elementType = faultTree + modelName = ftafemeaModel
OR + elementType = or + modelName = ftafmeaModel
AND + elementType = and + modelName = ftafmeaModel
Event + elementType = event + modelName = ftafmeaModel

Figure 6.10. Excerpt of the weaving model.

provided by product line asset metamodels, references to system model elements that provide the required information to instantiate each abstract "term" were defined. For example, the "*contextFeatures*" element from the *context* metamodel contains the required information to instantiate the "*operationalEnv"* abstract "*term*" associated with *Hazard Avoidance* pattern shown in Figure 6.7. The *"elementType"* attribute value from "*contextFeatures"* element determines that the "*feature"* model element provides the required information to instantiate the "*operationalEnv"* abstract "*term"*. The relationship between these elements is defined via mapping links indicated by dashed arrows connecting the left-hand side elements to right-hand side elements.

Mapping links also allow specifying properties for referencing attributes from a given "*system model element"*, which provides the required information for instantiating a given abstract "*term"*. For example, the mapping link connecting the "*operationalEnv*" abstract "*term"* to the "*contextFeature*" element indicates that the "*name*" attribute from the "*feature"* model element provides the required information to instantiate the "*operationalEnv"* abstract "*term"*. Additionally, it is also possible to define whether multiple occurrences of a given "*system model element"* are required to instantiate a single occurrence of an abstract "*term"*, via "*isMulti" Boolean* property set with a *"true"* value. For example, for instantiating the *"operationalEnv"* abstract "*term*" associated with "*C2*" context element in "*Hazard Avoidance"* pattern (see Figure 6.7), the *"name*" of each context feature is required. Therefore, the instantiation of "*operationalEnv"* abstract *term* is a list of "*feature.name*" information elements separated by commas, e.g., "*feature$_1$.name*", "*feature$_2$.name*",…, "*feature$_n$.name"*. Considering a given HBS-SPL instance, the instantiation of *"operationalEnv" "term"* requires "*Four Wheels", "OffRoad", "Rally"* feature names provided by the context model, as illustrated in Figure 6.11.

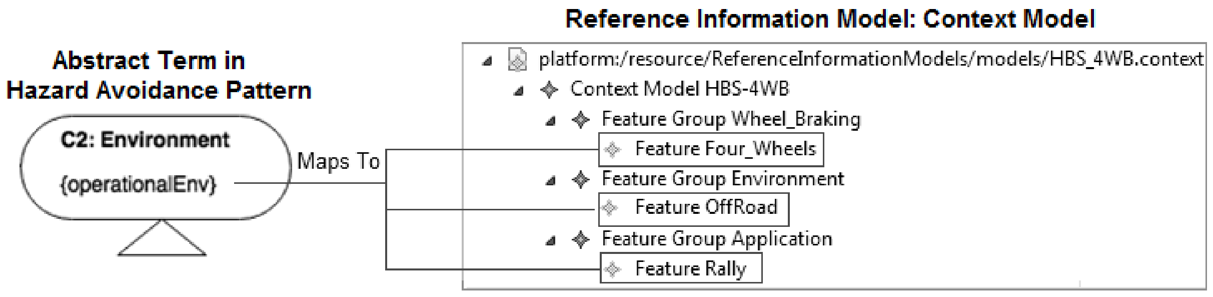
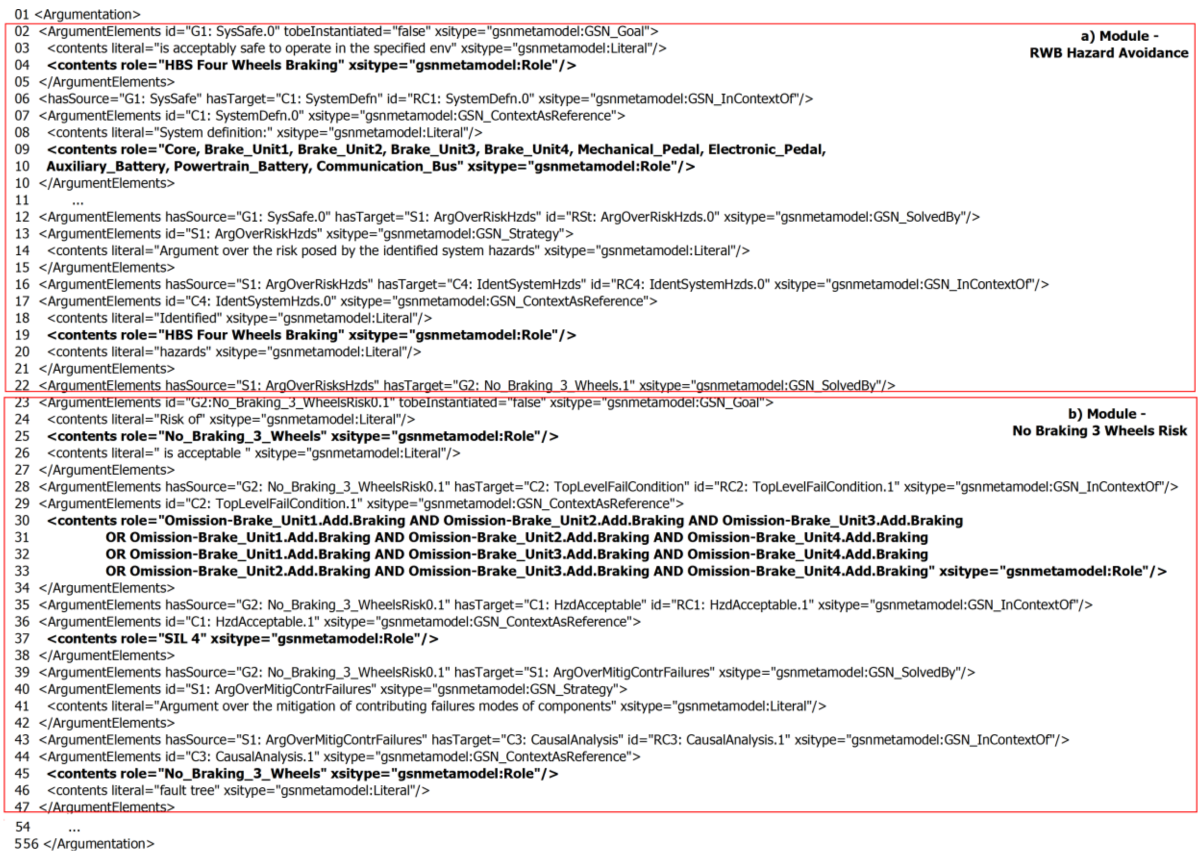

Abstract Term in Hazard Avoidance Pattern
C2: Environment
{operationalEnv}
Maps To
Reference Information Model: Context Model
platform:/resource/ReferenceInformationModels/models/HBS_4WB.context
Context Model HBS-4WB
Feature Group Wheel_Braking
Feature Four_Wheels
Feature Group Environment
Feature OffRoad
Feature Group Application
Feature Rally

Figure 6.11. Mapping links between abstract terms and system model elements.

6.3.4 SC-PLE Phase 13: HBS-SPL Assurance Case Model Generation

The assurance case patterns, reference information metamodels, and the weaving model created in domain engineering were reused in application engineering for generating assurance case models for three HBS-SPL instances with the support of the MBAC EOL instantiation program (see Section 2.4.3.4). Assurance cases were generated for the following HBS-SPL instances: four wheel braking (4WB), front wheel braking (FWB), and rear wheel braking (RWB). Appendix D shows the generated assurance cases for instances of the Tiriba Flight Control aerospace product line.

The MBAC tool generated HBS-SPL variant-specific assurance cases in “*.gsnml”* and tabular formats. The “*gsnml”* file contains the specification of the instantiated assurance case pattern elements and their relationships, and the tabular format shows the relationships between assurance case pattern abstract *terms* and their corresponding instantiation information in an instantiation table. Figure 6.12 shows an excerpt of the *“.gsnml*” representation of the assurance case generated for the 4WB product line instance. The highlighted text represents abstract *terms* instantiated with information from a diverse set of


a) Module -
RWB Hazard Avoidance
b) Module -
No Braking 3 Wheels Risk

```
<Argumentation>
<ArgumentElements id="G1: SysSafe.0" tobeInstantiated="false" xsitype="gsnmetamodel:GSN_Goal">
  <contents literal="is acceptably safe to operate in the specified env" xsitype="gsnmetamodel:Literal"/>
  <contents role="HBS Four Wheels Braking" xsitype="gsnmetamodel:Role"/>
</ArgumentElements>
<hasSource="G1: SysSafe" hasTarget="C1: SystemDefn" id="RC1: SystemDefn.0" xsitype="gsnmetamodel:GSN_InContextOf"/>
<ArgumentElements id="C1: SystemDefn.0" xsitype="gsnmetamodel:GSN_ContextAsReference">
  <contents literal="System definition:" xsitype="gsnmetamodel:Literal"/>
  <contents role="Core, Brake_Unit1, Brake_Unit2, Brake_Unit3, Brake_Unit4, Mechanical_Pedal, Electronic_Pedal,
  Auxiliary_Battery, Powertrain_Battery, Communication_Bus" xsitype="gsnmetamodel:Role"/>
</ArgumentElements>
      ...
<ArgumentElements hasSource="G1: SysSafe.0" hasTarget="S1: ArgOverRiskHzds" id="RSt: ArgOverRiskHzds.0" xsitype="gsnmetamodel:GSN_SolvedBy"/>
<ArgumentElements id="S1: ArgOverRiskHzds" xsitype="gsnmetamodel:GSN_Strategy">
  <contents literal="Argument over the risk posed by the identified system hazards" xsitype="gsnmetamodel:Literal"/>
</ArgumentElements>
<ArgumentElements hasSource="S1: ArgOverRiskHzds" hasTarget="C4: IdentSystemHzds" id="RC4: IdentSystemHzds.0" xsitype="gsnmetamodel:GSN_InContextOf"/>
<ArgumentElements id="C4: IdentSystemHzds.0" xsitype="gsnmetamodel:GSN_ContextAsReference">
  <contents literal="Identified" xsitype="gsnmetamodel:Literal"/>
  <contents role="HBS Four Wheels Braking" xsitype="gsnmetamodel:Role"/>
  <contents literal="hazards" xsitype="gsnmetamodel:Literal"/>
</ArgumentElements>
<ArgumentElements hasSource="S1: ArgOverRisksHzds" hasTarget="G2: No_Braking_3_Wheels.1" xsitype="gsnmetamodel:GSN_SolvedBy"/>
<ArgumentElements id="G2:No_Braking_3_WheelsRisk0.1" tobeInstantiated="false" xsitype="gsnmetamodel:GSN_Goal">
  <contents literal="Risk of" xsitype="gsnmetamodel:Literal"/>
  <contents role="No_Braking_3_Wheels" xsitype="gsnmetamodel:Role"/>
  <contents literal=" is acceptable " xsitype="gsnmetamodel:Literal"/>
</ArgumentElements>
<ArgumentElements hasSource="G2: No_Braking_3_WheelsRisk0.1" hasTarget="C2: TopLevelFailCondition" id="RC2: TopLevelFailCondition.1" xsitype="gsnmetamodel:GSN_InContextOf"/>
<ArgumentElements id="C2: TopLevelFailCondition.1" xsitype="gsnmetamodel:GSN_ContextAsReference">
  <contents role="Omission-Brake_Unit1.Add.Braking AND Omission-Brake_Unit2.Add.Braking AND Omission-Brake_Unit3.Add.Braking
         OR Omission-Brake_Unit1.Add.Braking AND Omission-Brake_Unit2.Add.Braking AND Omission-Brake_Unit4.Add.Braking
         OR Omission-Brake_Unit1.Add.Braking AND Omission-Brake_Unit3.Add.Braking AND Omission-Brake_Unit4.Add.Braking
         OR Omission-Brake_Unit2.Add.Braking AND Omission-Brake_Unit3.Add.Braking AND Omission-Brake_Unit4.Add.Braking" xsitype="gsnmetamodel:Role"/>
</ArgumentElements>
<ArgumentElements hasSource="G2: No_Braking_3_WheelsRisk0.1" hasTarget="C1: HzdAcceptable" id="RC1: HzdAcceptable.1" xsitype="gsnmetamodel:GSN_InContextOf"/>
<ArgumentElements id="C1: HzdAcceptable.1" xsitype="gsnmetamodel:GSN_ContextAsReference">
  <contents role="SIL 4" xsitype="gsnmetamodel:Role"/>
</ArgumentElements>
<ArgumentElements hasSource="G2: No_Braking_3_WheelsRisk0.1" hasTarget="S1: ArgOverMitigContrFailures" xsitype="gsnmetamodel:GSN_SolvedBy"/>
<ArgumentElements id="S1: ArgOverMitigContrFailures" xsitype="gsnmetamodel:GSN_Strategy">
  <contents literal="Argument over the mitigation of contributing failures modes of components" xsitype="gsnmetamodel:Literal"/>
</ArgumentElements>
<ArgumentElements hasSource="S1: ArgOverMitigContrFailures" hasTarget="C3: CausalAnalysis" id="RC3: CausalAnalysis.1" xsitype="gsnmetamodel:GSN_InContextOf"/>
<ArgumentElements id="C3: CausalAnalysis.1" xsitype="gsnmetamodel:GSN_ContextAsReference">
  <contents role="No_Braking_3_Wheels" xsitype="gsnmetamodel:Role"/>
  <contents literal="fault tree" xsitype="gsnmetamodel:Literal"/>
</ArgumentElements>
      ...
</Argumentation>
```

Figure 6.12. 4WB assurance case *gsnml* representation.

system models. For example, the "*HBS Rear Wheels Braking*" information, provided by the architecture model, instantiates the "*system*" abstract term associated with "*G1*" goal in line 4. Figure 6.12a shows the instantiation of *Hazard Avoidance* pattern. Figure 6.12b illustrates the instantiation of the "*Risk Argument"* pattern for "*No braking three wheels"* system hazard (lines 23-47). For each "*4WB*" related hazard, an argument module arguing that the "*risk posed by each hazard is acceptable*" has been generated. This claim is further decomposed into fault mitigation claims arguing that all causes of each contributing failure mode are acceptable.

Whereas context and design variation directly impact in the variant-specific architecture and safety assessment models, which are inputs to MBAC generating an assurance case for a given variant, such variation is propagated throughout the assurance case. Figure 6.13 shows excerpts of modular views of the assurance cases generated for *"4WB"* and *"FWB"* hybrid braking system variants. The generated assurance case models for these variants share the structure defined in the assurance case patterns (Section 6.3.1). Therefore, a variant-specific assurance case comprises a top-level argument module, arguing that the system variant is acceptably safe, which is further decomposed into modules arguing that the risks posed by the associated hazards are acceptable. Each *"Risk Argument"* module is further decomposed into modules arguing the absence of each contributing component faults. Finally, "*Absence of Hazardous Software Failure Mode"* modules argue that primary, secondary, and control faults, which can cause a given hazardous software failure mode, are acceptable. An "*Absence of Hazardous Software Failure Mode"* module is further supported by other fault mitigation claims arguing that the causes of failures in other components contributing to hazardous software failure mode are acceptable, and optionally, into other "*Absence of Hazardous Software Failure Mode"* modules.

Variation in architecture and safety models are reflected in the number of risk and fault mitigation argument modules in *"4WB"* and *"FWB"* variant-specific assurance cases. The 4WB assurance case comprises risk argument modules for *"Value Braking"*, *"No Braking Rear"*, *"No Braking Diagonal"*, "*No Braking Front"*, *"No Braking Four Wheels"*, and *"No Braking Three Wheels"* hazards (Figure 6.13a), whereas the "*FWB"* assurance case comprises argument modules for *"No Braking Front",* and *"Value Braking"* hazards (Figure 6.13b). Although the assurance cases for these variants having common risk argument modules, variability has also been found in "*Absence of Hazardous Software Failure Mode*"

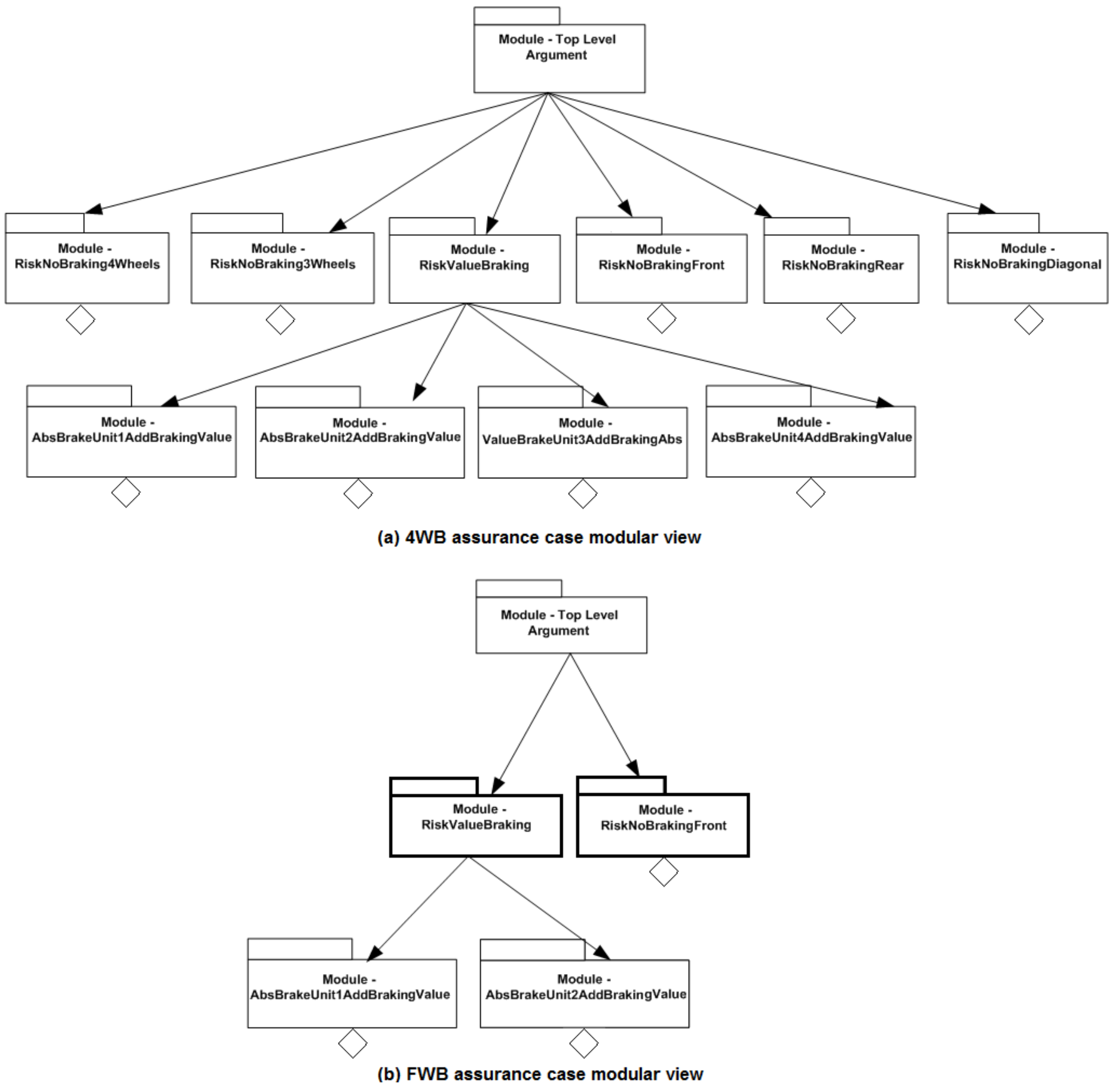

Module - Top Level Argument
Module - RiskNoBraking4Wheels
Module - RiskNoBraking3Wheels
Module - RiskValueBraking
Module - RiskNoBrakingFront
Module - RiskNoBrakingRear
Module - RiskNoBrakingDiagonal
Module - AbsBrakeUnit1AddBrakingValue
Module - AbsBrakeUnit2AddBrakingValue
Module - ValueBrakeUnit3AddBrakingAbs
Module - AbsBrakeUnit4AddBrakingValue
(a) 4WB assurance case modular view
Module - Top Level Argument
Module - RiskValueBraking
Module - RiskNoBrakingFront
Module - AbsBrakeUnit1AddBrakingValue
Module - AbsBrakeUnit2AddBrakingValue
(b) FWB assurance case modular view

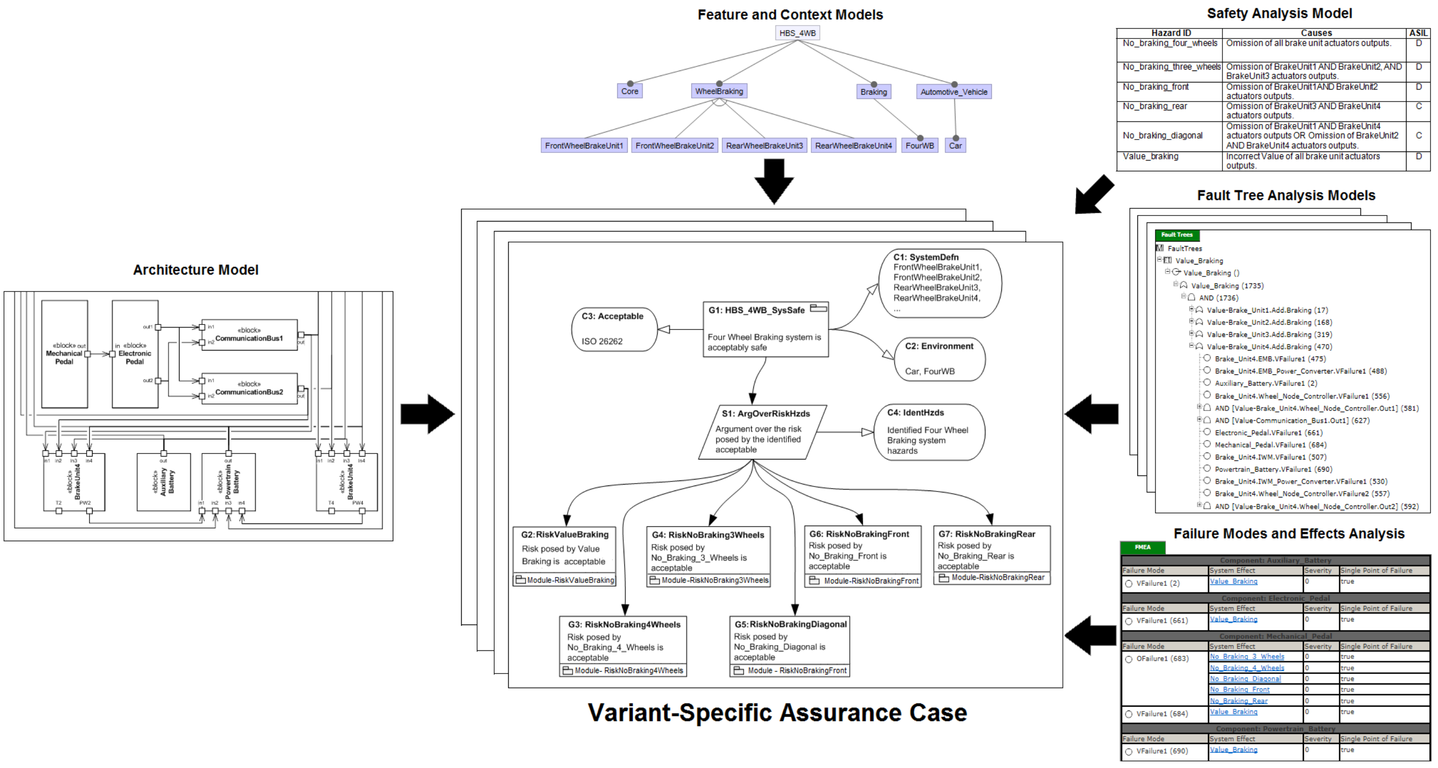
Figure 6.13.HBS-SPL variant-specific assurance cases.

modules. For example, the supporting modules for "*RiskValueBraking*" argument module are "*AbsBrakeUnit1AddBrakingValue*" and *"AbsBrakeUnit2AddBrakingAbs"* in the *"FWB"* variant. On the other hand, the "*RiskValueBraking*" module is supported by "*AbsBrakeUnit1AddBrakingValue*", "*AbsBrakeUnit2AddBrakingValue*", "*AbsBrakeUnit3-AddBrakingValue*", and "*AbsBrakeUnit4AddBrakingValue*" modules in *"4WB"* variant-specific assurance case. Such variation is further propagated throughout other "*Absence of Hazardous Software Failure Mode"* modules and their supporting modules. The following subsections detail the "*Top-Level Argument"*, "*Risk Argument"*, and *"Absence of Hazardous Software Failure Mode"* modules generated for the *"4WB"* variant shown in Figure 6.13a.

6.3.4.1 HBS-4WB Top-Level Argument

The "*Top-Level Argument"* generated for the four wheel braking system variant, shown in Figure 6.14, is a hazard and risk-directed module. This module decomposes the claim arguing that the four wheel braking system variant is acceptably safe to operate in its environment, into sub-claims ("*G2", "G3", "G4", "G5", "G6",* and "*G7"*) arguing that the risk posed by each variant-specific hazard is acceptable. Each risk argument is encapsulated in a separated module that justifies the absence of component failures that can cause a given hazard, i.e., do not lead the system variant to an unsafe state. Such justification is defined in the context of the ASIL allocated to each system hazard stated in "*C3"* element.

The *"4WB"* assurance case shown in Figure 6.14 was generated from the information provided by different design and safety assessment models. For example, the information provided by feature and context models was used to instantiate *"C1"* and *"C2"* context elements. The architecture model was used to instantiate *"G1"* and *"C4"* elements. Finally, safety analysis, fault trees, and FMEA results were used to respectively instantiating "*Risk Argument*" and "*Absence of Hazardous Software Failure Modes*" modules. Therefore, changes in the design and context are automatically propagated throughout these assets and the assurance case.


Feature and Context Models
HBS_4WB
Core
WheelBraking
Braking
Automotive_Vehicle
FrontWheelBrakeUnit1
FrontWheelBrakeUnit2
RearWheelBrakeUnit3
RearWheelBrakeUnit4
FourWB
Car
Architecture Model
«block» Mechanical Pedal
out
in
«block» Electronic Pedal
out1
out2
in1
in2
«block» CommunicationBus1
out
«block» CommunicationBus2
«block» BrakeUnit4
in1
in2
in3
in4
T2
PW2
«block» Auxiliary Battery
«block» Powertrain Battery
T4
PW4
C1: SystemDefn
FrontWheelBrakeUnit1, FrontWheelBrakeUnit2, RearWheelBrakeUnit3, RearWheelBrakeUnit4, ...
C3: Acceptable
ISO 26262
G1: HBS_4WB_SysSafe
Four Wheel Braking system is acceptably safe
C2: Environment
Car, FourWB
S1: ArgOverRiskHzds
Argument over the risk posed by the identified acceptable
C4: IdentHzds
Identified Four Wheel Braking system hazards
G2: RiskValueBraking
Risk posed by Value Braking is acceptable
Module-RiskValueBraking
G4: RiskNoBraking3Wheels
Risk posed by No_Braking_3_Wheels is acceptable
Module-RiskNoBraking3Wheels
G6: RiskNoBrakingFront
Risk posed by No_Braking_Front is acceptable
Module-RiskNoBrakingFront
G7: RiskNoBrakingRear
Risk posed by No_Braking_Rear is acceptable
Module-RiskNoBrakingRear
G3: RiskNoBraking4Wheels
Risk posed by No_Braking_4_Wheels is acceptable
Module- RiskNoBraking4Wheels
G5: RiskNoBrakingDiagonal
Risk posed by No_Braking_Diagonal is acceptable
Module - RiskNoBrakingFront
Variant-Specific Assurance Case
Safety Analysis Model
Fault Tree Analysis Models
Fault Trees
Failure Modes and Effects Analysis
FMEA

Hazard ID	Causes	ASIL
No_braking_four_wheels	Omission of all brake unit actuators outputs.	D
No_braking_three_wheels	Omission of BrakeUnit1 AND BrakeUnit2, AND BrakeUnit3 actuators outputs.	D
No_braking_front	Omission of BrakeUnit1AND BrakeUnit2 actuators outputs.	D
No_braking_rear	Omission of BrakeUnit3 AND BrakeUnit4 actuators outputs.	C
No_braking_diagonal	Omission of BrakeUnit1 AND BrakeUnit4 actuators outputs OR Omission of BrakeUnit2 AND BrakeUnit4 actuators outputs.	C
Value_braking	Incorrect Value of all brake unit actuators outputs.	D

```
FaultTrees
 Value_Braking
  Value_Braking ()
   Value_Braking (1735)
    AND (1736)
     Value-Brake_Unit1.Add.Braking (17)
     Value-Brake_Unit2.Add.Braking (168)
     Value-Brake_Unit3.Add.Braking (319)
     Value-Brake_Unit4.Add.Braking (470)
      Brake_Unit4.EMB.VFailure1 (475)
      Brake_Unit4.EMB_Power_Converter.VFailure1 (488)
      Auxiliary_Battery.VFailure1 (2)
      Brake_Unit4.Wheel_Node_Controller.VFailure1 (556)
      AND [Value-Brake_Unit4.Wheel_Node_Controller.Out1] (581)
      AND [Value-Communication_Bus1.Out1] (627)
      Electronic_Pedal.VFailure1 (661)
      Mechanical_Pedal.VFailure1 (684)
      Brake_Unit4.IWM.VFailure1 (507)
      Powertrain_Battery.VFailure1 (690)
      Brake_Unit4.IWM_Power_Converter.VFailure1 (530)
      Brake_Unit4.Wheel_Node_Controller.VFailure2 (557)
      AND [Value-Brake_Unit4.Wheel_Node_Controller.Out2] (592)
```

Failure Mode	System Effect	Severity	Single Point of Failure
Component: Auxiliary_Battery			
VFailure1 (2)	Value_Braking	0	true
Component: Electronic_Pedal			
VFailure1 (661)	Value_Braking	0	true
Component: Mechanical_Pedal			
OFailure1 (683)	No_Braking_3_Wheels	0	true
	No_Braking_4_Wheels	0	true
	No_Braking_Diagonal	0	true
	No_Braking_Front	0	true
	No_Braking_Rear	0	true
VFailure1 (684)	Value_Braking	0	true
Component: Powertrain_Battery			
VFailure1 (690)	Value_Braking	0	true

Figure 6.14. Four wheel braking top-level argument module.

6.3.4.2 HBS-4WB Risk and Absence of Hazardous Failure Mode Argument Modules

The *"4WB"* top-level argument module is supported by six risk argument modules arguing that the risk posed by each associated system hazard is acceptable. Each risk module is supported by sub-claims arguing the absence of each contributing *hazardous software failure mode*. Figure 6.15 shows the "*RiskValueBraking"* argument module. "*G2"* top- level claim is stated in the context of ASIL *"D"* allocated to "*Value Braking"* system hazard (*C5*), and the top-level failure condition leading to this hazard (*C6*). This claim is decomposed into sub-claims arguing the mitigation of component failures that directly contribute to the occurrence of "*Value Braking"* hazard ("*S1"*), in this case, incorrect values in "*Brake_Unit1.Add" ("G8"), "Brake_Unit2.Add" ("G9"), "Brake_Unit3.Add" ("G10"),* and

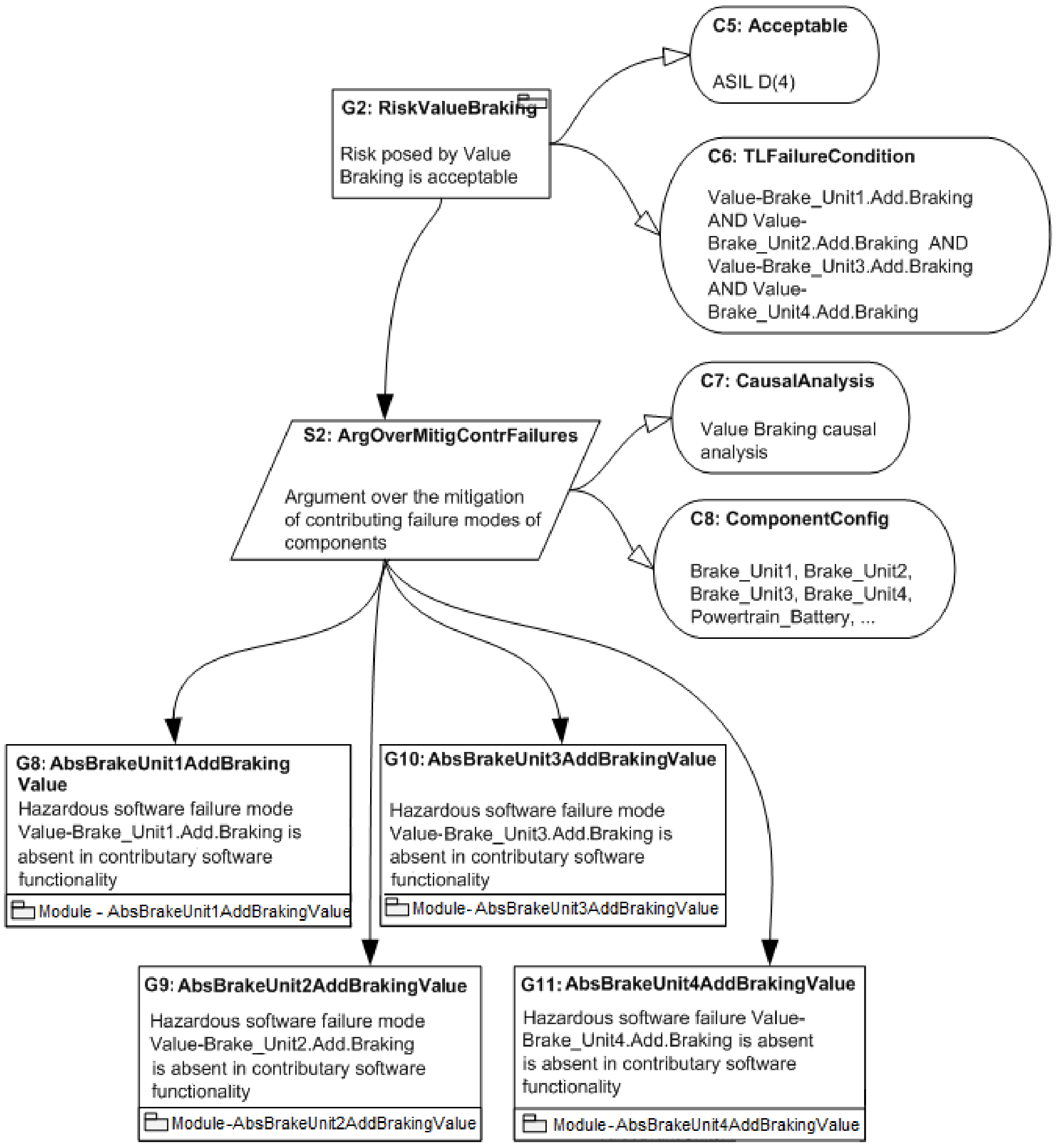

C5: Acceptable
ASIL D(4)
G2: RiskValueBraking
Risk posed by Value Braking is acceptable
C6: TLFailureCondition
Value-Brake_Unit1.Add.Braking AND Value-Brake_Unit2.Add.Braking AND Value-Brake_Unit3.Add.Braking AND Value-Brake_Unit4.Add.Braking
C7: CausalAnalysis
Value Braking causal analysis
S2: ArgOverMitigContrFailures
Argument over the mitigation of contributing failure modes of components
C8: ComponentConfig
Brake_Unit1, Brake_Unit2, Brake_Unit3, Brake_Unit4, Powertrain_Battery, ...
G8: AbsBrakeUnit1AddBraking Value
Hazardous software failure mode Value-Brake_Unit1.Add.Braking is absent in contributary software functionality
Module - AbsBrakeUnit1AddBrakingValue
G10: AbsBrakeUnit3AddBrakingValue
Hazardous software failure mode Value-Brake_Unit3.Add.Braking is absent in contributary software functionality
Module- AbsBrakeUnit3AddBrakingValue
G9: AbsBrakeUnit2AddBrakingValue
Hazardous software failure mode Value-Brake_Unit2.Add.Braking is absent in contributary software functionality
Module-AbsBrakeUnit2AddBrakingValue
G11: AbsBrakeUnit4AddBrakingValue
Hazardous software failure Value-Brake_Unit4.Add.Braking is absent is absent in contributary software functionality
Module - AbsBrakeUnit4AddBrakingValue

Figure 6.15. Value braking risk argument module.

"*Brake_Unit4.Add" ("G11")* component outputs, which can cause an incorrect braking torque while braking. Such decomposition strategy is defined in the context of the causal chain defined in the "*Value Braking*" fault tree, and *"4WB"* variant specific components defined in the architecture model. "*G8"*, "*G9", "G10",* and "*G11"* are references to "*AbsBrakeUnit1AddBrakingValue", "AbsBrakeUnit2AddBrakingValue", "AbsBrakeUnit3AddBrakingValue",* and *"AbsBrakeUnit4AddBrakingValue"* modules, respectively. These modules decompose claims over the absence of value failures in all brake unit outputs into sub-claims arguing that the occurrence of primary, secondary, and control failure modes that can cause these component failures are acceptable.

Figure 6.16 illustrates the "*AbsBrakeUnit3AddBrakingValue"* argument module. This module decomposes the claim "*G10"* into sub-claims arguing that: an internal failure in "*Brake_Unit3.Add"* component is acceptable ("*G10.1"*), the failure modes of other components that contribute to incorrect value of *"Brake_Unit3.Add.Braking"* output port are

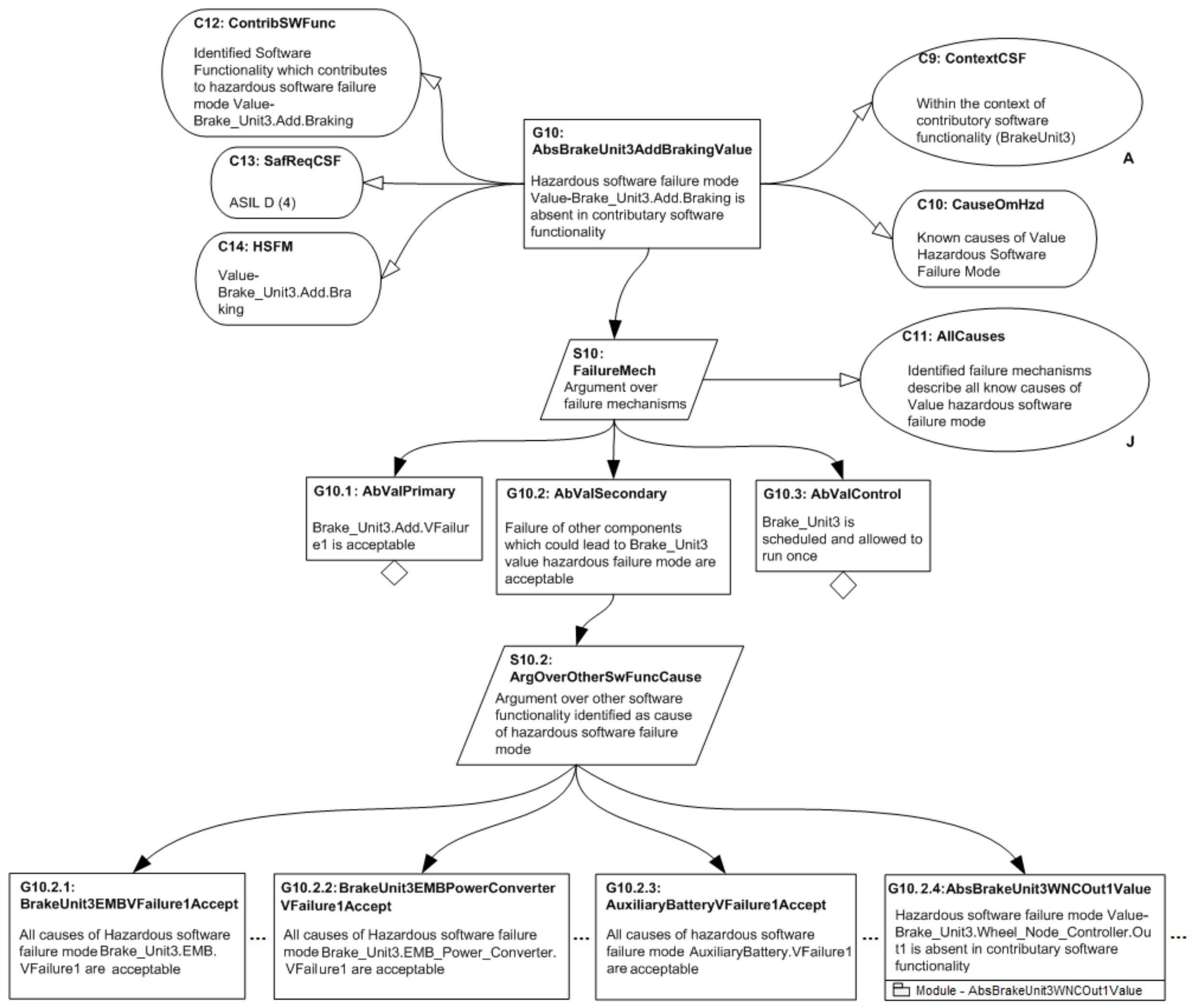

C12: ContribSWFunc
Identified Software Functionality which contributes to hazardous software failure mode Value-Brake_Unit3.Add.Braking
C13: SafReqCSF
ASIL D (4)
C14: HSFM
Value-Brake_Unit3.Add.Braking
G10: AbsBrakeUnit3AddBrakingValue
Hazardous software failure mode Value-Brake_Unit3.Add.Braking is absent in contributary software functionality
C9: ContextCSF
Within the context of contributory software functionality (BrakeUnit3)
A
C10: CauseOmHzd
Known causes of Value Hazardous Software Failure Mode
S10: FailureMech
Argument over failure mechanisms
C11: AllCauses
Identified failure mechanisms describe all know causes of Value hazardous software failure mode
J
G10.1: AbValPrimary
Brake_Unit3.Add.VFailure1 is acceptable
G10.2: AbValSecondary
Failure of other components which could lead to Brake_Unit3 value hazardous failure mode are acceptable
G10.3: AbValControl
Brake_Unit3 is scheduled and allowed to run once
S10.2: ArgOverOtherSwFuncCause
Argument over other software functionality identified as cause of hazardous software failure mode
G10.2.1: BrakeUnit3EMBVFailure1Accept
All causes of Hazardous software failure mode Brake_Unit3.EMB.VFailure1 are acceptable
G10.2.2: BrakeUnit3EMBPowerConverterVFailure1Accept
All causes of Hazardous software failure mode Brake_Unit3.EMB_Power_Converter.VFailure1 are acceptable
G10.2.3: AuxiliaryBatteryVFailure1Accept
All causes of hazardous software failure mode AuxiliaryBattery.VFailure1 are acceptable
G10.2.4: AbsBrakeUnit3WNCOut1Value
Hazardous software failure mode Value-Brake_Unit3.Wheel_Node_Controller.Out1 is absent in contributary software functionality
Module - AbsBrakeUnit3WNCOut1Value
...
...
...
...

Figure 6.16. Absence of value Brake_Unit3.Add.Braking argument module.

acceptable (“*G10.2”*), and that “*Brake_Unit3.Add”* component is scheduled and allowed to run once” (“*G10.3”*). “*G10.2”* is further decomposed into fault mitigation sub-claims arguing that incorrect values in *“Brake_Unit3EMB” (“G10.2.1”), “Brake_Unit3.EMB_ Power_Converter” (“G10.2.2”), and “Auxiliary_Battery” (“G10.2.3”)* components, and other components, are acceptable. These claims argue that all causes of each failure event specified in fault tree leaf nodes do not lead the system to an unsafe state. The claim *“G10.2”* is also decomposed into other “*Absence Hazardous Software Failure Mode*” fault mitigation argument modules, e.g., the *“G10.2.4”* argument module. An “*Absence Hazardous Software Failure Mode*” fault mitigation module argues that the occurrence of primary, secondary, and control failure modes of a given fault tree gate, e.g., AND/OR gates, do not lead the system to an unsafe state.

Table 6.1 illustrates the relationship between “*Top-Level Argument”* elements, and information elements from different system models. For example, the “*Four Wheel Braking system*” information stated in “*G1*” and “*C4*” elements in Figure 6.14 is provided by *“4WB”* architecture model. Hazard analysis information elements stated in “*G2”, “G3”, “G4”, “G5”, “G6”,* and *“G7”* module references are provided by *“4WB”* safety analysis model. Therefore, changes in the system variant are automatically propagated throughout the assurance case.

In order to assess the feasibility of the MBAC approach to support the automated generation of assurance cases in software product line engineering, from the perspective of product line application engineers and non-safety professionals, an experimental study was carried out. The following section presents the planning, execution, and the results of this experimental study.

Table 6.1 – Assurance case pattern instantiation and system model elements.

Assurance Case Model	***Abstract Term***	***Information Element***	***Model Element***	***Source System Model***
Top-Level Argument	systemX	Four Wheel Braking system	Model.name	Simulink model
	systemDefinition	Brake_Unit1, Brake_Unit2, Brake_Unit3, Brake_Unit4, Auxiliary_Battery, …	Feature.name	Feature model
	Environment	Car, Four Wheels	ContextFeature.name	Context model
	targetSafetyStandard	ISO 26262	Model.description	Failure model
	Hazard	Value Braking, No Braking Four Wheels, No Braking Three Wheels, No Braking Diagonal, No Braking Rear, No Braking Front	Hazard.name	Failure Model

6.4 Experimental Study

This section presents the results of an experimental study carried out to evaluate the effectiveness, in terms of quality, and the efficiency, in terms of productivity, of applying the MBAC (HAWKINS *et al.* 2015) approach and tooling to support the automatic generation of assurance cases in software product line engineering. The "*time*" metric is used to assess the productivity in terms of the time spent using both the Conventional (C) manual process of assurance case construction, and the MBAC (M) to generate an assurance case for a targeted product line instance. The *"error rate"* metric is used to evaluate the quality in terms of the number of errors found in the generated assurance case, e.g., wrong information used to instantiate assurance case pattern elements. These metrics have been chosen to assess the "*Efficiency*" and "*Effectiveness*" of both assurance case construction approaches in product line engineering. This experimental study has been defined and executed according to guidelines established in Empirical Software Engineering (WOHLIN *et al.* 2000). The following sections present the study definition, planning, operation, data analysis interpretation, and hypothesis testing.

6.4.1 Study Definition

6.4.1.1 Objetives

The primary objective of this experimental study is to compare the effort in designing assurance case models for a safety-critical system variant using the Conventional manual process (C) and the MBAC (M) approach. The secondary objective of this study is to compare the quality of the generated assurance case models using both approaches.

The objective statement in the form prescribed by Wohlin *et al.* (2000) is detailed in the following:

Analyze the Conventional (C) and MBAC (M) approaches

For the purpose of evaluating the impact of both approaches in assurance case construction in terms of effort and quality

With respect to their Efficiency and Effectiveness

From the point of view of Safety Engineers

In the context of Safety-Critical Software Product Line Engineering Processes.

6.4.1.2 Experimental Objects

The experimental objects of this study are: the Hazard Avoidance assurance case pattern, and system models from four wheel braking variant from a braking system product line (HBS-4WB), and all pilot modes variant from Tiriba Flight Control product line (TFC-ALL), used to create assurance case models for these variants using conventional and model-based approaches.

6.4.1.3 Quantitative Focus

It involves the identification of effort in terms of "*time*" spent by each participant to conclude the design of an assurance case model for a given product variant, and the quantification of quality in terms of the *error rate* identified in the generated assurance case.

6.4.1.4 Qualitative Focus

It is intended to identify the technique that requires less effort to design a variant-specific assurance case model, and the technique that generates an assurance case model with fewer errors.

6.4.1.5 Perspective

The experiment was conducted from the perspective of safety engineers concerned in designing an assurance case that provides the justification that a given product variant is acceptably safe in its environment to achieve goal-based certification.

6.4.1.6 Study Objects

The "*effort*" to generate an assurance case and the "*quality*" of the generated artefact are the study objects.

6.4.2 Planning

This experimental study was planned by considering the following questions:

- **Q1:** What assurance case construction approach (manual or model-based) requires less effort to design an assurance case for a targeted product variant?

- **Q2:** What assurance case construction approach generates a higher quality assurance case for a targeted product variant?

These two questions guided the data analysis to evaluate which assurance case construction approach is more efficient and effective.

6.4.2.1 Participant Selection Criteria

The participants were selected through a non-probabilistic approach by convenience, i.e., the probability of all population elements belonging to the same sample is unknown. MSc. and Ph.D. students in Software Engineering from the Institute of Mathematics and Computer Science from the University of São Paulo and Federal University of São Paulo were invited.

6.4.2.2 Context Selection

This experimental study was performed by Software Engineering post-graduate students from Institute of Mathematics and Computer Science of the University of São Paulo (ICMC-USP) and Federal University of São Carlos, referred here as “participants”. Twenty four participants performed the experiment, from which fourteen are PhD. students, and ten are MSc. students. All the participants have background on software product line engineering, and model-driven development, and no prior experience with system safety engineering and assurance cases. The participants were selected by convenience. The participants have answered a profile characterization form about their level of experience in software product lines, model-driven development, assurance cases, and development of safety-critical systems. The information obtained from these questionnaires was further used to allocate the participants in groups. During the experiment, the participants created assurance case models for two different variants from automotive and aerospace product lines using both conventional and MBAC approaches.

6.4.2.3 Formulation of Hypothesis

Table 6.2 shows the formulated hypothesis for the “*Efficiency*” study, used to compare the productivity of the product line MBAC approach with the conventional manual process of assurance case construction. The “DT_c” (i.e., Design Time using the conventional approach) and “DT_m” (Design Time using the product line MBAC) variables were defined. “DT_c” represents the overall time to design an assurance case for a given product variant using the conventional approach. "DT_m" represents for the overall time to design an assurance case for the same variant using product line MBAC. Therefore, there are two hypotheses on Table 6.2: "$H0_{dt}$" and "Hp_{dt}". *"H0"* is the null hypothesis, which is true when both techniques are equivalent or when the time to design an assurance case model for a given product variant using the conventional approach is less than using the product line MBAC. *"Hp_{dt}"* is the positive hypothesis, which is true when product line MBAC takes less time to design an assurance case for a given product variant in comparison with the conventional approach.

Table 6.2 – Hypotheses for the efficiency study.

$H0_{dt}$	The time required to design an assurance case for a product variant from the analysis of its design and safety assessment models using the conventional approach (C) is less or equal than the time required to perform the same task using the product line MBAC (M). $DT_c <= DT_m$
Hp_{dt}	The time required to design an assurance case for a product variant from the analysis of its design and assessment models using the conventional approach (C) is greater than the time required to perform the same task using the product line MBAC (M). $DT_c > DT_m$

Table 6.3 shows the formulated hypothesis for the "*Effectiveness*" study to compare the quality of the assurance case models generated using the product line MBAC and conventional approaches. The "ER_c" (Error Rate using C), and "ER_m" (Error Rate using M) variables were defined on Table 6.3. *"ER_c"* and *"ER_m"* represent the error rate associated with variant-specific assurance case models designed using respectively "C" and "M" approaches.

Since both "*Efficiency*" and "*Effectiveness*" hypotheses consider different ranges of a single real value, they are mutually exclusive, i.e., only one of them can be true. The *"$H0_{dt}$"*

Table 6.3 – Hypotheses for the effectiveness study.

$H0_{er}$	The quality, in terms of number of errors, of assurance case models designed using the conventional approach (C) is better or equal to the quality of assurance case models designed using product line MBAC (M). $ER_c <= ER_m$
Hp_{er}	The quality, in terms of number of errors, of assurance case models designed using the product line MBAC (M) approach is better than the quality of assurance case models designed using the conventional approach (C). $ER_c > ER_m$

and *"$H0_{er}$"* hypotheses have the precedence because the results are dependent upon the error margin stipulated in statistical testing.

6.4.2.4 Variable Selection

Dependent variables: the time spent to design an assurance case model, and the error rate identified in the generated assurance case model.

Independent variables: the targeted product variants (HBS-4WB and TFC-ALL), and the assurance case construction approaches (C and M), which are controlled and manipulated.

Environmental variables: the level of the experience of the participants.

6.4.2.5 Study Design

The participants were divided into two balanced groups, by considering the answers of the profile characterization form provided by each participant. Each group is composed by twelve participants. A pilot study was conducted prior the experiment to verify possible problems in order to guarantee the precise execution of the tasks. Table 6.4 shows the planned phases for this experimental study. In the first phase, the participants from "Group 1" had designed an assurance case for "*TFC-ALL*" variant using the conventional approach, while the participants from "Group 2" had used the MBAC approach. In the second phase, the participants from "Group 1" have used the MBAC approach to design an assurance case for "*4WB*" variant, and the participants from "Group 2" have used the conventional approach.

Table 6.4 – Experiment design.

Phase		**Group 1**	**Group 2**
Training		Concepts of system safety engineering, Goal Structuring Notation, and Assurance Case design using GSN graphical editor and MBAC tooling for aircraft braking and pacemaker product variants.	
1st Phase:	Treatment	Conventional	Model-Based
	Factor	Tiriba Flight Control ALL Pilot Modes (TFC-ALL) variant.	
2nd Phase	Treatment	Model-Based	Conventional
	Factor	Four Wheels Braking (4WB) variant.	

6.4.2.6 Instrumentation

The material supplied to the participants performing the assurance case construction tasks were: variant-specific *Architecture*, *Feature*, *Context,* and *Failure* models, a manual describing how to use the tooling support for both conventional and MBAC approaches, and a document with the description of the *"SysSafe"* assurance case pattern fragment and a mapping table showing the relationships between abstract terms from the pattern and system model elements. The documents used during the experiment preparation and operation are available in Appendix C.

The provided models for "*4WB*" and "*TFC-ALL*" variants have the same complexity and structure. During the experiment the participants had to identify the following information elements in variant-specific design and safety assessment models: *"Feature.name"*, *"systemImpl.name"*, *"systemImpl.description"*, a set of *"Hazard.name"*, and *"Hazard.SafetyRequirement"*. Such information was used to instantiate abstract terms from "*SysSafe*" assurance case pattern fragment using both conventional and MBAC approaches.

After the participants had designed variant-specific assurance case models using both conventional and MBAC approaches, the experiment applicant had analyzed each one of these models against the correct template solution, and then, the number of errors associated with each variant-specific assurance case model was obtained.

6.4.3 Operation

6.4.3.1 Preparation

A pilot study was executed to simulate the experiment environment, including all phases planned in Table 6.4. Later, prior the execution of the experiment phase, training on conventional and MBAC approaches and tooling has been provided for the participants. All the participants have designed assurance cases for two different product variants using only one of the assurance case construction approaches.

6.4.3.2 Execution

In each phase, the participants had designed assurance cases for the given product variant using different assurance case approaches (C or M). At the end of each phase, the participants had delivered assurance case models for the given variant and the filled data collection form with the time spent to conclude the phase. Later on the experiment, the applicant analyzed the assurance case models developed by each participant against the correct solution in order to evaluate the quality, in terms of absence of errors, of the generated assurance cases.

6.4.3.3 Data Validation

The profile characterization forms filled by the participants were analyzed prior the allocation of the participants in each group. During the pilot study, the experiment applicant observed the subjects to verify whether they had gathered the needed data to perform the experiment phases. Thus, controllability of the *independent variables* was achieved, avoiding their influence under the *dependent variables*. The subjects considered in the pilot study did not participate of the experiment. The data collection forms were checked after the experiment to ensure that the participants correctly filled in.

6.4.3.4 Data Collection

For the "*Efficiency*" study, Table 6.5 shows the time spent by each participant during the assurance case construction phases using both conventional and MBAC approaches. Each line has five columns: *"G"* represents the group in which the given participant was allocated; *"V"* represents the product variant from which the assurance case was generated, *"A"* represents the assurance case construction approach used to develop the assurance case, *"P"* refers to the participant identifier, and *"Time"* represents the time spent to generate an assu-

Table 6.5 – Timing data for the efficiency study.

Phase 1					**Phase 2**				
G	**P**	**V**	**A**	**Time *(mm:ss)***	**G**	**P**	**V**	**A**	**Time *(mm:ss)***
1	23	TFC-ALL	C	02:54	1	9	4WB	M	02:35
1	3	TFC-ALL	C	03:21	1	18	4WB	M	02:41
2	24	TFC-ALL	M	04:14	1	2	4WB	M	03:39
2	13	TFC-ALL	M	04:39	1	17	4WB	M	03:50
2	19	TFC-ALL	M	05:00	1	3	4WB	M	04:04
2	14	TFC-ALL	M	05:41	1	11	4WB	M	04:42
1	11	TFC-ALL	C	06:12	1	21	4WB	M	05:03
2	20	TFC-ALL	M	06:24	1	23	4WB	M	05:05
1	18	TFC-ALL	C	06:25	2	5	4WB	C	05:10
2	8	TFC-ALL	M	06:27	2	14	4WB	C	05:12
2	5	TFC-ALL	M	06:33	1	4	4WB	M	05:45
2	12	TFC-ALL	M	06:33	1	1	4WB	M	06:07
1	9	TFC-ALL	C	06:38	2	24	4WB	C	06:20
1	21	TFC-ALL	C	07:37	1	7	4WB	M	06:20
2	10	TFC-ALL	M	08:00	2	8	4WB	C	06:21
1	1	TFC-ALL	C	08:08	2	22	4WB	C	08:09
2	6	TFC-ALL	M	10:00	1	15	4WB	M	08:40
2	16	TFC-ALL	M	10:25	2	20	4WB	C	09:29
2	22	TFC-ALL	M	10:26	2	6	4WB	C	09:38
1	4	TFC-ALL	C	10:30	2	19	4WB	C	10:06
1	15	TFC-ALL	C	15:43	2	10	4WB	C	11:00
1	17	TFC-ALL	C	16:08	2	13	4WB	C	13:06
1	7	TFC-ALL	C	17:37	2	16	4WB	C	26:20
1	2	TFC-ALL	C	27:51	2	12	4WB	C	39:47

rance case.

For the "*Effectiveness*" study, Table 6.6 shows the error rate associated with assurance case models designed by each participant using both conventional and MBAC assurance case construction approaches. Each line has five columns: the first four columns are similar to Table 6.5, and the fifth column (*"Error Rate"*) refers to the percentage of errors found in the delivered assurance case models. The error rate was calculated in the following way: each instantiated assurance case pattern term received a value between 0 and 1. Zero indicates that the instantiation information is 100% incorrect and 1 indicates that no errors have been found. In addition, each instantiated term received a weigh according to its complexity where two assurance case pattern terms, which require a single information element to be instantiated received 0.125 weigh, and three terms, which require multiple information elements received 0.25 weigh. Finally, the error rate associated with a given assurance case model is 1 subtracted by the sum of pairs "error*weigh" associated with each instantiated term, multiplied by 100. For example, considering the pairs ("error*weigh") for the assurance case model delivered by the participant "18" in phase 1 we have: 1 – ((1*0.125) + (1*0.125) + (1*0.25) + (0*0.25) + (1*0.25)) = 0.25*100 = 25% error rate.

Table 6.6 – Error rate data for the effectiveness study.

Phase 1					**Phase 2**				
G	**P**	**V**	**A**	**Error Rate**	**G**	**P**	**V**	**A**	**Error Rate**
2	14	TFC-ALL	M	0%	1	3	4WB	M	0%
2	6	TFC-ALL	M	0%	1	18	4WB	M	0%
2	10	TFC-ALL	M	0%	1	1	4WB	M	0%
2	5	TFC-ALL	M	0%	1	2	4WB	M	0%
2	12	TFC-ALL	M	0%	1	7	4WB	M	0%
2	13	TFC-ALL	M	0%	1	15	4WB	M	0%
2	22	TFC-ALL	M	0%	1	11	4WB	M	0%
2	8	TFC-ALL	M	0%	1	17	4WB	M	0%
2	19	TFC-ALL	M	0%	1	21	4WB	M	0%
2	20	TFC-ALL	M	0%	1	9	4WB	M	0%
2	24	TFC-ALL	M	0%	1	4	4WB	M	0%
2	16	TFC-ALL	M	0%	2	14	4WB	C	0%
1	3	TFC-ALL	C	0%	2	22	4WB	C	0%
1	17	TFC-ALL	C	0%	2	8	4WB	C	0%
1	21	TFC-ALL	C	0%	2	19	4WB	C	0%
1	18	TFC-ALL	C	25%	2	24	4WB	C	0%
1	7	TFC-ALL	C	25%	2	5	4WB	C	10%
1	11	TFC-ALL	C	25%	2	10	4WB	C	12,50%
1	9	TFC-ALL	C	25%	2	16	4WB	C	22,50%
1	4	TFC-ALL	C	25%	2	13	4WB	C	25%
1	15	TFC-ALL	C	32,50%	2	12	4WB	C	27,50%
1	2	TFC-ALL	C	50%	2	6	4WB	C	40%
1	1	TFC-ALL	C	55%	2	20	4WB	C	50%
1	23	TFC-ALL	C	75%	1	23	4WB	M	-

From the analysis of the assurance case models delivered by all the participants, it has been identified that only the participant "23" has not delivered the "*4WB*" assurance case model created using the MBAC approach. In addition, the "TFC-ALL" assurance case delivered by this participant using a conventional approach had the highest error rate in comparison with other participants. Therefore, this participant was considered an outlier and then, excluded from the data analysis.

6.4.3.5 Efficiency and Effectiveness Data Analysis and Interpretation

The "*Efficiency*" data shown in Table 6.5 is ordered by the time taken by the participants to design assurance cases using both conventional and model-based approaches. From the analysis of the 23 best results in both phases it has been found that 17 results were obtained using the MBAC approach, identified by the letter "*M*", and the remaining 6 results were obtained with the support of the Conventional approach identified by letter "*C*". The "*Efficiency*" data shown in Table 6.5 is also presented in a line graph in Figure 6.17. In this graph, the lines for both conventional and MBAC approaches are paired peer each participant identifier, where the taller line means more time has been taken to designing an assurance case using a specific approach. The time spent by the participants "3", "8" and "14" were similar in both conventional and MBAC approaches due their level of experience with the EMF platform.

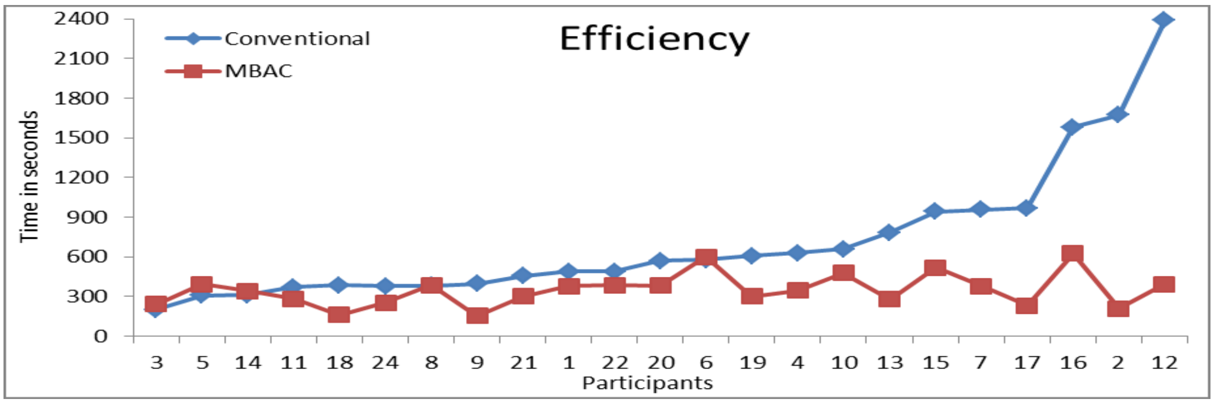


Figure 6.17. Efficiency line graph.

Table 6.7 shows the average of the time taken by all the participants designing assurance cases using both conventional and MBAC approaches, and their proportions. By analyzing the average times of the participants from both groups it was verified that using the conventional approach to design assurance cases took approximately 100.87% longer than MBAC.

Table 6.7. Efficiency average timings.

Approach	**Avg.(*mm:ss:mil*)**	**Percent**
Conventional	12:02.0869565	66.76314%
MBAC	05:59.4782609	33.23685%
Total	18:01.5652174	100.00000%

The "*Effectiveness*" data shown in Table 6.6 is ordered by the error rate found in the assurance cases models developed using both conventional and model-based approaches. From the analysis of all assurance case models delivered in both phases, no errors were found in the assurance case models developed using the MBAC approach. Also, eight assurance case models produced with the support of the conventional approach did not present errors, and the remaining fifteen assurance case models presented error rates ranging from 10% to 55%. The "*Effectiveness*" data is also presented in a line graph diagram that shows the data dispersion (Figure 6.18).

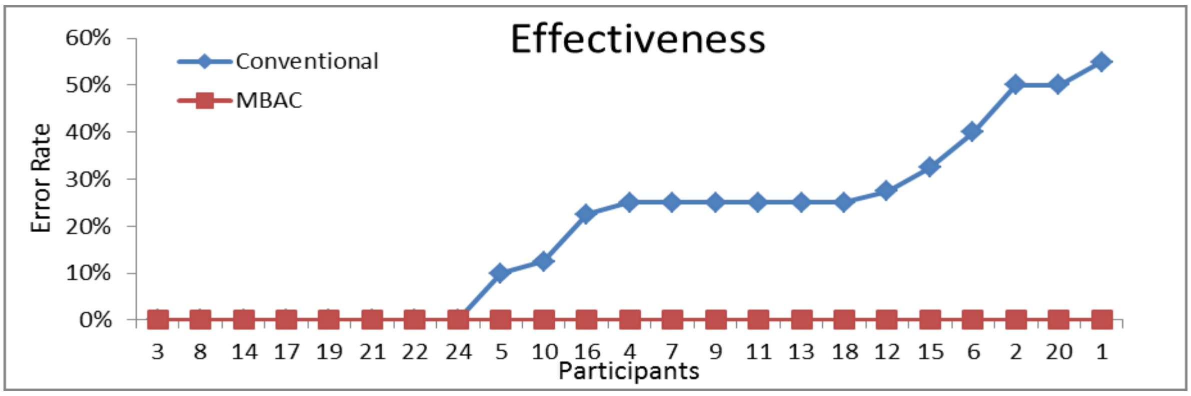


Figure 6.18. Effectiveness line graph.

Table 6.8 shows the error rate average found in the assurance case models developed using both conventional and MBAC approaches. From the data analysis, it has been verified that no errors were found in the assurance case models generated with the support of the MBAC approach. However, the error rate average found in the assurance case models developed using the conventional approach is 20.81%. Considering each group of participants, the error rate averages for the conventional approach are respectively 24% for "*Group 1*", and 15.625% for "*Group 2*" against 0% for the MBAC approach in both groups.

Table 6.8. Effectiveness error rate average.

Approach	Avg. Group	Avg. (% error rate)	Percent
Conventional	Group 1: 24% Group 2: 15.625%	20.81%	100%
MBAC	Group 1: 0% Group 2: 0%	0%	0%
Total		20.81%	100%

6.4.3.6 Hypothesis Testing

This section presents the statistical hypothesis testing used to evaluate "*Efficiency*" and "*Effectiveness*" studies. Paired T-Test was applied to test both effectiveness and efficiency hypothesis presented in Section 6.4.2.3, and "Two-sided" unpaired T-Test were applied after removing four outliers. The time spent by each participant in developing assurance cases using both approaches were processed in the form of "seconds" time unit with the support of "R" statistical computing environment (FREE SOFTWARE FOUNDATION, 2016).

Table 6.9 shows the "Paired", and "Two-sided" unpaired T-Test results for the "*Efficiency*" study. The first column indicates the type of T-Test, and the second column the source of the data, in this case, the time spent by the participants to design assurance cases. The column "*Means*" indicates the resultant mean for the given T-Test. In a Paired T-Test there is one mean, resultant from the average of subtracting each set member from one side by its counterpart in the other set. For "Two-sided" T-Test there are two means, one for the conventional approach and other for MBAC approach. The column "*d.f*" indicates the degrees of freedom with respect to how many different values can be found in the sets. Finally, "*t*" and "*p*" are variables considered in the hypothesis testing. The "*Paired*" T-Test was chosen to compare the timing differences between two samples associated with each participant. "Two-sided" T-Test was used to calculate the means for conventional and MBAC data sets, because participants can be outliers in a given approach, breaking the pairs. "Two-sided" means that both conventional and MBAC data sets have the same number of elements, which the same number of outliers was removed from each set.

Table 6.9. T-test results for the efficiency study.

T-Test	Data	Means	d.f.	T	P
Paired	Efficiency	362.6087	22	3.3715	0.002752
Two-sided	Efficiency	597.6190 339.1429	40	3.441	0.00137

“Chi-squared test” was applied to both “*Effectiveness*” and “*Efficiency*” studies to detect outliers in the samples from both groups of participants. The outliers were then removed before executing the “Two-sided” unpaired T-Test. Table 6.10 shows the results of the “Chi-squared test” for the “*Efficiency*” study. The column “*Group*” indicates the number of the group. The column “X^2” indicates the result of subtracting each data set value by the variance of the complete set, and “*p*” indicates the p-value associated with “X^2”. “*Position*” indicates the position of the identified outlier on the set, i.e., highest or lowest. Finally, the “*Outlier*” column indicates the abnormal timings in seconds. Figure 6.19 illustrates the visualization of the identified outliers in boxplot diagrams for conventional and MBAC samples. The squares represent the range in which the sample values are concentrated. A filled square represents the mean of a given sample, and “*o*” symbols represent the outliers.

Table 6.10 – Chi-squared testing for the efficiency study.

Approach	Group	X^2	p-value	Position	Outlier
Conventional	1	5.2057	0.02251	Highest	1671
	2	7.0119	0.008097	Highest	2387
MBAC	1	4.5111	0.03367	Highest	520
	2	2.3839	0.1226	Highest	626

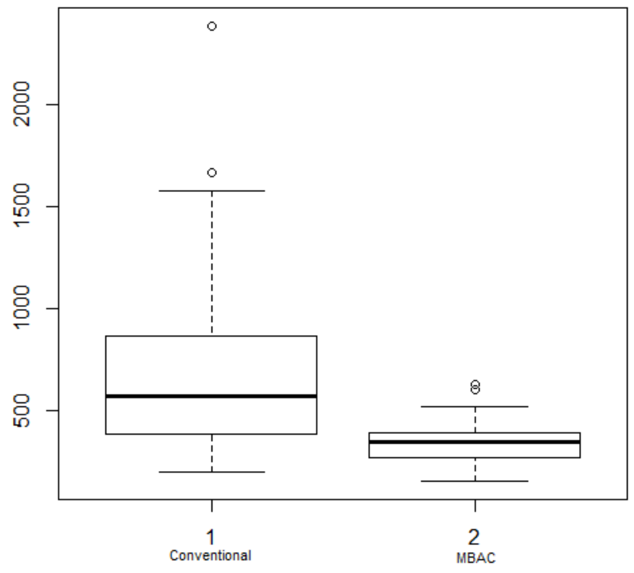


Figure 6.19. Boxplots for the efficiency study.

From the analysis of the T-Test results for the "*Efficiency*" study shown in Table 6.9, it was verified that "*$H0_{dt}$*" hypothesis can be statistically rejected since all p-values from T-Test results are lower than the margin of error (0.01%), conforming to the established significance level of 99.99%. Additionally, since the t-value is positive in both paired and unpaired T-Tests, "*Hp_{dt}*" hypothesis can be accepted. Therefore, the conventional assurance case construction approach takes more time than MBAC.

With regard to the "*Effectiveness*" study, the results for "Paired" and "Two-sided" unpaired T-Tests are presented in Table 6.11. Similarly to the "*Efficiency*" study, the Paired T-Test was executed first, then, two outliers were removed, and the "Two-sided" T-Tests were executed. In the "*Effectiveness*" study, paired and unpaired T-Tests were executed to test the effectiveness hypothesis separately in both groups of the participants, i.e., "Group 1" and "Group 2". This is done to verify the effectiveness of both approaches in each group of participants. The first column indicates the type of the T-Test, and the second column, the source of data, in this case error rates for each delivered assurance case model in each group. The column "*Means*" indicates the resultant mean for the given T-Test. The column "*d.f*" indicates the degrees of freedom with respect to how many different error rates can be found in the sets. Finally, "*t*" and "*p*" are variables considered in the hypothesis testing. "Chi-squared test" has also been executed to detect outliers in the "*Effectiveness*" data sets, as shown in Table 6.12. In this table, "X^2" indicates the result of subtracting each error rate data set value by the variance of the complete set, and "*Outliers*" column indicates abnormal error rate values associated with each data set, e.g., error rate data from "Group 1" using the conventional approach. Figure 6.20 illustrates the visualization of the identified error rate outliers in boxplot diagrams for each group of participants using both conventional and MBAC approaches.

Table 6.11 - T-test results for the effectiveness study.

T-Test	Data	Means	d.f.	T	P
Paired	Effectiveness/ Group1	23.86364%	10	4.266	0.001647
Paired	Effectiveness/ Group 2	15.625%	11	3.1184	0.0009779
Two-sided	Effectiveness/ Group 1	15.75% 00.0%	18	3.6232	0.001944
Two-sided	Effectiveness/ Group2	17.04545% 00.00000%	20	3.2383	0.0041119

Table 6.12 – Chi-squared testing for the effectiveness study.

Approach	Group	X^2	p-value	Position	Outlier
Conventional	1	2.8166	0.00933	Highest	55%
	2	3.9221	0.04766	Highest	50%
MBAC	1	NaN	NA	-	-
	2	NaN	NA	-	-

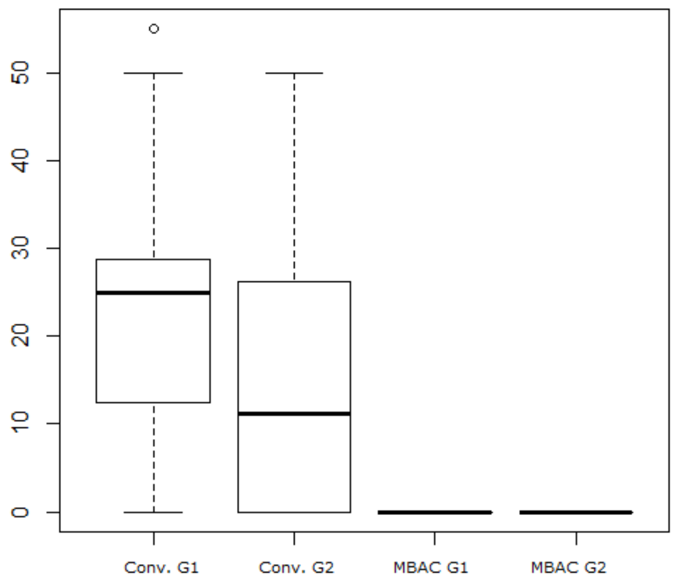


Figure 6.20. Boxplots for the effectiveness study.

By analyzing paired and unpaired T-Test results for the "*Effectiveness*" study shown in Table 6.11, *"$H0_{er}$"* can be statistically rejected since all p-values from effectiveness T-Test results are lower than the margin of error (0.01%), conforming to the established significance level of 99.99%. In addition, since the t-value is positive in both paired and unpaired T-Tests, "*Hp_{er}*" hypothesis can be statistically accepted. Therefore, statistically we can conclude that the MBAC approach contributes to reduce the errors in the construction of assurance cases for product line instances, improving their quality.

6.4.4 Threats to Validity

6.4.4.1 Internal Validity

- *Participants experience level*: since different levels of knowledge from the participants could compromise the data, the participants were allocated into two balanced groups. The level of experience of the participants on the techniques involved in the experimental study, collected from the analysis of profile characterization forms, was considered to allocate them in separated groups. Although some of the participants have prior experience on model-driven techniques and modeling tools, a training covering the concepts of safety-critical product lines, assurance cases, and model driven development and tools has been provided to all the participants. This is done to ensure similar levels of experience of the participants in both conventional and model-based approaches for assurance case construction;

- *Facilities used during the study*: different computers and configurations may affect the timings. Therefore, to mitigate this threat, all the participants used the same configuration, model, and operating system in equal numbers. During the course of the experiment, the participants were not allowed to change their computers when performing the tasks.

6.4.4.2 Construction Validity

- *Hypothesis expectations*: some of the participants already knew this research project and they were aware that the MBAC approach is supposed to reduce the complexity of the assurance case construction process, which reflects one of the hypotheses. Therefore, in order to avoid partiality, we have enforced the participants to keep a "steady pace" during the whole experimental study.

6.4.4.3 External Validity

- *Interaction between configuration and treatment*: since it could be possible that assurance case construction tasks for product line instances are not accurate for real-world safety-critical systems, we have created assurance case construction tasks with real-world prototype systems derived from automotive (Hybrid Braking System) and aerospace (Tiriba Flight Control) product lines, using assurance case patterns;

- *Interaction between selection and treatment*: since the experimental study was applied to Software Engineering post-graduate students, there is a threat that these students are not a representative of the population safety-critical systems engineers. In order to mitigate this threat, a training showing the concepts needed to develop assurance case models for safety-critical product line instances, examples and exercises have been provided to all the participants.

6.4.4.4 Conclusion Validity

- *Measure reliability*: it refers to the metrics used to measure the effort in developing, and the quality of assurance case models generated for a particular product line instance using both conventional and MBAC approaches. To mitigate this threat, the "*error rate*", and the "*time*" taken to design an assurance case were considered;
- *Low statistic power*: the ability of a statistical testing in reveal the reliable data. T-Tests were used for statistically analyzing the collected data to avoid a low statistic power.

The results of this experimental study have shown the effectiveness of the MBAC in supporting the automatic generation of correct assurance case models for product line instances. No errors were found in 23 assurance cases generated with the support of the MBAC approach. The results of this experimental study have also shown that the MBAC approach is more efficient in terms of the time spent by safety engineers in developing assurance cases. The usability of both conventional and MBAC approaches was qualitatively evaluated by means of a post-experiment questionnaire answered by the participants after performing the tasks assigned in the experiment. For 18 participants, the MBAC approach and tooling provides better usability in comparison with the conventional approach. For 2 participants, both Conventional and MBAC assurance case construction approaches have the same degree of usability, and for 3 participants the Conventional approach provides better usability in comparison with MBAC. Such information was extracted from the analysis of the answers provided by the participants for questions defined in the post-experiment questionnaire available in Appendix C.

6.5 Discussion and Limitations

The following limitations of the current model-based assurance cases approach and tooling (HAWKINS *et al.* 2015) have been identified during the generation of assurance case models for HBS-SPL product variants:

- **Lack of precedence control in weaving time**: there is a lack of mechanisms to express and control the order in which abstract terms from assurance case patterns should be instantiated. This is necessary to guarantee the absence of instantiation errors caused by data dependence between these terms. For example, considering the weaving model presented in Figure 6.10, the instantiation of *"hazard"* abstract term should precede the instantiation of *"HSFMType"* term because the instantiation of a "*hazard*" term provides the required information to instantiate *"HSFMType"*;

- **Lack of mechanisms for mapping dependencies between pattern abstract terms**: dependence relations between abstract terms establish data dependence relationships that should be considered during the instantiation of each term. The instantiation of an abstract term may require information provided by the instantiation of other abstract term in a weaving operation. For example, the instantiation of multiple occurrences of "*HSFMType*" term requires the knowledge about the instantiation of a "*hazard*" term. Therefore, it is not possible instantiating the "*HSFMType*" abstract *"term"* without the information about the hazard since it determines the component failure mode information required to instantiate the "*HSFMType*";

- **Lack of mechanisms for specifying and handling constraints in a model weaving operation**: since the specification of an abstract "*term*" in an assurance case pattern may embed implicit references to additional information that is part of the instantiation of an abstract "*term*", it should be considered during a model weaving operation. For example, the "Identified *{systemX}* hazards" expression pattern in "C4" element shown in Figure 6.6, implicitly refers to the set of hazards associated with an instance of "*systemX*" term. This implies that a "*System"* object from an architecture model, e.g., developed in AADL language, is associated with zero or more "*Hazard*" objects from a failure model. Such reference can be seen as a constraint that should be considered during the instantiation of "*systemX"* abstract term. The current MBAC tolling solution does not provides mechanisms to specify and handle this kind of constraint in a model weaving operation;

- **Lack of mechanisms to trace claims to evidence items**: this is related to the lack of means to trace assertions embedded into a claim to the provenance information of evidence items that substantiate that claim. By provenance, we mean information about how the evidence was built, comprising information about personnel, tools and techniques, and activities performed to generate the evidence. Such traceability is important for certification purposes. Therefore, a third-part certification authority would not gain enough confidence in the safe operation of a given safety-critical system if the evidence is not managed or traced to the assurance case (NAIR et al. 2014); and

- **Lack of mechanisms for handling pattern choices in a weaving operation:** it is concerned to handling pattern choices according to the input data provided by reference information models (system models). For example, the instantiation of assurance case patterns arguing the absence of omission, commission, value, early and late failure modes is dependent upon the information provided by system models. Therefore, for an omission failure mode, the respective assurance case pattern should be instantiated.

In order to overcome the above limitations, the following changes should be incorporated into the MBAC approach: mechanisms for specifying precedence control and dependence relations between abstract terms should be added to the weaving modeling; the MBAC instantiation program should be modified to support precedence control and management of dependencies between abstract terms of assurance case patterns; and mechanisms for handling assurance case pattern instantiation choices during weaving operations should also be incorporated into the instantiation program. In addition, the MBAC approach should also be restructured to support the specification and handling of constraints in a model weaving operation. Some of these modifications have been done in the MBAC approach during the course of this thesis and are detailed in Chapter 7.

6.6 Summary

This chapter has presented a systematic approach to integrate Model-Based Assurance Cases into software product line engineering processes to support the automatic generation of assurance cases for product variants. The approach comprises a set of phases to be included into product line domain engineering, to support the specification of reusable assurance case patterns, asset metamodels, and the weaving model. These artefacts are further used in application engineering to support the automatic generation of assurance cases from a diverse set of variant-specific design and safety assessment models. The results of an experimental study that assessed the feasibility of the MBAC approach to support the generation of assurance cases in product line application engineering has also been presented. The analysis of the results has shown the effectiveness, in terms of quality of the generated assurance cases, and efficiency, in terms of timing, of the MBAC approach in comparison with the conventional manual process of creating an assurance case for a product line instance. In addition to the benefits of traceability between design and safety assessment models, and the assurance case provided by MBAC approach, allowing the coevolution of the system design and the assurance case, its use in software product line engineering automates traceability links between variation in design and context and their impact in variant-specific architecture and safety-related models (safety analysis), and the assurance case. Thus, changes in a given product variant are automatically propagated throughout the assurance case.

Combining the systematic product line variability management and compositional safety analysis approach presented in chapters 4 and 5, and model-based assurance cases has brought automation to traceability of context and design variation throughout safety assessment and assurance case assets defined in "*Product Line Safety Metamodel*" (HABLI, 2009). It filled the gap related to automation left by previous research on model-based safety assessment and assurance cases in safety-critical product lines (HABLI and KELLY, 2010, HABLI 2009, HABLI et al., 2009). Although the benefits of the MBAC approach in safety-critical product line engineering, the success of such approach is dependent upon the correct specification of assurance case patterns, asset metamodels, and the weaving model in domain engineering. The following chapter shows the modifications performed into the MBAC approach to support: the specification and management of assurance case pattern instantiation constraints in model weaving operations, and the concept of artefact pattern and its relationships with assurance case patterns.

Chapter 7

INSTANTIATION CONSTRAINTS AND ARTEFACT PATTERNS IN MODEL-BASED ASSURANCE CASES

7.1 Introduction

The previous chapter has presented an approach to integrate Model-Based Assurance Cases (MBAC) (HAWKINS *et al.* 2015) into safety-critical software product line engineering processes. Such integration allows safety analysts to reuse modeling artefacts required by the MBAC, such as assurance case pattern models and the weaving model, to support the automatic generation of assurance cases for different product variants. Although there are benefits provided by the current MBAC model weaving solution for assurance cases (HAWKINS *et al.* 2015), a set of limitations, aforementioned in Chapter 6, has been identified during the conduction of MBAC case studies presented throughout this thesis. The lack of mechanisms for handling precedence and dependence relationships between assurance case pattern elements that requires instantiation, named *abstract terms*, may lead to errors during the assurance case pattern instantiation. In addition, the lack of explicit traceability links between the assurance case, evidence items and their provenance can affect the confidence in the generated assurance case. Evidence provenance information describes how artefacts used as evidence have been created and managed over their lifecycle, and what techniques, resources were used in their generation (OMG, 2015a). The artefact provenance information increases the degree of assurance of an artefact referenced as evidence in an assurance case, increasing the confidence in the assurance case. Because of that, the current MBAC approach and tooling should be adapted to address these limitations.

The confidence in the assurance case is dependent upon explicit traceability links between assurance claims and provenance information about development and assessment artefacts, e.g., requirements, hazard analysis, fault trees and FMEA results, which substantiates assurance claims (NAIR *et al.* 2014). Because of this, it is desirable to automate such traceability into the current MBAC approach. The traceability between development artefacts used as safety evidence, such as requirements and hazards (RIDDERHOF *et al.* 2007), requirements and source code (MASON *et al.* 2003), requirements and design (NEJATI *et al.* 2012), and requirements and components (LEE *et al.* 2010), and specifically, between assurance case and safety evidence (OMG, 2015a; NAIR *et al.* 2014) have already been addressed in past research. However, there is not an approach to automate the traceability between specific claims and the provenance information about the safety evidence referenced in the assurance case. Earlier research is focused on the motivation and challenges for evidence traceability as well as the proposal of traceability models, and not on the automation of traceability links between the assurance case and evidence provenance information (NAIR *et al.* 2014).

This chapter presents a model-based approach, built upon the MBAC and the OMG Structured Assurance Case Metamodel 2.0 (OMG, 2015a), to support the management of assurance case pattern instantiation constraints, such as precedence and dependence relationships between assurance case pattern *abstract terms* at weaving time, and the explicit traceability between assurance case and safety evidence provenance information. The proposed approach provides guidance to integrate into the current MBAC approach: the specification of traceability links between assurance claims and evidence provenance information; the specification of instantiation constraints in assurance case pattern modeling; and changes in the MBAC instantiation program for handling assurance case pattern instantiation constraints, and supporting the instantiation of artefact patterns and their links to assurance case patterns. The latest version of the OMG Structured Assurance Case Metamodel (SACM) (OMG-SACM, 2015a) and Epsilon model management languages (KOLOVOS *et al.* 2013) have been used to support the automated traceability between assurance claims and provenance information about artefacts that substantiate these claims.

The proposed approach was applied to adapt the MBAC approach, which outputs assurance case models in Goal Structuring Notation (GSN), towards supporting the instantiation of assurance case and artefact patterns in SACM 2.0. Additionally, GSN to SACM 2.0 model transformations and an SACM Eclipse-based editor were developed to support the specification of pattern instantiation constraints and artefact patterns in the assurance case pattern modeling. Although we have instantiated the proposed approach using GSN and Eclipse Modeling Framework platform, the approach is applicable to other notations such as CAE and other model-based development technologies.

The remainder of this chapter is organized as follows. Section 7.2 presents an overview of the OMG SACM 2.0 metamodel. Section 7.3 presents the proposed approach and tooling to support the specification of artefact patterns and their links to assurance case patterns, and pattern instantiation constraints into model-based assurance cases processes (HAWKINS *et al.* 2015). Section 7.4 presents the validation of the approach in an automotive Hybrid Braking System product variant. Section 7.5 shows the discussion and limitations of the proposed solution. Section 7.6 presents the summary of this chapter.

7.2 An Overview of OMG SACM 2.0 Metamodel

This section presents an overview of the newer version of the SACM (SACM 2.0) metamodel used in the proposed approach to support the automated traceability between assurance case claims and evidence management information. SACM 2.0 defines the structural representation for assurance case modeling languages and notations as illustrated in Figure 7.1 (OMG, 2015a). The first part of the specification defines common SACM elements covering Base Classes (SACM 2.0 - Clause 8), Terminology Classes (SACM 2.0 – Clause 10), Assurance Case, Terminology, Argument, and Artefact Packages (SACM 2.0 – Clause 9). The subsequent parts define the Argumentation metamodel (SACM 2.0 – Clause 11) and Artefact metamodel (SACM 2.0 – Clause 12).

7.2.1 SACM 2.0 Argumentation and Artefact Metamodels

The "Argumentation Metamodel", denoted by the green area in Figure 7.1, represents structured assurance arguments, i.e., a convincing argument that a system meets its assurance requirements, which may contain extensive references to the safety evidence. The "*Artefact Metamodel*" represents the evidence and its provenance properties in detail. The "*Argumentation Metamodel*" is intended to allow the interchange between structured arguments and diverse assurance case modeling tools provided by different vendors (OMG,

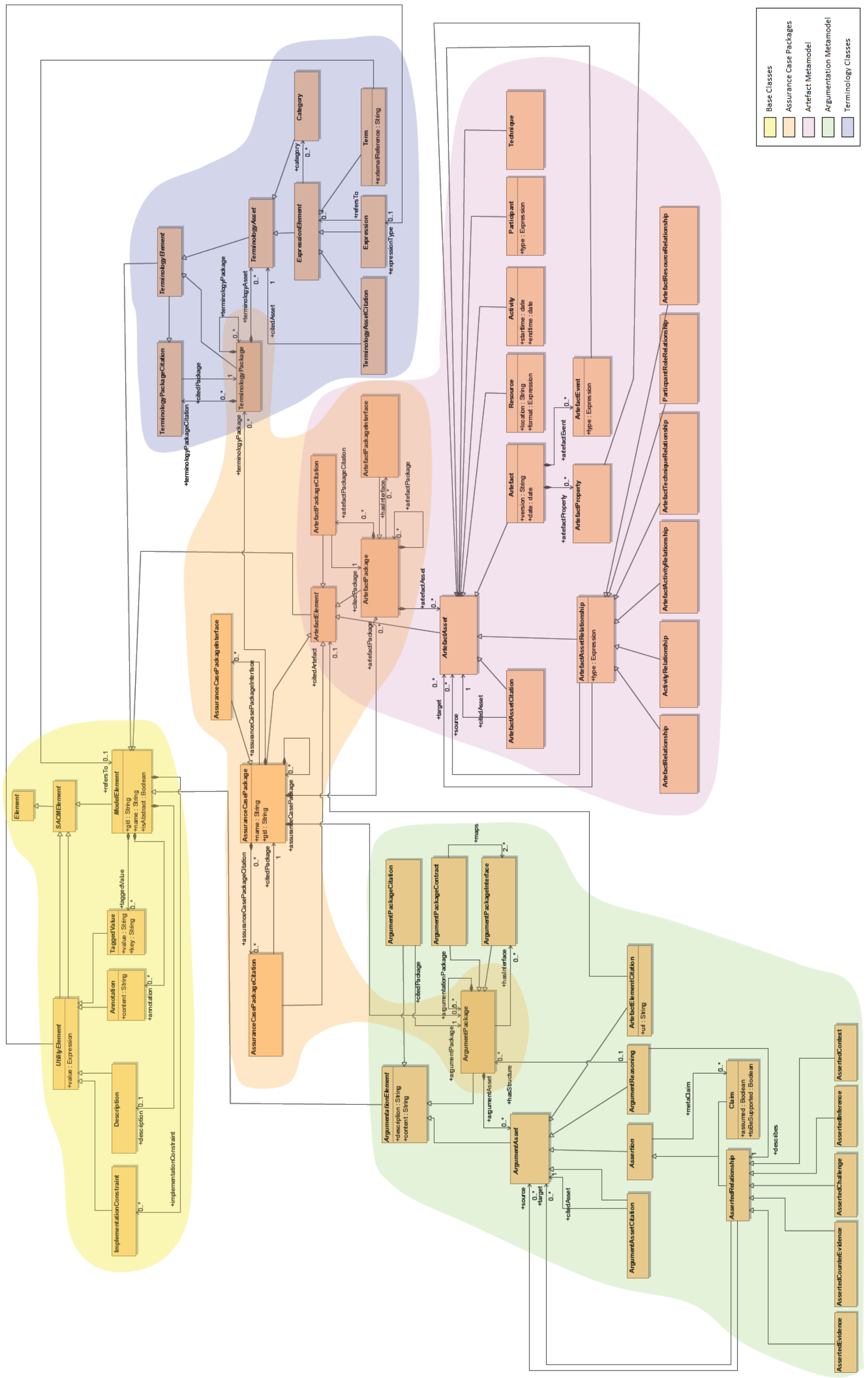


Figure 7.1. Structured assurance case metamodel version 2.0 (OMG, 2015a).

2015a). In SACM 2.0, the "*Argumentation Metamodel*" specification is designed to be standalone, or used in conjunction with the "*Artefact Metamodel*", enabling a simplified support for modeling relationships between evidence and a structured argument. "*Argumentation Metamodel*" represents the core concepts for structured argumentation that underlies existing argumentation notations such as GSN (KELLY, 2003) and CAE (Claim-Argument-Evidence) (BLOOMFIELD and BISHOP, 2010).

The "*Artefact Metamodel*", denoted by the purple area in Figure 7.1, is intended to communicate the form in which safety artefacts are collected and documented by participants using techniques, resources and activities. This metamodel defines a catalogue of elements for constructing and interchanging packages of evidence communicating how the evidence was collected (evidence provenance). The "*Artefact Metamodel*" can be used in conjunction with the "*Argumentation Metamodel*" to allow the authors of assurance claims to offer evidentiary support for their positions. The detailed specification of the SACM 2.0 metamodel is available elsewhere (OMG, 2015a).

7.2.2 SACM 2.0 Bases Classes and Terminology Metamodel

The SACM 2.0 metamodel enhancements in *Base Classes* in comparison with SACM 1.0 (OMG, 2013), shown in the yellow area in Figure 7.1, and the newer "*Terminology Metamodel*" (the blue area in Figure 7.1) allow rich specification of assurance case patterns. Assurance case pattern concepts are detailed in Chapter 2 (Section 2.4.2.2). Patterns support the systematic reuse and effective composition of assurance cases along with the underlying argumentation supporting claims. Assurance case patterns support the definition of constraints conventions to specify assurance cases within a particular domain in order to address regulatory requirements or accepted practices in that domain (OMG, 2015a). The *"ModelElement"* is the base element for the majority of the SACM modeling elements, i.e., the majority of the SACM elements extend it. *"ModelElement"* was redefined with the inclusion of "*isAbstract*" Boolean attribute used to indicate whether an element is part of an assurance case pattern. In addition, associations to zero or multiple "*ImplementationConstraint"* elements attached to a *"ModelElement"* allow the specification of any implementation constraint associated with the conversion of an SACM element, e.g., *"ArgumentAsset"*, *"Expression"*, being abstract to being concrete. *"ImplementationConstraint"* element allows specifying detailed pattern instantiation rules that should be addressed to instantiate an abstract *"ModelElement",* which is a fragment of an

SACM 2.0 assurance case pattern. Therefore, *"ImplementationConstraints"* should only be attached to *"ModelElements"* in which the value for the *"isAbstract"* attribute is "*true*" (OMG, 2015a).

The inclusion of "*ImplementationConstraint*" elements into the assurance case pattern specification also allows safety analysts to attach references to information available in development artefacts that substantiates assertions specified in argument expression patterns (YAMAMOTO, 2014) embedded into assurance claims. For example, considering the *"Risk Top-Level Argument"* pattern shown in Chapter 6 (Section 6.3.1), the "*Identified {systemX} hazards*" argument expression pattern embedded into *"C4"* context element contain references to hazard information available into the product failure model. Therefore, when *"C4"* element is instantiated, references to a set of "*Hazard*" elements provided by the product failure model can be attached to instantiations of "*C4*" element via "*TaggedValue*" elements. A "*TaggedValue"* represents a simple "*key/value*" pair that can be attached to any SACM 2.0 "*ModelElement"* (OMG, 2015a). It allows users to add custom attributes such as references to safety evidence into assurance claims.

The "*Terminology Metamodel*" provides a set of model elements to organize abstract and concrete assurance case pattern "*Terms"* and "*Expressions"* that can be used within the naming and description of SACM 2.0 arguments and artefacts. "*TerminologyPackage"* is the container element for terminology assets such as "*Categories", "Terms"* and "*Expressions"*. "*Category"* elements define categories for "*Term"* and "*Expression"* elements, which can be used to group these elements within "*TerminologyPackages"*. For example, a *Category* can be used to describe the terminology associated with a given safety standard, project or system. *"Term"* is used to model abstract and concrete SACM 2.0 assurance case pattern elements. Abstract "*Terms"* represent assurance case pattern elements that require further instantiation, i.e., abstract *Terms* are placeholders for concrete terms. In an abstract "*Term"*, the inherited "*isAbstract"* attribute is set to *"true"*. In a concrete "*Term"* this attribute is set to *"false"*. "*Expression"* elements are used to specify argument expression patterns embedded into assurance claims. The definition of an "*Expression*" element may comprise combinations between abstract and concrete "*Terms*" and other "*Expression*" elements. Abstract "*Expressions"* are denoted by the "*isAbstract"* attribute being set to *"true"* and concrete ones by being set to *"false"*. For example, the "*Identified {systemX} hazards*" argument "*Expression*" pattern can be defined in the SACM 2.0 as the concatenation of one abstract "*Term*" and two concrete "*Expression*" elements.

SACM 2.0 "*Base Classes"* and "*Terminology Metamodel*" provide abstractions that allow the specification of instantiation rules for assurance case pattern elements. "*Argumentation"* and "*Artefact"* elements support the specification of explicit traceability links between argument elements, safety evidence and provenance information in assurance case patterns. *Artefact* model elements provide support for the specification of artefact patterns linked to assurance case patterns. Artefact patterns define the structure of the safety evidence and its provenance. The expressiveness of the SACM 2.0 metamodel provides support for adapting the current Model-Based Assurance Cases approach to support the automated generation of explicit traceability links between the assurance case, safety evidence and their provenance, reducing the complexity and ensuring the consistency of evidence traceability. In addition, other types of constraints can be attached to an assurance case pattern specification, allowing the specification of pattern instantiation constraints between abstract "*Terms"* from assurance case patterns which require instantiation.

7.3 Integrating Instantiation Constraints and Artefact Patterns into Model-Based Assurance Cases

Restructuring the current MBAC approach (Hawkins *et al.* 2015) to support evidence traceability and instantiation constraints in assurance case pattern modeling based on SACM 2.0 requires changes in the assurance case pattern specification and pattern instantiation processes. Changes in the assurance case pattern specification comprise additional steps to specify: pattern instantiation constraints such as precedence control, i.e., to establish the order in which abstract "*Terms*" defined in the assurance case pattern should be instantiated, and data dependencies between assurance case pattern abstract "*Terms*"; traceability links between abstract pattern elements and evidence; and artefact patterns for evidence referenced in the assurance case. These changes are propagated throughout the assurance case pattern instantiation process with the addition of mechanisms for handling pattern instantiation constraints and evidence traceability links defined into assurance case and artefact patterns. This section presents the proposed extensions for the MBAC approach and tooling (HAWKINS *et al.* 2015) to support the management of pattern instantiation constraints, and traceability between evidence provenance and the assurance case. Section 7.3.1 presents the extensions in the MBAC tooling support and Section 7.3.2 presents the redefined model-based assurance cases process.

7.3.1 Extensions in Model-Based Assurance Cases Tool Support

Figure 7.2 shows the extensions to the Model-Based Assurance Cases tooling highlighted in gray. These extensions comprise: the addition of an EMF-based editor for the SACM 2.0 assurance case modeling notation; a model transformation plugin, which converts a GSN assurance case pattern specification into an equivalent in SACM 2.0, developed with the support of EMF and Epsilon Transformation Language (ETL) (KOLOVOS *et al*. 2013); and changes into the MBAC instantiation program. In addition, language specific editors have been added to support the manipulation of different design and assessment models such as architecture and failure models.

The GSN editor (HAWKINS *et al.* 2015) provides abstractions to specify GSN assurance case patterns. The GSN assurance case patterns are input artefacts for the GSN2SACM model transformation plugin, developed in the course of this thesis, to support the transformation of GSN assurance case pattern specifications into an equivalent SACM 2.0 pattern specification. The SACM 2.0 editor allows safety analysts to specify assurance case pattern instantiation constraints in the assurance case pattern itself. Instantiation constraints allow safety analysts to establish data dependencies between assurance case pattern elements, and defining the order in which patterns elements should be instantiated to avoid errors during

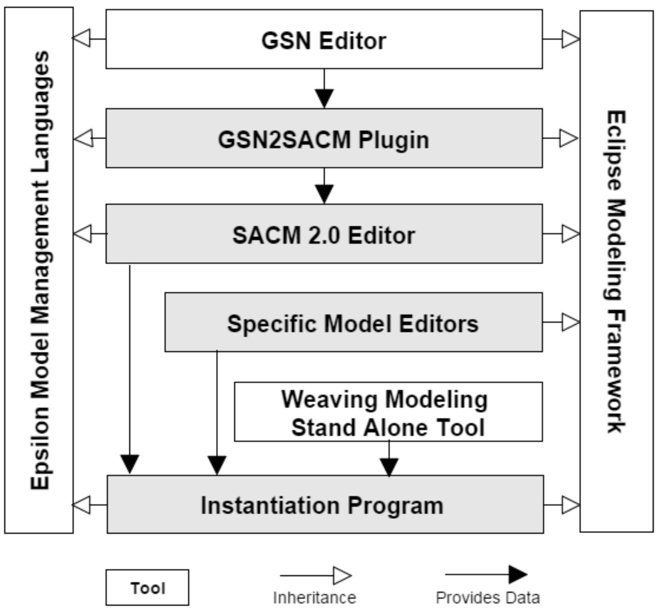
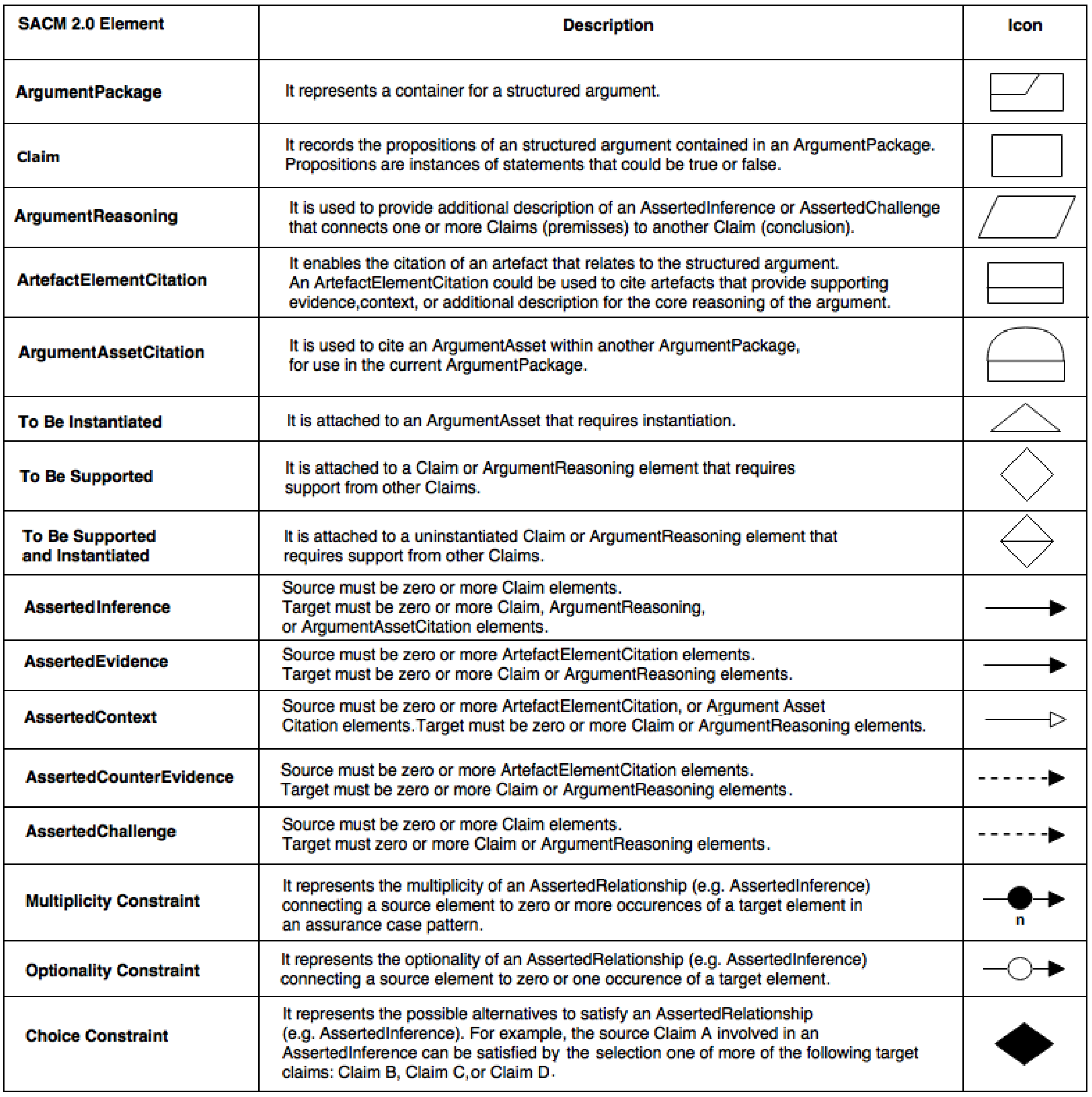


Figure 7.2. MBAC extensions.

assurance case pattern instantiation processes. In addition, the SACM 2.0 allows safety analysts to specify patterns for artefacts referenced as evidence items in assurance case, which is not allowed in the current MBAC approach (HAWKINS *et al*. 2015). Therefore, the addition of support for specifying assurance case pattern instantiation constraints and artefact patterns into the current MBAC approach and tooling is the main research contribution in model-based assurance cases techniques. In the following, design and assessment models that comply with metamodels defined in specific model editors provide the information required to instantiate assurance case and artefact pattern elements. No changes were performed in the weaving modeling tool that defines how assurance case pattern elements relate to the information provided by different system models. Finally, the MBAC instantiation program has been changed to support the manipulation and instantiation of assurance case and artefact pattern specifications in SACM 2.0. The following subsections present the detailed description of the SACM 2.0 model editor, GSN2SACM model transformation plugin, and the instantiation program.

7.3.1.1 SACM 2.0 Notation and Editor

The SACM 2.0 model editor has been developed upon the OMG SACM 2.0 metamodel presented in section 7.2, and Eclipse Modeling Framework platform (ECLIPSE, 2016). The SACM editor supports the specification of assurance case and artefact patterns, and their links, and assurance case pattern instantiation constraints. Figure 7.3 shows the description of the proposed SACM 2.0 assurance case modeling notation elements and their respective icons. The SACM 2.0 notation elements and their respective graphical representations were defined from the analysis of SACM 2.0 abstractions and their correspondences with GSN assurance case modeling abstractions, and artefact modeling abstractions defined in SPEM standard (OMG, 2008). From such analysis, GSN "*Module*", "*Goal*", "*Strategy*", "*Module Reference*" and "*AwayGoal/Context/Justification /Assumption/Solution*" (GSN Standard, 2011) were respectively mapped to the following assurance case modeling abstractions in SACM 2.0: "*ArgumentPackage*", "*Claim*", "*ArgumentReasoning*", "*ArtefactElementCitation*" and "*ArgumentAssetCitation*". Thus, graphical representations for these GSN elements were used to represent their equivalent abstractions in SACM 2.0. In addition, the graphical icons used to represent "*isAbstract*", "*toBeSupported", and "isAbstract and ToBeSupported"* constraints associated with an SACM 2.0 element are the same used to represent "*Undeveloped*", "*Uninstantiated*", and "*Undeveloped and Uninstantiated*" GSN entity abstractions (HABLI and KELLY, 2010).



SACM 2.0 Element	Description	Icon
ArgumentPackage	It represents a container for a structured argument.	⏢
Claim	It records the propositions of an structured argument contained in an ArgumentPackage. Propositions are instances of statements that could be true or false.	▭
ArgumentReasoning	It is used to provide additional description of an AssertedInference or AssertedChallenge that connects one or more Claims (premisses) to another Claim (conclusion).	▱
ArtefactElementCitation	It enables the citation of an artefact that relates to the structured argument. An ArtefactElementCitation could be used to cite artefacts that provide supporting evidence,context, or additional description for the core reasoning of the argument.	⊟
ArgumentAssetCitation	It is used to cite an ArgumentAsset within another ArgumentPackage, for use in the current ArgumentPackage.	⌓
To Be Instantiated	It is attached to an ArgumentAsset that requires instantiation.	△
To Be Supported	It is attached to a Claim or ArgumentReasoning element that requires support from other Claims.	◇
To Be Supported and Instantiated	It is attached to a uninstantiated Claim or ArgumentReasoning element that requires support from other Claims.	◇
AssertedInference	Source must be zero or more Claim elements. Target must be zero or more Claim, ArgumentReasoning, or ArgumentAssetCitation elements.	→
AssertedEvidence	Source must be zero or more ArtefactElementCitation elements. Target must be zero or more Claim or ArgumentReasoning elements.	→
AssertedContext	Source must be zero or more ArtefactElementCitation, or Argument Asset Citation elements.Target must be zero or more Claim or ArgumentReasoning elements.	⇾
AssertedCounterEvidence	Source must be zero or more ArtefactElementCitation elements. Target must be zero or more Claim or ArgumentReasoning elements.	⇢
AssertedChallenge	Source must be zero or more Claim elements. Target must zero or more Claim or ArgumentReasoning elements.	⇢
Multiplicity Constraint	It represents the multiplicity of an AssertedRelationship (e.g. AssertedInference) connecting a source element to zero or more occurences of a target element in an assurance case pattern.	—●→ n
Optionality Constraint	It represents the optionality of an AssertedRelationship (e.g. AssertedInference) connecting a source element to zero or one occurence of a target element.	—○→
Choice Constraint	It represents the possible alternatives to satisfy an AssertedRelationship (e.g. AssertedInference). For example, the source Claim A involved in an AssertedInference can be satisfied by the selection one of more of the following target claims: Claim B, Claim C,or Claim D.	◆

Figure 7.3. SACM 2.0 argumentation elements.

With regard to abstractions for linking assurance case elements into an argument structure, a filled arrow is used to represent: the inference that exists between one or more assertions ("*AssertedInference*"); and the declaration that one or more "*Evidence*" artefacts, cited via "*Artefact Element Citations*", provide information that helps to establish the truth of an assurance "*Claim*" (i.e., "*AssertedEvidence*"). A dashed arrow is used to represent: a counter-argument that a user declares to exist between one or more "*Claims*" and another "*Claim*" ("*AssertedChallenge*"); and an "*AssertedCounterEvidence*" relationship used to associate a safety "*Evidence*", cited via "*Artefact Element Citations*", to a "*Claim*", where the evidence which is being asserted infers that the "*Claim*" is false. The empty arrow represents an "*AssertedContext*" relationship that declares the information cited in an "*Artefact Element*

Citation" and provides the context for the interpretation and definition of an assurance "*Claim"* or "*ArgumentReasoning"* element.

GSN multiplicity and optionality structural abstractions (GSN STANDARD, 2011), used to respectively represent generalized *n-ary*, optional and alternative relationships between GSN assurance case elements, are represented by "*ImplementationConstraint"* elements in SACM 2.0. A "*Multiplicity"* implementation constraint is represented by a solid ball (meaning zero or more) attached to an "*AssertedRelationship"* element and a label next the ball indicates the assurance case pattern element that determines the cardinality of the relationship, e.g., n = "*Term*". The "*Optionality"* implementation constraint is represented by a hollow ball attached to an "*AssertedRelationship"* element indicating the optionality of the relationship. Finally, the GSN Option/Choice element is also represented by an "*ImplementationConstraint"* in SACM 2.0. A "*Choice"* constraint is used to represent possible alternatives that satisfy an "*AssertedRelationship"* where annotations are used to indicate the nature of the choice to be made (e.g., *1-of-n* and *m-of-n* selection). In addition, the SACM 2.0 model editor allows users to specify combinations between abstract and concrete "*Terms"* and "*Expressions"* elements from "*Terminology*" package to create argument expression patterns to be attached to argumentation elements specified in assurance case patterns.

Additionally, SACM 2.0 editor also provides artefact modeling abstractions to support the specification of patterns for artefacts, i.e., safety evidence, referenced in assurance case. It allows safety analysts to define patterns for linking assurance cases to the safety evidence. Figure 7.4 shows the definition of SACM 2.0 artefact elements and their respective graphical representations. From the analysis of both SACM 2.0 "*Artefact Metamodel"* and SPEM 2.0 metamodel it has been verified that SACM 2.0 "*Artefact Metamodel*" abstractions have similar semantics of the UML stereotypes defined in SPEM 2.0. Thus, some of SPEM 2.0 graphical representations were used to represent the SACM 2.0 artefact modeling elements. The SPEM 2.0 "*WorkProductDefinition"*, "*Process"*, "*Activity"*, "*RoleDefinition"*, and "*ToolDefinition"* graphical representations were used to respectively represent the following SACM 2.0 artefact model elements: "*Artefact"*, "*ArtefactEvent"*, "*Activity"*, "*Participant"* and "*Technique"*. "*ArtefactPackage"* is a container for artefact model elements and their relationships.

SACM 2.0 Artefact Element	Description	Icon
ArtefactPackage	It represents a container for distinguishable units of data referenced in an assurance case.	
Artefact	It represents the distinguishable units of data used in an assurance case.	
ArtefactProperty	It enables the specification of characteristics of an Artefact.	N/A
ArtefactEvent	It enables the specification of relevant happenings in the lifecycle of an Artefact, e.g., creation, modification.	
Resource	It is the tangible objects that represent an Artefact. Artefacts are located and accessible somewhere,in the form of some electronic file, e.g., doc, pdf and png format.	
Activity	It represents an unit of work related to Artefact Assets.	
Participant	It represents the specification of the parties involved in the management of Artefact Assets.	
Technique	It represents the specific ways of creating Artefacts.	
ArtefactAssetCitation	It represents a reference to Artefact Assets of another ArtefactPackage.	
ArtefactRelationship	It represents Artefacts linked together.	
ActivityRelationship	It represents Activities linked together.	
ArtefactActivityRelationship	It represents Artefacts and Activities linked together.	
ArtefactTechniqueRelationship	It represents Artefacts and Techniques linked together.	
ParticipantRoleRelationship	It represents a Participant linked to other Artefact Assets.	Artefact Asset
ArtefactResourceRelationship	It represents Artefacts and Resources linked together.	

Figure 7.4. SACM 2.0 artefact elements.

Relationships between *artefact assets* are represented by a filled arrow connecting them. The SACM 2.0 model editor provides abstractions to represent each type of artefact asset relationship defined into the SACM 2.0 "*Artefact Metamodel*" shown in Figure 7.4. Whereas the artefacts managed during the development lifecycle do not exist in isolation, *"ArtefactRelationship"* elements record associations between different artefacts such as test cases used to validate a system requirement. "*Artefacts"* are managed in the scope of "*Activities"*, where "*Artefacts*" can be used as input or output ("*ArtefactActivityRelationship"*). "*Artefacts"* are the result of applying a given "*Technique"* and it is usually stored into specific "*Resources",* e.g., pdf file, source code. An "*Artefact"* can also be associated with a *"Participant"* that plays a role or function with regard to the artefact. For example, a "*Participant"* can be the owner or the reviewer of an "*Artefact*" expressed by "*ParticipantRoleRelationship"*. This type of relationship allows associating a "*Participant"* with other *artefact assets* where a "*Participant"* can be the executor of an "*Activity"* or having relationships with other "*Participants"* (e.g., supervisor) (OMG, 2015a). Finally, "*ActivityRelationship"* supports the specification of associations between "*Activities"* defined for a given system/software development process.

7.3.1.2 The GSN2SACM Model Transformation Plugin

The GSN2SACM plugin is a model-transformation tool built using the EMF platform and ETL during the course of this thesis. The tool provides support for transforming an input GSN assurance case pattern specification into an equivalent SACM 2.0 specification. Relationships between GSN (GSN STANDARD, 2011) and SACM 2.0 argumentation elements (OMG, 2015a) were established from the analysis of their metamodels. Table 7.1 shows the correspondence relationships between GSN and SACM 2.0 abstractions. These relationships were considered to implement model transformations in ETL. For example, *Case* is the root element of a GSN assurance case specification, and "*AssuranceCasePackage"* represents the root element of an SACM 2.0 specification. Additionally, GSN "*Goal"*, "*Assumption"*, and "*Justification"* elements are mapped to "*Claim"* elements in SACM 2.0.

Table 7.1 – Relationships between GSN and SACM 2.0 abstractions.

GSN Element	**SACM 2.0 Argumentation Element**
Case	AssuranceCasePackage
Module	ArgumentPackage, ArgumentPackageInterface ArgumentPackageContract,
Goal, Assumption, Justification	Claim
Strategy	ArgumentReasoning
Context, Solution, ModuleReference	ArtefactElementCitation
InContextOf Context	AssertedContext
InContextOf Justification	AssertedInference
InContextOf Assumption	AssertedInference
SolvedBy Strategy or Goal	AssertedInference
SolvedBy Solution or ModuleReference	AssertedEvidence
ChallengedBy (Solution to Goal)	AssertedCounterEvidence (ArtefactElementCitation(s) to Claim(s))
ChallengedBy (Goal to Goal)	AssertedChallenge (Claim(s) to Claim(s))
Choice	Choice ImplementationConstraint
Multiplicity	Multiplicity ImplementationConstraint
Optionality	Optionality ImplementationConstraint
AwayGoal, AwayJustification, AwayAssumption, AwayContext, AwaySolution	ArgumentAssetCitation
Role	Term isAbstract = true
Literal	Term, Expression isAbstract = false

Table 7.2 shows the GSN to SACM model transformations defined for GSN "*Goal*" and "*Strategy*" elements. GSN entity abstractions were mapped to SACM 2.0 "*isAbstract*" and "*toBeSupported*" abstractions. Expressions elements (*"Role"* and *"Literal"*) stored in GSN "*Goal*" and "*Strategy*" elements were mapped to an OCL constraint in SACM 2.0 that sets a "*refersTo*" property of a GSN "*ExpressionElement*" with the corresponding SACM 2.0 "*Claim*" or "*ArgumentReasoning*" element.

Table 7.2 – GSN to SACM 2.0 model transformations.

GSN Element (s)	**SACM 2.0 Element (s)**
Goal	*Claim*
toBeInstantiated: EBoolean	isAbstract: EBoolean
toBeSupported: EBoolean	toBeSupported: EBoolean
nodeText: 0..* ExpressionElement	For each *Expression* associated with a *Claim* context ExpressionElement inv refersToClaim: self.refersTo → r.isTypeOf(Claim)
…	…
Strategy	*ArgumentReasoning*
toBeInstantiated: EBoolean	isAbstract: EBoolean
toBeSupported: EBoolean	toBeSupported: EBoolean
nodeText: 0..* ExpressionElement	For each *Term* associated with an ArgumentReasoning context Term inv refersToArgumentReasoning: self.refersTo → r.isTypeOf(ArgumentReasoning)
-	describes: AssertedRelationship

7.3.1.3 SACM Instantiation Program

The MBAC assurance case pattern instantiation program developed by Hawkins *et al.* (2015) has been adapted to support the instantiation of assurance case and artefact pattern specifications in compliance with OMG SACM 2.0 metamodel. Therefore, a prototype implementation of the SACM Model-Based Assurance Cases (SACM-MBAC) instantiation program has been developed in this thesis. The SACM-MBAC instantiation program has similarities with the original MBAC instantiation program. The SACM-MBAC program executes on the EMF platform and requires the following input artefacts: an SACM 2.0 assurance case and artefact pattern specification, reference information models and metamodels, i.e., system models that provide information to instantiate SACM 2.0 pattern elements, and the weaving model, which defines mapping links between abstract "*terms*" from SACM assurance case pattern, and system model elements. The SACM-MBAC instantiation process has additional steps (steps 2 and 3) in comparison with the original MBAC instantiation program:

1. Firstly, the instantiation program identifies the SACM assurance case and artefact pattern elements that require instantiation;
2. In the following, from the information provided by SACM pattern specification, the SACM-MBAC instantiation program establishes the precedence order and data dependencies between abstract "*Terms*" that should be considered during the pattern instantiation. It avoids data dependencies between SACM pattern elements, ensuring the correct instantiation of SACM assurance case pattern elements that require information available in instances of other pattern elements;
3. The SACM-MBAC program also defines the required information from system models to instantiate each SACM assurance case and artefact pattern element by considering data dependencies between SACM pattern elements, and by querying the weaving model;
4. SACM-MBAC obtains the required information to instantiate SACM assurance case and artefact pattern elements from a diverse set of system models; and
5. Finally, SACM-MBAC outputs the instantiation information in the form of an SACM assurance case model enhanced with artefact information (evidence management information). This output can be viewed with the Eclipse-based SACM assurance tree view and diagram editor.

7.3.2 Redefined Model-Based Assurance Cases Process

The Model-Based Assurance Cases (HAWKINS *et al.* 2015) process has been redefined with modifications in the assurance case pattern modeling to support: the specification of dependence relationships between abstract "*terms*" defined in assurance case patterns; the specification of artefact patterns and their relationships with assurance case pattern elements; and the addition mapping links between abstract "*terms*" defined into artefact patterns and system model elements in the weaving model. The linking between abstract "*terms*" defined in artefact pattern and system model elements in the weaving model is performed in the same way as linking abstract "*terms*" from assurance case patterns to system model elements presented in Chapter 6 (Section 6.2.3). Therefore, the specification of the weaving model remains the same in comparison with the original MBAC approach.

7.3.2.1 Assurance Case Pattern Specification

The starting point of the SACM assurance case pattern specification is an existing GSN assurance case pattern specification, which is transformed in an equivalent SACM specification with the support of the GSN2SACM Eclipse-based model transformation tool (Section 7.3.1.2) as illustrated in Figure 7.5. After transforming a GSN assurance case pattern specification into an equivalent in SACM, precedence and dependence relationships between abstract "*Term*s" defined in the SACM pattern specification should be specified in the form of pattern instantiation constraints using SACM 2.0 "*ImplementationContraint*" model elements. The specification of pattern instantiation constraints using the SACM 2.0 model editor is detailed in Section 7.4.

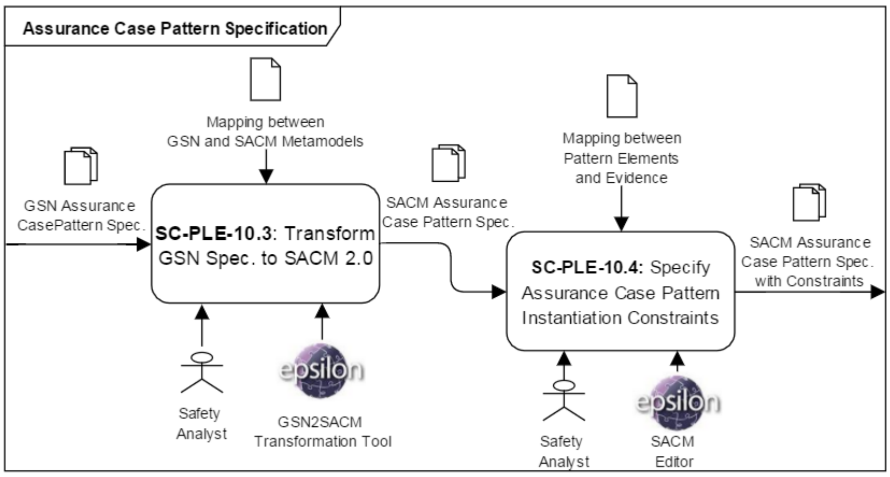

Assurance Case Pattern Specification
Mapping between GSN and SACM Metamodels
GSN Assurance CasePattern Spec.
SC-PLE-10.3: Transform GSN Spec. to SACM 2.0
SACM Assurance Case Pattern Spec.
Safety Analyst
GSN2SACM Transformation Tool
Mapping between Pattern Elements and Evidence
SC-PLE-10.4: Specify Assurance Case Pattern Instantiation Constraints
SACM Assurance Case Pattern Spec. with Constraints
Safety Analyst
SACM Editor

Figure 7.5 Assurance case pattern specification steps.

Pattern instantiation constraints are intended to prevent instantiation errors caused by data dependencies between abstract "*Terms"* defined in an SACM pattern specification, which comprises artefact patterns. For example, considering *"HSFMType"* and "*hazard"* abstract "*Terms*" defined in "*Risk Argument*" pattern shown in Figure 7.6a, the instantiation of *"HSFMType",* which requires the information of a leaf or gate node of a fault tree model, is dependent upon the information about an instance of a *"hazard"* "*Term*" in order to instantiate the correct *"HSFMType"*. Therefore, the instantiation of "*HSFMType*" abstract "*Term*" requires information provided by an instance of a "*hazard*" "*Term*", to perform the search in the correct fault tree model in order to find a fault tree leaf or gate node information element required to instantiate "*HSFMType*" "*Term*" (Figure 7.6b). Thus, pattern instantiation constraints ensure that instances of *"HSFMType"* corresponds to elements of a fault tree associated with an instance of a *"hazard" "Term"* in the assurance case pattern instantiation (Figure 7.6c).

In this thesis, from the analysis of existing assurance case pattern catalogues (WEAVER, 2003; HABLI, 2009), two types of recurrent constraints necessary to ensure the correct instantiation of assurance case patterns have been identified: precedence control and data dependence (also named requires constraints). A precedence constraint aims to establish the order in which abstract "*Terms*" defined in an assurance case pattern specification should be instantiated. This is important to reduce the probability of conflicts between assurance case pattern "*Terms*", which can lead to instantiation errors. Data dependence constraints work together with precedence control to prevent errors during the assurance case pattern instantiation. They establish, in the assurance case pattern specification, data dependencies between abstract *"Terms"* that require instantiation. These dependencies define that the ins-

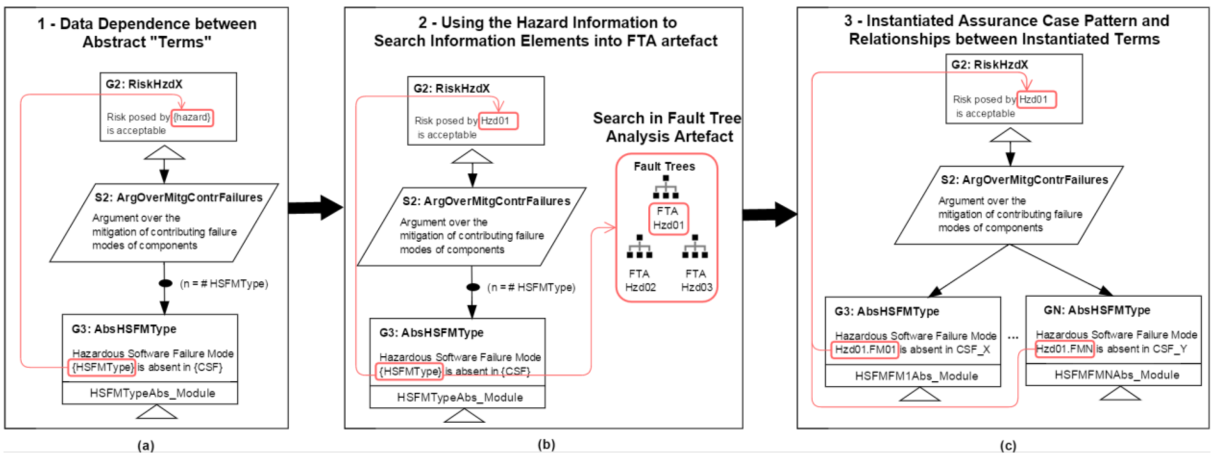

1 - Data Dependence between Abstract "Terms"
G2: RiskHzdX
Risk posed by {hazard} is acceptable
S2: ArgOverMitgContrFailures
Argument over the mitigation of contributing failure modes of components
(n = # HSFMType)
G3: AbsHSFMType
Hazardous Software Failure Mode {HSFMType} is absent in {CSF}
HSFMTypeAbs_Module
2 - Using the Hazard Information to Search Information Elements into FTA artefact
G2: RiskHzdX
Risk posed by Hzd01 is acceptable
S2: ArgOverMitgContrFailures
Argument over the mitigation of contributing failure modes of components
(n = # HSFMType)
G3: AbsHSFMType
Hazardous Software Failure Mode {HSFMType} is absent in {CSF}
HSFMTypeAbs_Module
Search in Fault Tree Analysis Artefact
Fault Trees
FTA Hzd01
FTA Hzd02
FTA Hzd03
3 - Instantiated Assurance Case Pattern and Relationships between Instantiated Terms
G2: RiskHzdX
Risk posed by Hzd01 is acceptable
S2: ArgOverMitgContrFailures
Argument over the mitigation of contributing failure modes of components
G3: AbsHSFMType
Hazardous Software Failure Mode Hzd01.FM01 is absent in CSF_X
HSFMFM1Abs_Module
...
GN: AbsHSFMType
Hazardous Software Failure Mode Hzd01.FMN is absent in CSF_Y
HSFMFMNAbs_Module
(a)
(b)
(c)

Figure 7.6. Data dependency between terms and assurance case pattern instantiation.

tantiation of an abstract "*Term"* referenced in the weaving model, is dependent upon the instantiation of other abstract "*Term"*, as previously explained in Figure 7.6. Figure 7.7 shows an excerpt of the weaving model. In this model, the *"HSFMType"* abstract "*Term*" has a data dependence relationship with *"hazard"* "*Term*". This relationship denotes that the instantiation of *"HSFMType"* is dependent upon the value of an instance of a *"hazard"* "*Term*" as depicted by the query in Structured Query Language (SQL). Thus, *"HSFMType"* "*Term*" should be instantiated with the information about the failure modes associated with a fault tree that matches with an instance of the "*hazard"* term.

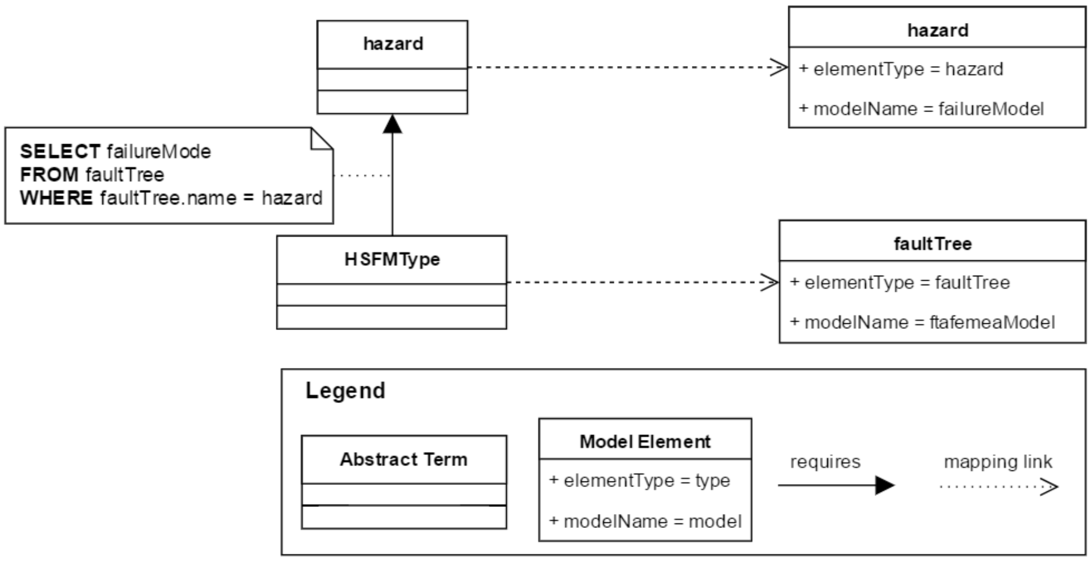


Figure 7.7. Weaving model and pattern instantiation constraints.

7.3.2.2 Artefact Pattern Specification

Artefacts play an important role in the assurance case construction process by providing the evidentiary elements for assurance cases. By means of assertions, artefacts are used to support claims and arguments of an assurance case (OMG, 2015a). Artefacts can be managed in different ways during the software/system development life-cycle. Therefore, artefact management might require the specification of patterns in the same way as assurance cases. These patterns allow safety engineers to specify artefacts, their relationships with other artefacts, development process activities, persons involved in the development process (e.g., system analysts, safety engineers), and techniques used to develop them. The specification of artefact patterns establish the structure of artefacts and their relationships with elements of the development process required to provide the assurance of the evidence used to support the certification of a safety-critical system.

The SACM 2.0 metamodel provides abstractions that allow safety engineers to specify artefact patterns and link them to assurance case pattern elements. Figure 7.8 shows the required steps to integrate artefact pattern specification into the Model-Based Assurance Cases process with the support of the SACM 2.0 assurance case editor. Design and assessment models and their respective metamodels, e.g., process, requirements, design, architecture, and safety analysis models, are input information to create artefact patterns. System/Software development processes and safety standards considered in the development of a given safety-critical system provide the information about the required artefacts and their relationships to achieve safety certification. Therefore, SACM 2.0 artefact pattern modeling elements can be used to define artefact patterns for linking evidence items to their provenance provided by a diverse set of design, process, and assessment models. SACM 2.0 artefact notation elements are detailed in Section 7.3.1.1.

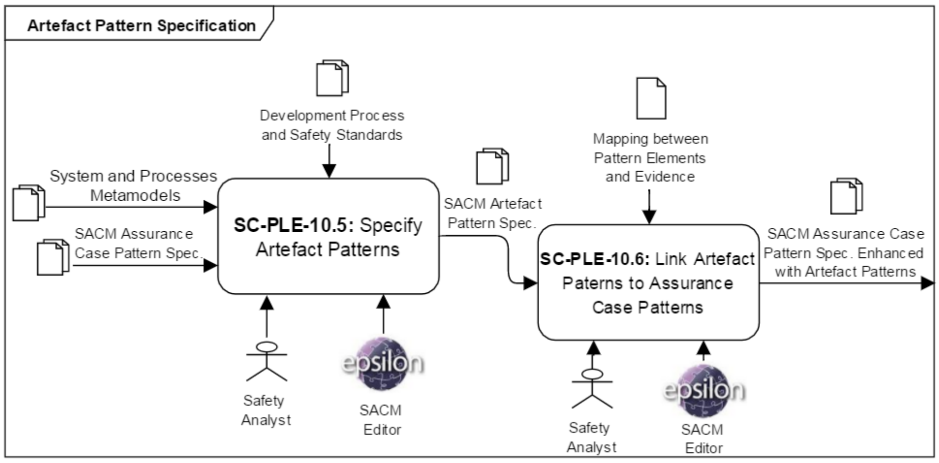

Artefact Pattern Specification
Development Process and Safety Standards
System and Processes Metamodels
SACM Assurance Case Pattern Spec.
SC-PLE-10.5: Specify Artefact Patterns
SACM Artefact Pattern Spec.
Mapping between Pattern Elements and Evidence
SC-PLE-10.6: Link Artefact Paterns to Assurance Case Patterns
SACM Assurance Case Pattern Spec. Enhanced with Artefact Patterns
Safety Analyst
epsilon
SACM Editor
Safety Analyst
epsilon
SACM Editor

Figure 7.8. Artefact pattern specification steps.

Firstly, from the identification of the artefacts used as evidence items to support assurance claims defined in assurance case patterns, artefact patterns are specified to define structural relationships between artefacts used as evidence items with other artefacts, and information elements provided by process models such as techniques, activities, tool support, which provides the artefact provenance. Finally, after specifying artefact patterns, they should be linked to assurance case pattern elements using "*Artefact Element Citation*" elements. Section 7.4 shows detailed examples of artefact patterns and their links to assurance case pattern elements.

7.3.2.3 SACM 2.0 Assurance Case and Artefact Pattern Instantiation

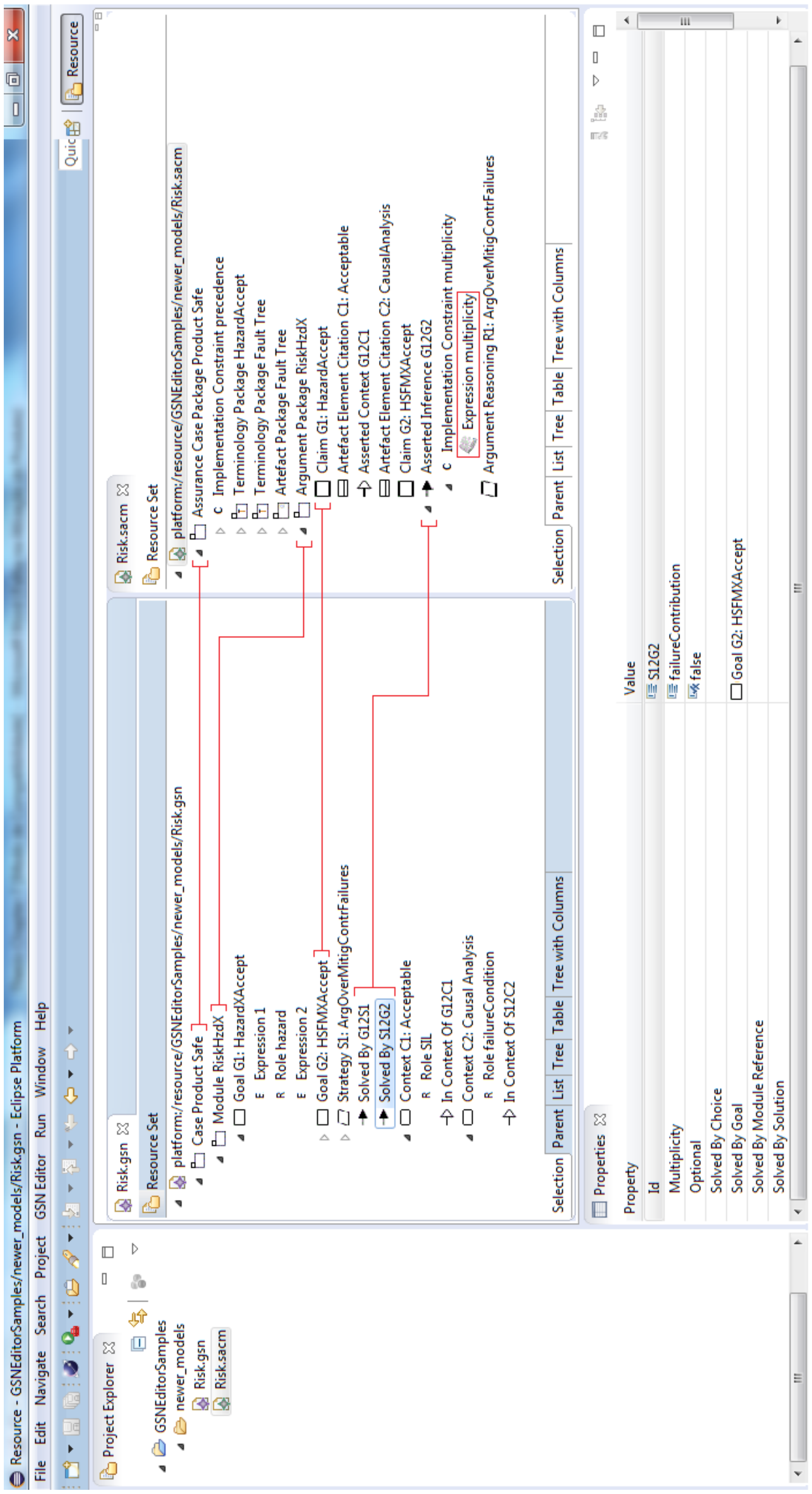
The specification of traceability links between argument and evidence in assurance case and artefact patterns provided by the SACM 2.0 notation contributes to increase the confidence on assurance case models generated with the support of the MBAC approach (HAWKINS et al. 2015). Therefore, the output of assurance case pattern specification is an SACM assurance case pattern model enhanced with artefact patterns. This model, a diverse set of system models and their respective metamodels, and the weaving model are inputs artefacts for the SACM-MBAC instantiation program described in section 7.3.1.3. The process of configuring the instantiation program with the input models is the same described in Chapter 6 (Section 6.2.4) for the original MBAC approach.

7.4 Validation

This section presents the validation of the SACM 2.0 Model-Based Assurance Cases (SACM-MBAC) approach and tooling to support: the specification of pattern instantiation constraints and artefact patterns, and the generation of assurance cases and evidence management information for an automotive braking system product line (HBS-SPL) instance. The GSN assurance case pattern specifications presented in Chapter 6 (Section 6.3.1) are used as input artefacts to illustrate the proposed SACM-MBAC approach.

7.4.1 HBS-SPL Assurance Case Pattern Specification in SACM 2.0

The starting point of the SACM-MBAC approach is a GSN pattern specification, which is transformed into an equivalent SACM 2.0 specification with the support of the GSN2SACM Eclipse-based model transformation plugin. Figure 7.9 illustrates the GSN specification of "*Risk Argument"* assurance case pattern and its equivalent representation in SACM 2.0, after model transformation, in both GSN and SACM 2.0 Eclipse-based model editors. "*Risk Argument*" pattern decomposes the claim arguing that the risk posed by a given system hazard are acceptable into sub-claims arguing that the effects of the occurrence of each contributing component failure is acceptable. Figure 7.9 also shows mapping links between GSN assurance case pattern elements and their representations in the SACM 2.0 assurance case pattern specification. The following GSN elements: "*Case"*, "*Module"*, "*Go-*


Resource - GSNEditorSamples/newer_models/Risk.gsn - Eclipse Platform

File Edit Navigate Search Project GSN Editor Run Window Help

Project Explorer
- GSNEditorSamples
  - newer_models
    - Risk.gsn
    - Risk.sacm

Risk.gsn
- Resource Set
  - platform:/resource/GSNEditorSamples/newer_models/Risk.gsn
    - Case Product Safe
      - Module RiskHzdX
        - Goal G1: HazardXAccept
          - E Expression 1
          - R Role hazard
          - E Expression 2
        - Goal G2: HSFMXAccept
        - Strategy S1: ArgOverMitigContrFailures
        - Solved By G12S1
        - Solved By S12G2
        - Context C1: Acceptable
          - R Role SIL
        - In Context Of G12C1
        - Context C2: Causal Analysis
          - R Role failureCondition
        - In Context Of S12C2

Selection | Parent | List | Tree | Table | Tree with Columns

Risk.sacm
- Resource Set
  - platform:/resource/GSNEditorSamples/newer_models/Risk.sacm
    - Assurance Case Package Product Safe
      - C Implementation Constraint precedence
      - Terminology Package HazardAccept
      - Terminology Package Fault Tree
      - Artefact Package Fault Tree
      - Argument Package RiskHzdX
        - Claim G1: HazardAccept
        - Artefact Element Citation C1: Acceptable
        - Asserted Context G12C1
        - Artefact Element Citation C2: CausalAnalysis
        - Claim G2: HSFMXAccept
        - Asserted Inference G12G2
          - C Implementation Constraint multiplicity
            - Expression multiplicity
        - Argument Reasoning R1: ArgOverMitigContrFailures

Selection | Parent | List | Tree | Table | Tree with Columns

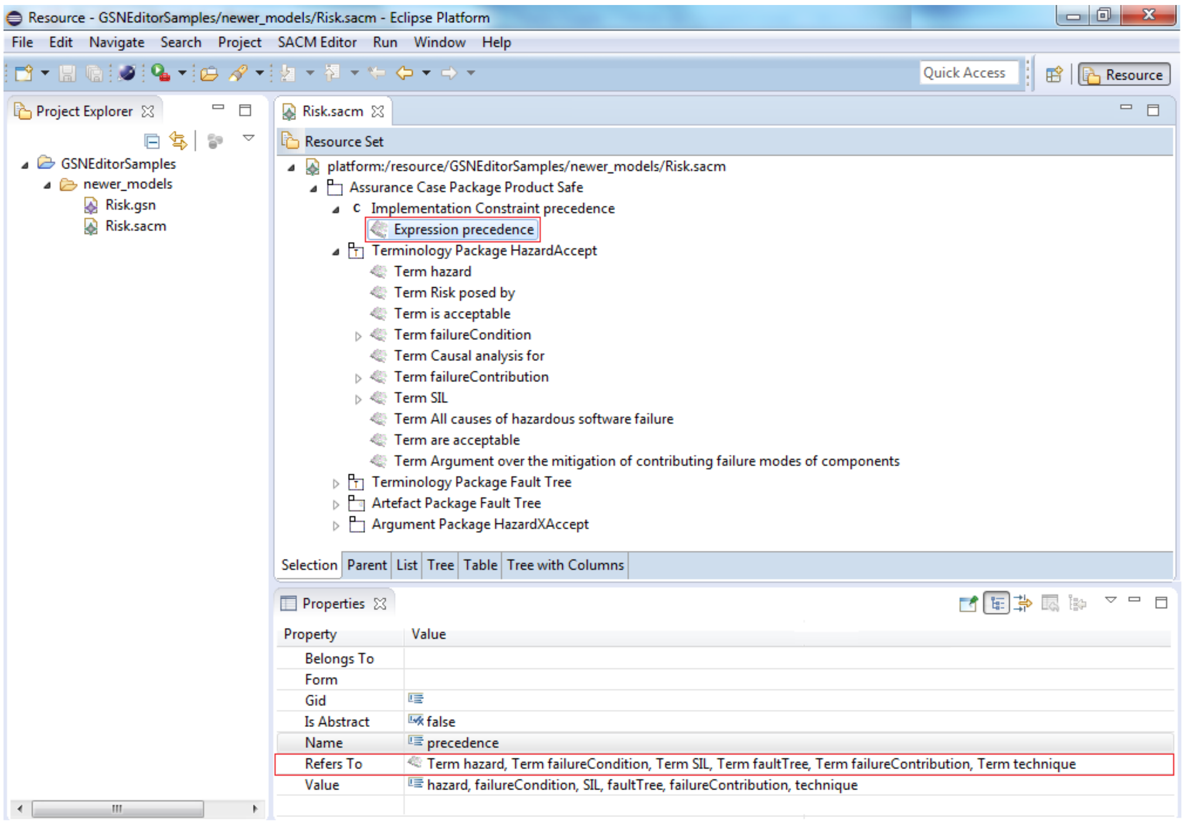
Properties

Property	Value
Id	S12G2
Multiplicity	failureContribution
Optional	false
Solved By Choice	
Solved By Goal	Goal G2: HSFMXAccept
Solved By Module Reference	
Solved By Solution	

Figure 7.9. Risk argument pattern specifications in GSN and SACM 2.0 model editors.

al", *"Strategy"*, *"Context"*, *"SolvedBy"*, *"Expression"*, and *"Role"* are respectively mapped to *"Assurance Case Package"*, *"Argument Package"*, *"Claim"*, *"Argument Reasoning"*, *"Artefact Element Citation"*, *"Asserted Inference"* and *"Term"* elements in the SACM 2.0 pattern specification. An SACM 2.0 abstract "*Term"* represents a GSN "*Role"* element, and a concrete "*Term"* represents an "*Expression"* element in GSN.

The *"G12S1"* and *"S12G2"* GSN "*SolvedBy*" relationships are mapped to *"G12G2"* "*Asserted Inference"* in the SACM 2.0 specification, which denotes relationships between claims and an "*Argument Reasoning"* element. In addition, GSN "*multiplicity"* is represented by a *multiplicity* constraint attached to an "*Asserted Inference"* in the SACM 2.0 pattern model. After transforming a GSN assurance case pattern specification into an equivalent in SACM 2.0, precedence and data dependencies between SACM 2.0 abstract "*Terms*", were specified in the form of implementation constraints. Figure 7.10 shows the specification of a precedence control constraint into "*Risk Argument*" assurance case pattern using the SACM 2.0 model editor. The implementation constraint element named *precedence* is attached to the root node, i.e., the "*Assurance Case Package*" element, of the assurance case pattern specification. The precedence control constraint comprises an expression that defines the order of instantiation for SACM 2.0 abstract *"Terms"* by setting the property "*refersTo"* or



Resource - GSNEditorSamples/newer_models/Risk.sacm - Eclipse Platform
File Edit Navigate Search Project SACM Editor Run Window Help
Quick Access
Resource
Project Explorer
GSNEditorSamples
newer_models
Risk.gsn
Risk.sacm
Risk.sacm
Selection Parent List Tree Table Tree with Columns
Properties

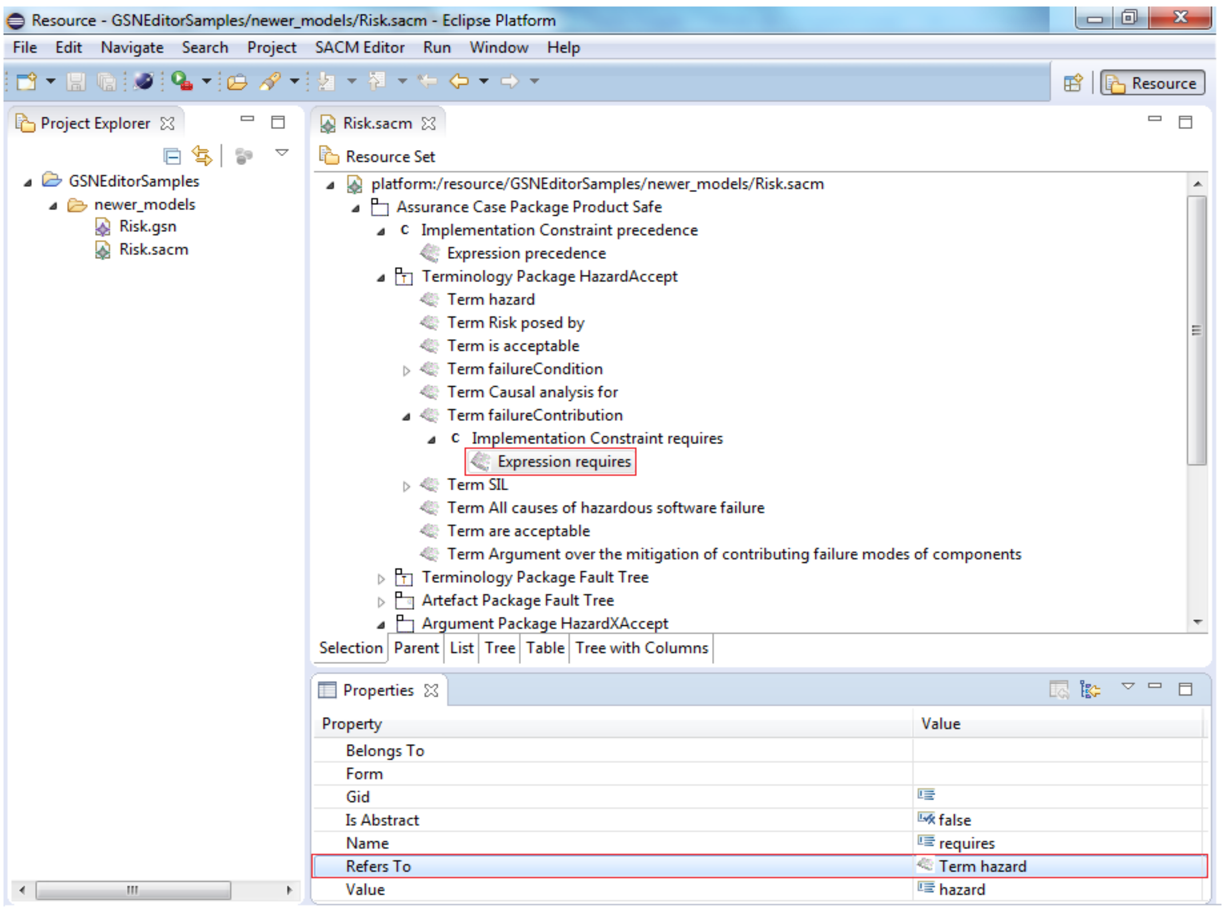
Resource Set
- platform:/resource/GSNEditorSamples/newer_models/Risk.sacm
  - Assurance Case Package Product Safe
    - Implementation Constraint precedence
      - Expression precedence
    - Terminology Package HazardAccept
      - Term hazard
      - Term Risk posed by
      - Term is acceptable
      - Term failureCondition
      - Term Causal analysis for
      - Term failureContribution
      - Term SIL
      - Term All causes of hazardous software failure
      - Term are acceptable
      - Term Argument over the mitigation of contributing failure modes of components
    - Terminology Package Fault Tree
    - Artefact Package Fault Tree
    - Argument Package HazardXAccept

Property	Value
Belongs To	
Form	
Gid	
Is Abstract	false
Name	precedence
Refers To	Term hazard, Term failureCondition, Term SIL, Term faultTree, Term failureContribution, Term technique
Value	hazard, failureCondition, SIL, faultTree, failureContribution, technique

Figure 7.10. Precedence control constraint specification in SACM 2.0.

by setting the "*value*" attribute with the names of the abstract *"Terms"* separated by comma. This information is further used by the instantiation program to control the instantiation of abstract *"Terms"* during the assurance case and artefact pattern instantiation process.

The specification of data dependencies relationships between abstract "*Terms*" in SACM 2.0 assurance case patterns is performed by attaching an implementation constraint named "*requires*" into the given "*Term*". Figure 7.11 shows a "*requires*" data dependence pattern instantiation constraint attached to the "*failureContribution*" abstract "*Term*" which determines that the instantiation of "*failureContribution*" is dependent upon the value assigned to an instance of a *"hazard" "Term"*. At the end of this process, an assurance case pattern specification with instantiation constraints is obtained.


Resource - GSNEditorSamples/newer_models/Risk.sacm - Eclipse Platform
File Edit Navigate Search Project SACM Editor Run Window Help
Resource
Project Explorer
GSNEditorSamples
newer_models
Risk.gsn
Risk.sacm
Risk.sacm
Resource Set
Selection Parent List Tree Table Tree with Columns
Properties

```
platform:/resource/GSNEditorSamples/newer_models/Risk.sacm
  Assurance Case Package Product Safe
    Implementation Constraint precedence
      Expression precedence
    Terminology Package HazardAccept
      Term hazard
      Term Risk posed by
      Term is acceptable
      Term failureCondition
      Term Causal analysis for
      Term failureContribution
        Implementation Constraint requires
          Expression requires
      Term SIL
      Term All causes of hazardous software failure
      Term are acceptable
      Term Argument over the mitigation of contributing failure modes of components
    Terminology Package Fault Tree
    Artefact Package Fault Tree
    Argument Package HazardXAccept
```

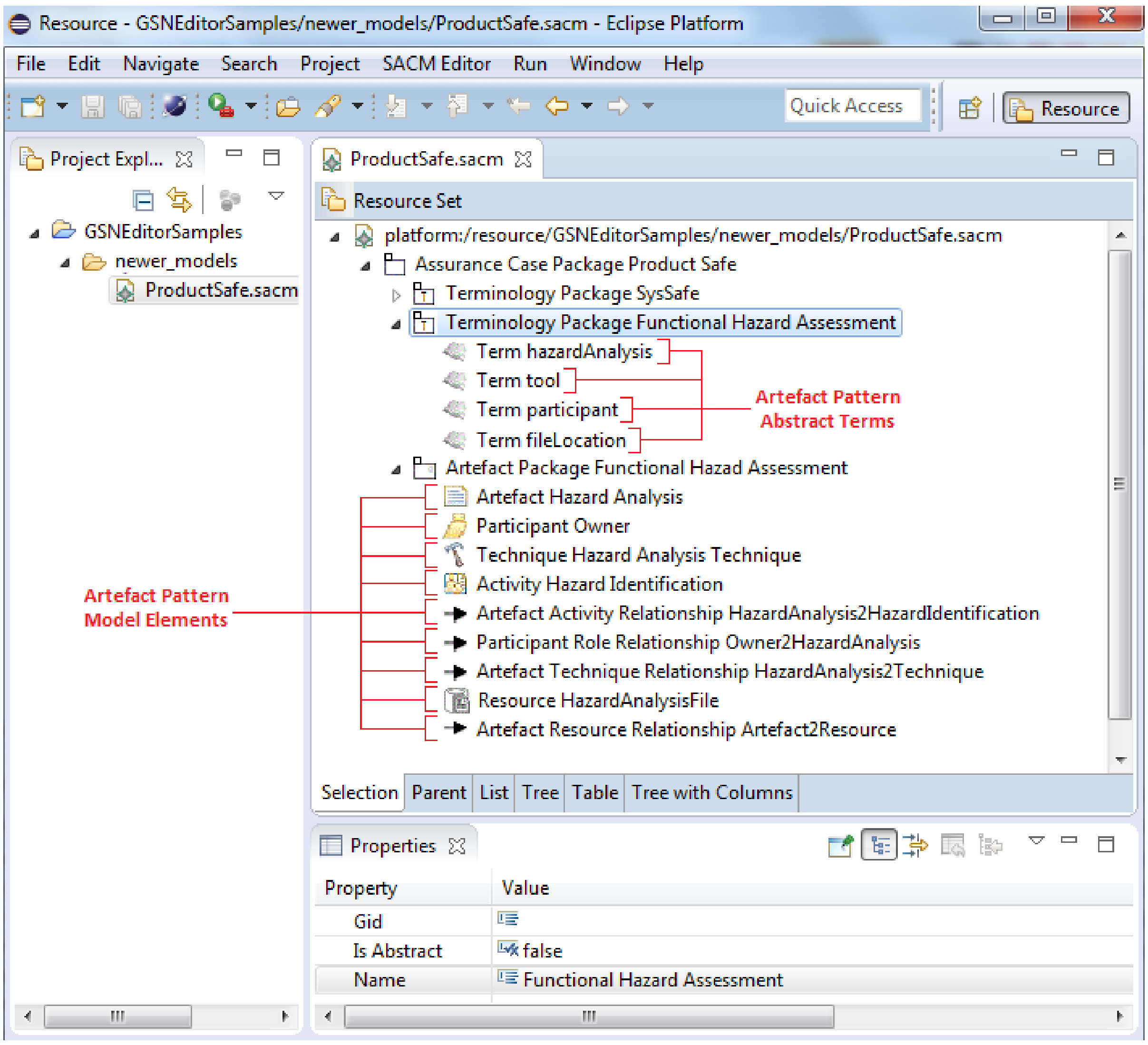
Property	Value
Belongs To	
Form	
Gid	
Is Abstract	false
Name	requires
Refers To	Term hazard
Value	hazard

Figure 7.11. Data dependence pattern instantiation constraint in the SACM 2.0 assurance case pattern.

7.4.2 HBS-SPL Artefact Pattern Specification in SACM 2.0

Considering the Hybrid Braking System automotive product line case study (OLIVEIRA *et al.* 2014) used though this thesis, its development process, and ISO 26262 -

Road Vehicles Functional Safety standard, artefacts and their relationships required for assurance were defined. From the analysis of such information, artefact patterns related to product requirements, architecture, functional hazard assessment, fault trees and FMEA results, can be created with the support of the SACM 2.0 model editor. Figure 7.12 shows the "*Functional Hazard Assessment*" artefact pattern specification in the SACM 2.0 model editor. This pattern defines that a "*Hazard Analysis*" artefact is resultant from performing the "*Hazard Identification*" activity with the support of a given "*Hazard Analysis Technique*" and tooling such as HaZOP (Hazard Origin and Propagation) and HiP-HOPS compositional safety analysis tool. A "*Hazard Analysis*" artefact can be owned by one or more safety analysts professionals involved in the execution of "*Hazard Analysis*" activities. Finally, a "*Hazard Analysis*" artefact is associated with a "*Hazard Analysis File*" resource generated in the end of the hazard analysis process. "*Hazard Analysis File*" element represents the stan-


Resource - GSNEditorSamples/newer_models/ProductSafe.sacm - Eclipse Platform
File Edit Navigate Search Project SACM Editor Run Window Help
Quick Access
Resource
Project Expl...
GSNEditorSamples
newer_models
ProductSafe.sacm
ProductSafe.sacm
Resource Set
Artefact Pattern Abstract Terms
Artefact Pattern Model Elements
Selection Parent List Tree Table Tree with Columns
Properties

platform:/resource/GSNEditorSamples/newer_models/ProductSafe.sacm
Assurance Case Package Product Safe
Terminology Package SysSafe
Terminology Package Functional Hazard Assessment
Term hazardAnalysis
Term tool
Term participant
Term fileLocation
Artefact Package Functional Hazad Assessment
Artefact Hazard Analysis
Participant Owner
Technique Hazard Analysis Technique
Activity Hazard Identification
Artefact Activity Relationship HazardAnalysis2HazardIdentification
Participant Role Relationship Owner2HazardAnalysis
Artefact Technique Relationship HazardAnalysis2Technique
Resource HazardAnalysisFile
Artefact Resource Relationship Artefact2Resource

Property	Value
Gid	
Is Abstract	false
Name	Functional Hazard Assessment

Figure 7.12. Functional hazard assessment artefact pattern specification in SACM 2.0.

dard output associated with a "*Hazard Analysis*" artefact. The "*hazardAnalysis*", "*tool*", "*participant*", and "*fileLocation*" abstract *Terms* defined into "Functional Hazard Assessment" artefact pattern are respectively associated with "*Hazard Analysis*", "*Hazard Analysis Technique*", "*Owner*", and "*Hazard Analysis File*" artefact pattern elements. Abstract "*Terms*" defined in an artefact pattern specification are referenced in the weaving model in the same way as abstract "*Terms*" defined in assurance case patterns.

Figure 7.13 illustrates an excerpt of the "*Functional Hazard Assessment*" artefact pattern. It is important to highlight that the SACM 2.0 model editor allow specifying both assurance case and artefact patterns in the same model. It supports the specification of traceability links between the assurance case pattern and artefact pattern elements via SACM "*Artefact Element Citation*" elements, which enable to reference an artefact that provides the supporting evidence for the reasoning embedded in an assurance case argument (OMG, 2015a).

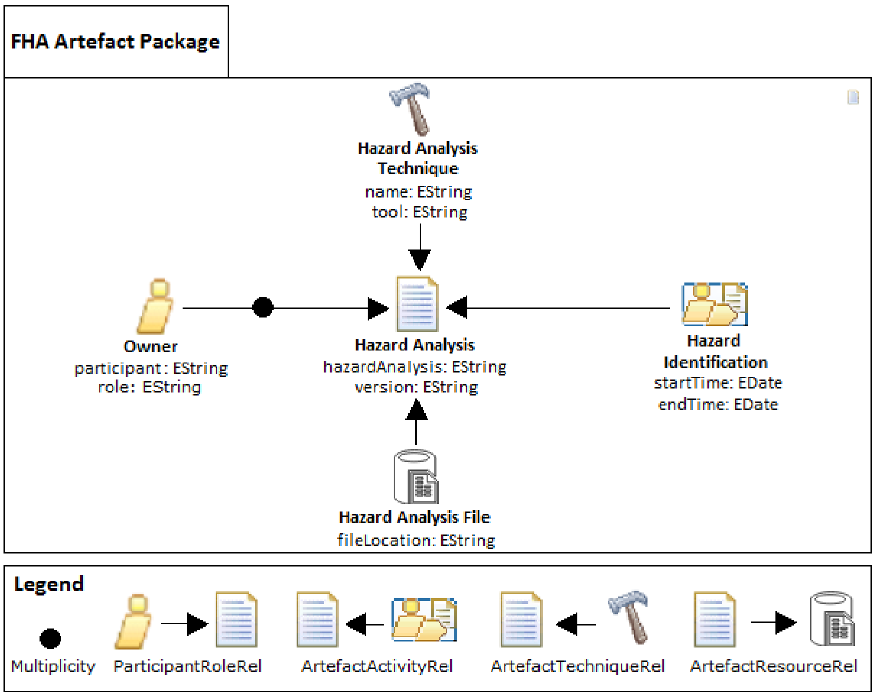
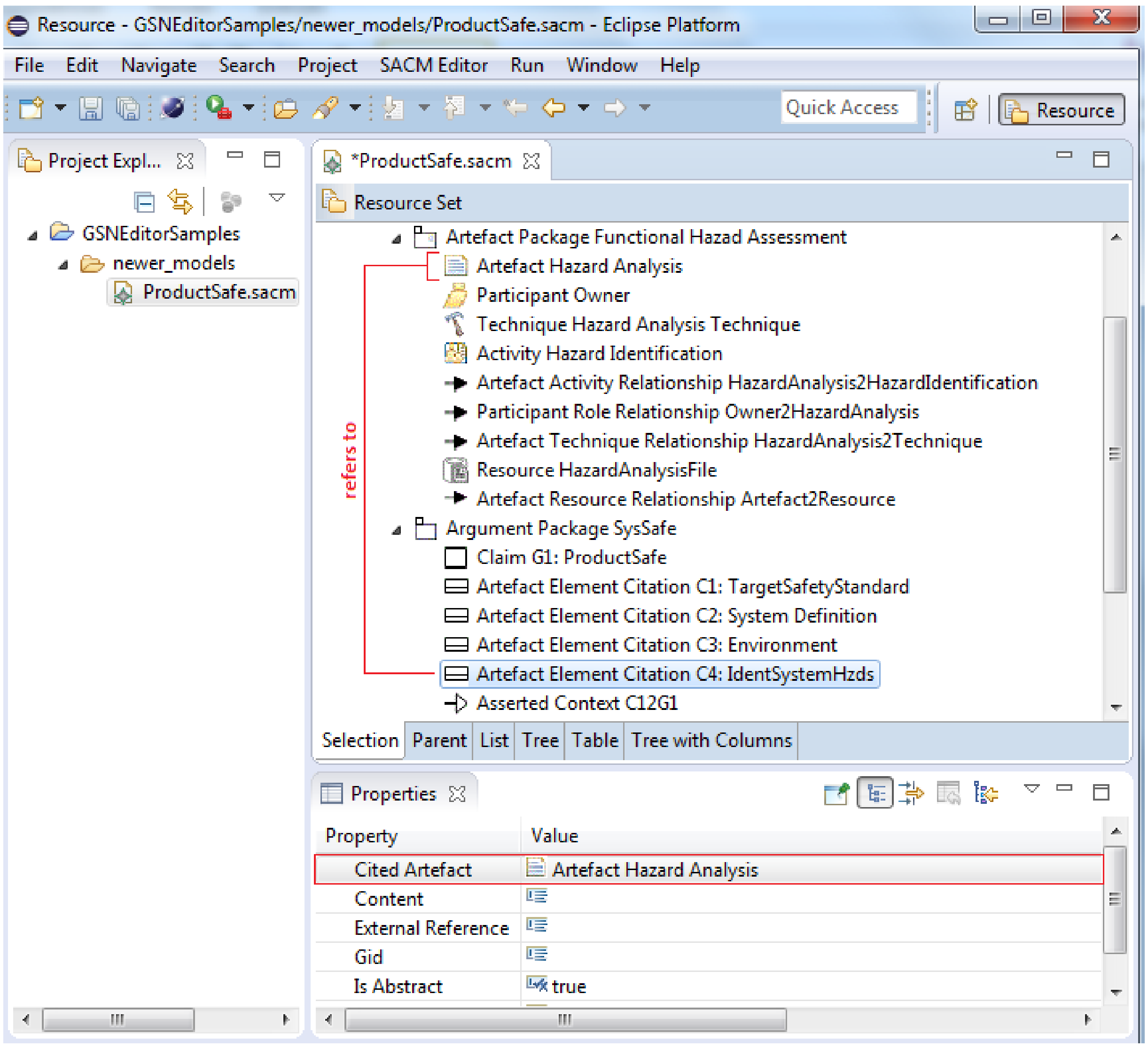


Figure 7.13. Functional hazard assessment artefact pattern diagram view.

Figure 7.14 shows the specification of the relationship between "*Hazard Analysis*" artefact and the reasoning embedded in the Hazard Avoidance (KELLY and McDERMID, 1997) assurance case pattern via "*C4: IdentSystem Hzds*" artefact element citation. It implies that an instance of a "*Hazard Analysis*" artefact provides the supporting evidence for an instance of Hazard Avoidance pattern.


Resource - GSNEditorSamples/newer_models/ProductSafe.sacm - Eclipse Platform
File Edit Navigate Search Project SACM Editor Run Window Help
Quick Access
Resource
Project Expl...
GSNEditorSamples
newer_models
ProductSafe.sacm
*ProductSafe.sacm
refers to
Selection Parent List Tree Table Tree with Columns
Properties

Resource Set
- Artefact Package Functional Hazard Assessment
  - Artefact Hazard Analysis
  - Participant Owner
  - Technique Hazard Analysis Technique
  - Activity Hazard Identification
  - Artefact Activity Relationship HazardAnalysis2HazardIdentification
  - Participant Role Relationship Owner2HazardAnalysis
  - Artefact Technique Relationship HazardAnalysis2Technique
  - Resource HazardAnalysisFile
  - Artefact Resource Relationship Artefact2Resource
- Argument Package SysSafe
  - Claim G1: ProductSafe
  - Artefact Element Citation C1: TargetSafetyStandard
  - Artefact Element Citation C2: System Definition
  - Artefact Element Citation C3: Environment
  - Artefact Element Citation C4: IdentSystemHzds
  - Asserted Context C12G1

Property	Value
Cited Artefact	Artefact Hazard Analysis
Content	
External Reference	
Gid	
Is Abstract	true

Figure 7.14. Linking assurance case pattern elements to artefact patterns.

7.4.3 HBS Assurance Case and Artefact Pattern Instantiation

The specification of traceability links between argument and evidence management information in SACM 2.0 assurance case and artefact patterns contributes to increase the confidence on the system assurance case generated with the support of Model-Based Assurance Cases approach (HAWKINS *et al.* 2015). The output of SACM 2.0 pattern instantiation process is an SACM assurance case model enhanced with artefact pattern instantiation information, i.e., an assurance case with information about the management of safety evidence referenced in the assurance case.

In the same way as the original MBAC instantiation program, the SACM-MBAC instantiation program requires the following input artefacts: an SACM 2.0 assurance case and artefact patterns specification, a set of system models and their respective metamodels, and the weaving model. The SACM-MBAC instantiation program was used to generate an SACM 2.0 assurance case model for the HBS-SPL four wheel braking (4WB) system variant. "*Hazard Avoidance*", "*Risk Argument",* and *"Absence of Hazardous Software Failure Mode"* assurance patterns presented in Chapter 6 (Section 6.3.1) were transformed into equivalent SACM 2.0 specifications using the GSN2SACM transformation plugin shown in Section 7.3.1.2. After performing model transformations, precedence control and data dependence constraints were added to the specification of these patterns. In the following, "*Functional Hazard Assessment"* and "*Fault Tree Analysis"* artefact patterns were added to the SACM 2.0 assurance case pattern specification. The weaving model shown in Chapter 6 (Section 6.3.3) was updated with references to abstract "*Terms*" required to instantiate artefact patterns and their links to system model elements. References to abstract "*Terms"* associated with artefact patterns, e.g., "*tool", "participant", "hazardAnalysis"*, and "*fileLocation"* shown in Figure 7.12, and their mappings to system model elements were added to the weaving model. In this case, feature, context, architecture, failure, and fault tree and FMEA system models were considered.

Figure 7.15 shows the modular view of the instantiated SACM assurance case model for the 4WB braking system variant. The difference between SACM assurance case model and its equivalent representation in GSN presented in Chapter 6 (Section 6.3.4) is the explicit traceability links between argument and evidence management artefacts that substantiate assurance claims embedded in the assurance case. For example, claims arguing the risk posed by hazards associated with 4WB system variant embedded into risk argument packages are substantiated by *"4WB FHA"*, "*NoBraking4WheelsFTA"*, *"NoBraking3WheelsFTA", "ValueBrakingFTA", "NoBrakingFrontFTA", "NoBrakingRear-FTA"*, and *"NoBraking-DiagonalFTA"* artefact packages. *"4WB FHA"* artefact package contains evidence management information for the provenance of the "*4WB*" hazard analysis artefact. This artefact package shows the "*4WB*" hazard analysis artefact and its relationships with safety analysis techniques/tools used to support the analysis, e.g., model-based or a manual technique, safety assessment activities associated with the construction of the hazard analysis artefact, the personnel involved in the process as well as different source formats in which the artefact is available, e.g., tables, a file, a html report.

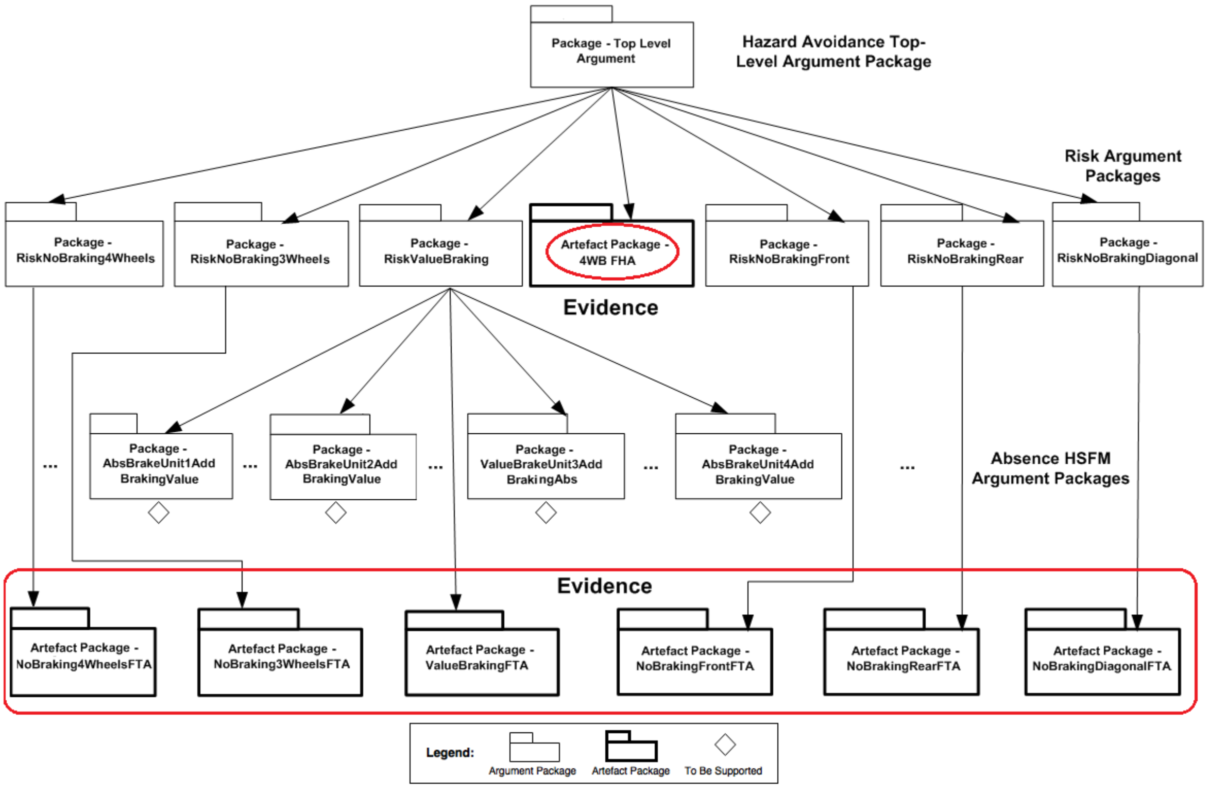
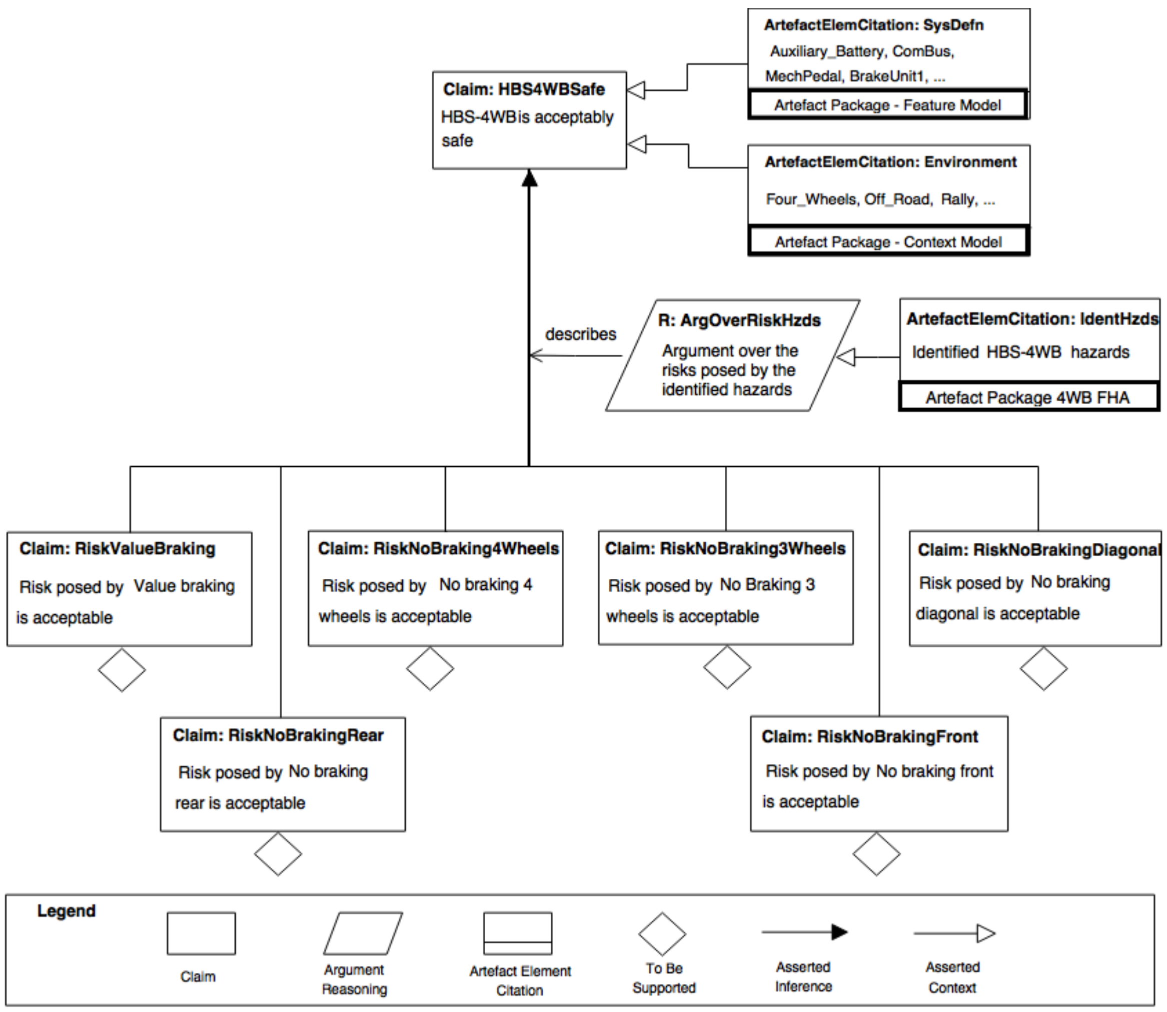

Package - Top Level Argument
Hazard Avoidance Top-Level Argument Package
Risk Argument Packages
Package - RiskNoBraking4Wheels
Package - RiskNoBraking3Wheels
Package - RiskValueBraking
Artefact Package - 4WB FHA
Evidence
Package - RiskNoBrakingFront
Package - RiskNoBrakingRear
Package - RiskNoBrakingDiagonal
Package - AbsBrakeUnit1Add BrakingValue
Package - AbsBrakeUnit2Add BrakingValue
Package - ValueBrakeUnit3Add BrakingAbs
Package - AbsBrakeUnit4Add BrakingValue
Absence HSFM Argument Packages
Evidence
Artefact Package - NoBraking4WheelsFTA
Artefact Package - NoBraking3WheelsFTA
Artefact Package - ValueBrakingFTA
Artefact Package - NoBrakingFrontFTA
Artefact Package - NoBrakingRearFTA
Artefact Package - NoBrakingDiagonalFTA
Legend:
Argument Package
Artefact Package
To Be Supported

Figure 7.15. 4WB assurance case modular view in SACM 2.0.

Assurance claims arguing the absence of the identified contributing failure modes to the occurrence of each system-level hazard are substantiated by instances of *Fault Tree Analysis* artefact pattern. These claims are embedded into risk argument packages and their supporting "*Absence HSFM*" argument packages, which provide the provenance of fault tree analysis evidence items referenced in these argument packages and their sub-packages. The provenance information for the evidence referenced in the assurance case is important to clearly communicate the evidentiary support for assurance claims, increasing the confidence in the assurance case. It contributes to the acceptance of the assurance case by certifying authorities in order to obtain certification credits for the target safety-critical system.

Figure 7.16 illustrates the instantiated Hazard Avoidance argument pattern (Product Safe in Figure 7.11) for the "*4WB*" system variant. The assurance case decomposes the safety claim arguing that "*4WB*" system variant is acceptably safe into sub-claims, based on the reasoning that the risk posed by the identified hazards are acceptable ("*R: ArgOverRiskHzds*"*)*. The top level assurance claim is substantiated by "*Feature Model*" and "*Context Model*" artefact packages, denoted by "*Artefact Element Citations*" linked to the "*HBS4WBSafe*" top-level claim. The sub-claims arguing the risk posed by the identified sys-



ArtefactElemCitation: SysDefn
Auxiliary_Battery, ComBus, MechPedal, BrakeUnit1, ...
Artefact Package - Feature Model
Claim: HBS4WBSafe
HBS-4WBis acceptably safe
ArtefactElemCitation: Environment
Four_Wheels, Off_Road, Rally, ...
Artefact Package - Context Model
describes
R: ArgOverRiskHzds
Argument over the risks posed by the identified hazards
ArtefactElemCitation: IdentHzds
Identified HBS-4WB hazards
Artefact Package 4WB FHA
Claim: RiskValueBraking
Risk posed by Value braking is acceptable
Claim: RiskNoBraking4Wheels
Risk posed by No braking 4 wheels is acceptable
Claim: RiskNoBraking3Wheels
Risk posed by No Braking 3 wheels is acceptable
Claim: RiskNoBrakingDiagonal
Risk posed by No braking diagonal is acceptable
Claim: RiskNoBrakingRear
Risk posed by No braking rear is acceptable
Claim: RiskNoBrakingFront
Risk posed by No braking front is acceptable
Legend
Claim
Argument Reasoning
Artefact Element Citation
To Be Supported
Asserted Inference
Asserted Context

Figure 7.16. HBS-4WB argument package.

tem hazards are substantiated by “*4WB FHA*” artefact package, as stated in the “*IdentHzds”* “*Artefact Element Citation”* element.

The claims arguing that the risk posed by the identified system hazards are acceptable are further supported by “*Risk”* argument packages, which decompose these claims into sub-claims arguing the absence of the contributing failure modes identified in fault tree analysis artefacts. The “*IdentHzds” Artefact Element Citation* explicitly links sub-claims arguing that the risk posed by system hazards are acceptable to the supporting evidence embedded into “*4WB FHA”* artefact package. The “*4WB FHA”* artefact package, illustrated in Figure 7.17, provides the provenance for the hazard log artefact referenced into the top-level argument package. This artefact is owned by two safety analysts who developed it with the support of “*HaZard and OPerability Study analysis”* technique and “*HiP-HOPs”* compositional safety analysis tool. The hazard log artefact was created during the “*Hazard Identification”* activity

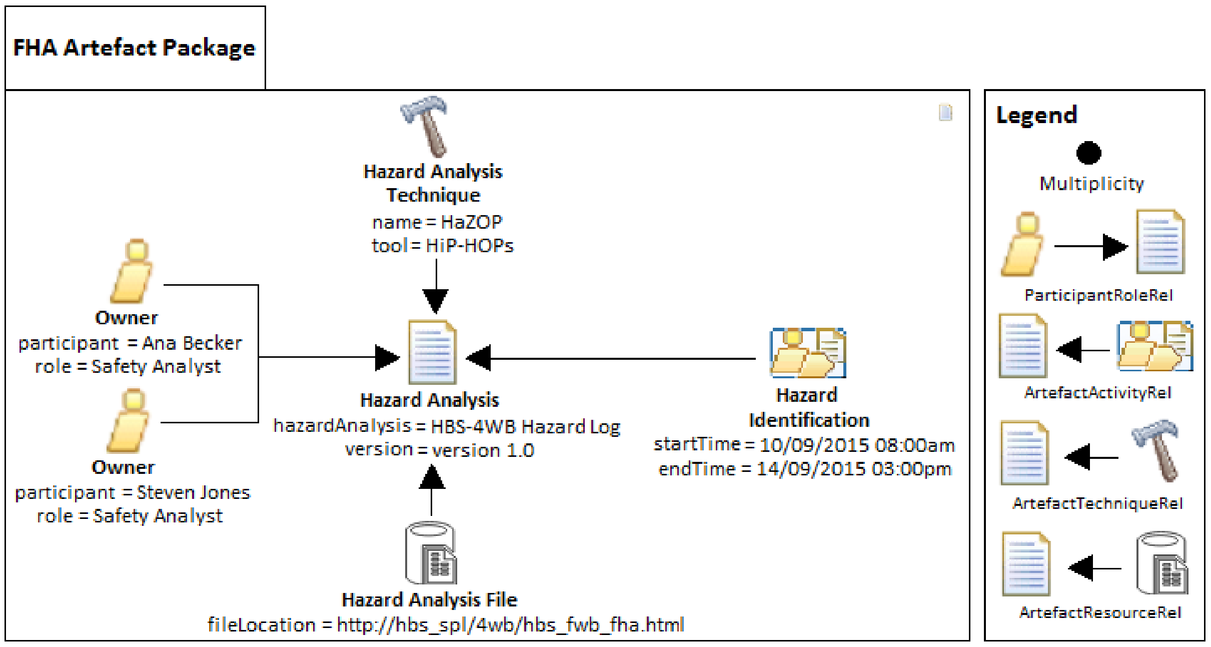


Figure 7.17. 4WB FHA artefact package.

initiated by the owners in “10 Set of 2015” and finished in “14 Set 2015”. The hazard log artefact is available in the form of hyperlinked web pages.

Following the reasoning of the top-level argument package, “*RiskNoBraking4Wheels*” claim is supported by “*Risk No Braking 4 Wheels*” argument package shown in Figure 7.18. In this argument package, the top-level claim is stated in the context of ISO 26262 ASIL 4 safety integrity requirement, and “*omission of four wheel brake unit outputs*” failure condition. The “*4WB FHA*” artefact package provides the evidentiary support for this claim. Therefore, the “*Claim 1*”is decomposed into sub-claims (claims 2, 3, 4, and 5) arguing the absence of failure modes of components which contribute to the occurrence of “*No Braking 4 Wheels*” system hazard. For example, “*Claim 2*”argues the absence of value failure into “*Brake_Unit1*” component outputs. This claim is supported in another argument package, which decomposes the “*Claim 2*” into sub-claims arguing the absence of primary, secondary, and control failure modes that contribute to an incorrect value in “*Brake_Unit1*” component outputs.

The “*4WB FTA*” artefact package provides the supporting evidence for the top-level claim from “*RiskNoBraking4Wheels*” argument package. Finally, the “*NoBraking4Wheels-FTA*” artefact package referenced into “*C3*” artefact element citation provides the supporting evidence for assurance of the sub-claims stated into this argument package.

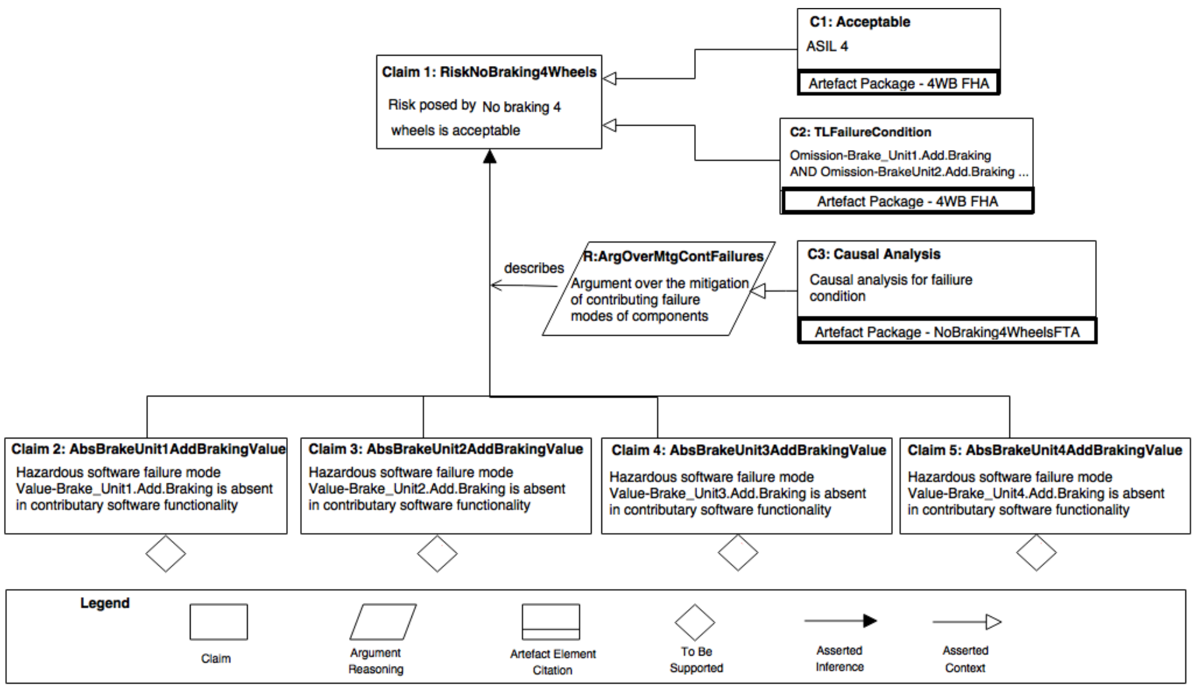

C1: Acceptable
ASIL 4
Artefact Package - 4WB FHA
Claim 1: RiskNoBraking4Wheels
Risk posed by No braking 4 wheels is acceptable
C2: TLFailureCondition
Omission-Brake_Unit1.Add.Braking AND Omission-BrakeUnit2.Add.Braking ...
Artefact Package - 4WB FHA
describes
R:ArgOverMtgContFailures
Argument over the mitigation of contributing failure modes of components
C3: Causal Analysis
Causal analysis for failiure condition
Artefact Package - NoBraking4WheelsFTA
Claim 2: AbsBrakeUnit1AddBrakingValue
Hazardous software failure mode Value-Brake_Unit1.Add.Braking is absent in contributary software functionality
Claim 3: AbsBrakeUnit2AddBrakingValue
Hazardous software failure mode Value-Brake_Unit2.Add.Braking is absent in contributary software functionality
Claim 4: AbsBrakeUnit3AddBrakingValue
Hazardous software failure mode Value-Brake_Unit3.Add.Braking is absent in contributary software functionality
Claim 5: AbsBrakeUnit4AddBrakingValue
Hazardous software failure mode Value-Brake_Unit4.Add.Braking is absent in contributary software functionality
Legend
Claim
Argument Reasoning
Artefact Element Citation
To Be Supported
Asserted Inference
Asserted Context

Figure 7.18. Risk no braking four wheels argument package.

7.5 Discussion and Limitations

Although there are benefits related to the management of constraints in assurance case and artefact pattern instantiation, and traceability between assurance case and evidence management artefacts provided by the proposed SACM Model-Based Assurance Cases approach, the following limitations have been identified:

- *Weaving modeling*: whereas the correct definition of the weaving model is crucial to the success of both original and SACM-MBAC approaches, and the manual specification of the weaving model is error prone, further work needs to be done to develop a weaving modeling tool integrated to assurance cases, and system modeling in a single platform, e.g., EMF;
- *Assurance case pattern instantiation*: the lack of mechanisms for handling choices in the SACM assurance case and artefact pattern instantiation. For example, the "*Contributions to Hazardous Software Failure Modes*" assurance case pattern catalogue (WEAVER, 2003) comprises pattern choices in which one or more patterns are chosen, according to the value provided to instantiate abstract "*Terms*" defined in these patterns;

- *Advanced model management*: the lack of mechanisms for assurance case model validation against predefined internal constraints, e.g., the coverage of a claim by the available evidence, and external constraints related to the faithfulness of the referenced evidence against actual project data (HAWKINS *et al.* 2015);
- *Scalability:* the evaluation presented in this chapter is focused in showing the feasibility of the proposed SACM-MBAC approach to generate explicit traceability links between assurance claims and evidence management artefacts. Further work should be done to evaluate the SACM-MBAC within other safety-critical systems (e.g., Tiriba Flight Control System), assurance case patterns (e.g., MISRA Working Group on Automotive Safety Cases and ISO 26262), and artefact patterns (e.g., architecture, test cases); and
- *Tooling support*: the SACM-MBAC prototype tool is limited to handle a subset of data dependence pattern instantiation constraints, and further work must be done to improve the assurance case and artefact pattern instantiation processes implemented in its current version.

7.6 Summary

This chapter has presented the SACM-MBAC approach to overcome the limitations from the original GSN model-based assurance cases approach related to the management of assurance case pattern instantiation constraints and integration of evidence management patterns into the MBAC process. The SACM-MBAC supports explicit traceability between assurance claims and evidence management information associated with artefacts referenced in the assurance case. Such traceability supports clear communication of the evidentiary support for assurance claims, increasing the confidence in the assurance case and contributing for its acceptance as certification evidence by certifying authorities. The following chapter presents the overall evaluation of the thesis contributions, and a case study that provides further validation for the thesis contributions presented in Chapter 4 (Variability Management in Product Line Architecture and Safety Models), Chapter 5 (Compositional Safety Analysis and Design Optimization), and Chapter 6 (Product Line Model-Based Assurance Cases).

Chapter 8

EVALUATION

8.1 Introduction

This chapter presents the evaluation of the thesis contributions presented in chapters 4, 5, 6 and 7 and a case study in the aerospace domain. This evaluation was performed on the basis of the thesis hypothesis defined in Chapter 1:

> *Through adopting a model-based approach for managing and* ***tracing*** *variability in architectural and safety models it is* ***feasible*** *to support the* ***traceable*** *and* ***systematic*** *construction of safety assessment and assurance case artefacts within a product line engineering approach.*

The terms ***feasible, tracing/traceable,*** and ***systematic*** provide the criteria to evaluate the thesis contributions related to model-based approaches to support: management and resolution of variability in architecture and safety models (Chapter 4), compositional safety analysis and design optimization (Chapter 5), and model-based assurance cases in product line engineering for safety-critical systems (Chapters 6 and 7). This chapter examines the ***feasibility*** of applying these model-based approaches to support software product line engineering in automotive and aerospace domains. This chapter also examines the potential of the proposed approaches in ***tracing*** product line context and design variation throughout architecture and safety analysis models in domain engineering to support the systematic reuse of these assets in application engineering. In application engineering, it is examined whether the proposed model-based approaches can support: the ***traceability*** of variant-specific requirements throughout architecture and safety analysis models, and the ***systematic*** generation of safety assessment artefacts from the reused variant-specific architecture and safety analysis assets. This chapter also examines whether the product line compositional safety analysis and design optimization approach (Chapter 5) can ***systematically*** support allocation and decomposition of safety integrity requirements throughout product line components in order to reduce the effort and costs of safety-critical product line development processes, contributing to achieve product line process-based certification. By ***systematic*** it is also examined whether the product line model-based assurance cases (Chapter 6) addresses the requirements for trustworthy product safety assessment and assurance cases in application engineering by supporting the automatic generation of a variant-specific assurance case from a diverse set of variant-specific development and safety assessment models, thus, linking assurance case claims to evidence.

In addition to the aerospace case study presented in this chapter, different forms of evaluation ranging from peer review to case studies were considered throughout this thesis to assess the research contributions. These forms of evaluation are detailed in the following section, followed by the description of the aerospace case study phases (Sections 8.3 and 8.4) and their outcomes (Section 8.5), the analysis of the thesis contributions (Section 8.6), and the summary of this chapter (Section 8.7).

8.2 Forms of Evaluation

The following forms of evaluation were used to assess the research contributions throughout this thesis:

- Case Studies;
- Peer Review;
- Tool Support;
- Experimental Study.

The forms of evaluation mentioned above address one or more of the following research criteria: *feasibility*, *traceability*, and *systematic* defined in Section 8.1. Table 8.1 shows the relationships between the forms of evaluation and the thesis contributions. Peer review was considered to evaluate the technical consistency of the phases, activities and tasks defined in the proposed model-based approaches to support management and resolution of va-

Table 8.1 – Forms of evaluation and thesis contributions.

Thesis Contrib./Evaluation	***Case Studies***	***Peer Review***	***Tool-Support***	***Experiment***
Variability Management in Product Line Architectural and Safety Models *(Chapter 4)*	Yes	Yes	Yes	N/A
Product Line Compositional Safety Analysis and Design Optimization *(Chapter 5)*	Yes	Yes	Yes	N/A
Product Line Model-Based Assurance Cases *(Chapter 6)*	Yes	Yes	Yes	Yes
Pattern Instantiation Constraints and Artefact Patterns in Model-Based Assurance Cases *(Chapter 7)*	Yes	Yes	Yes	N/A

riability in architectural and safety analysis models, product line compositional safety analysis and design optimization, and assurance case construction. On the other hand, case studies outcomes have shown the *feasibility* of the thesis contributions in supporting software product line engineering of safety-critical systems.

8.2.1 Case Studies

The following case studies were carried out during the course of this research:

- Hybrid Braking System Product Line Variability Management Case Study (Chapter 4);
- Hybrid Braking System Product Line Compositional Safety Analysis and Design Optimization Case Study (Chapter 5);
- Hybrid Braking System Product Line Model-Based Assurance Cases Case Study (Chapters 6 and 7).

These case studies are detailed along with the mentioned chapters and are considered as the first evaluation for the corresponding research contributions.

8.2.2 Aerospace Case Study

This case study has shown the feasibility of applying the thesis contributions in an aerospace flight control product line. The case study provides strong evidence of the feasibility of the proposed model-based approaches to support the management and resolution of architectural and safety-related variability, compositional safety analysis and design optimization, and assurance case construction in safety-critical software product line engineering processes. Section 8.3 presents the aerospace case study.

8.2.3 Peer Review

The research contributions presented in this thesis were reviewed by academic researchers and system engineers in the industry. The research contributions were presented and discussed with researchers from: Software Engineering Group at Mathematics and Computer Science Institute from the University of São Paulo, High-Integrity System Engineering (HISE) research group from the University of York, and Dependable Systems research group from the University of Hull. Further, the research was presented and discussed with system engineers and safety engineers from the University of Hull who developed the HiP-HOPS compositional safety analysis tool.

8.2.4 Tool Support

In this thesis, the following tools were developed to support the research contributions presented in chapters 4, 5, 6, and 7:

- Adapters to BVR variant management tool (VASILEVSKIY *et al.* 2015) were developed to integrate the management and resolution of variability into MATLAB/Simulink/HiP-HOPS, and OSATE AADL/Error Annex models. Hephaestus/Simulink tool was also extended to support the management and resolution of variability in safety analysis embedded into Simulink models enhanced with HiP-HOPS failure annotations (Chapter 4).
- Product Line Component SIL decomposition (PL-SILDec) tool was developed as an extension of HiP-HOPS ASIL and DAL decomposition tools (SOROKOS *et al.* 2015; AZEVEDO *et al.* 2014) to support the allocation of safety integrity requirements to

product line components from the analysis of multiple variant-specific SIL decomposition results. Such analysis supports product line engineers and safety engineers in establishing the safety objectives, development and safety assessment processes according to the integrity of individual product line components in order to achieve process-based certification (Chapter 5);

- A set of metamodels and models were created to support model-based assurance cases into software product line engineering. The Eclipse Modeling Framework platform was used to create the following metamodels: MATLAB/Simulink metamodel, HiP-HOPS Failure metamodel, Feature and Context metamodel, Fault Tree/FMEA metamodel, and SIL Allocation metamodel (Chapter 6);
- The Eclipse Modeling Framework was also used to create a diagram editor for the OMG Structured Assurance Case Metamodel version 2.0 (OMG, 2015a), and Epsilon Transformation Language (KOLOVOS *et al.* 2013) was used to implement a GSN to SACM 2.0 model transformation tool. These tools were developed to support the specification of pattern instantiation constraints in assurance case patterns, and the specification of artefacts patterns and their relationships with assurance case patterns into Model-Based Assurance Cases process (Chapter 7).

8.2.5 Experimental Study

An experimental study was conducted to evaluate the feasibility of applying the product line Model-Based Assurance Cases approach to support the automatic generation of variant-specific assurance cases in software product line engineering processes. The results of this experimental study have shown that model-based assurance cases approach is more effective and efficient in supporting assurance case construction into safety-critical software product line engineering than conventional techniques for assurance case construction. The experimental study is detailed in Chapter 6.

8.3 Case Study: Tiriba UAV Flight Control Product Line

8.3.1 Scope

The aerospace case study encompasses the flight control system from Tiriba Unmanned Aircraft System (BRANCO *et al.* 2011). Tiriba is a small-size electrical unmanned aircraft vehicle developed by AGX[22] Company and the National Institute of Science and Technologies for Critical Embedded Systems (INCT-SEC) (INCT-SEC, 2014). Tiriba system was developed using MATLAB/Simulink environment. An extractive product line development strategy was adopted to transform the Tiriba flight control system architecture into a software product line. The Tiriba Flight Control product line (TiribaFC-SPL) comprises four system variants related to the pilot mode: "*Manual Pilot*", which comprises a human operator sending commands to the unmanned aircraft from a ground control station; "*Autopilot*", which executes a pre-defined route; "*Assisted Mode*" that allows the operator to send commands to the UAV configured with "*Autopilot*" mode; and "*Autonomous Pilot*" intended to perform actions according to the current environmental conditions. With the exception of "*Manual Pilot*" mode, both pilot modes are optional. Such variation allows the derivation of several different system variants for the provision of the aircraft flight control capability. The pilot modes considered for the aircraft control may change according to the UAV size (e.g., small or light), the target application, which in the case of Tiriba UAV can be agriculture, environment monitoring, or defense, and the target airspace, which can be controlled or uncontrolled. Whereas different failure conditions may lead or contribute to the occurrence of hazards related to flight control capability in different product variants, the Tiriba flight control is considered a safety-critical product line. In this case study, the safety analysis was delimited to four TiribaFC-SPL system variants considered to operate into two different contexts defined by combining application, UAV size, and airspace contextual elements. This case study was performed by the author over an effort of a three months period.

[22] http://www.agx.com.br/n2/pages/index.php

8.3.2 Problem Statement

With the increase of the complexity of safety-critical systems in automotive and aerospace domains, achieving the systematic reuse of the product line architecture and safety analysis demands the automated traceability of context and design variation specified in functional and context features to their realization in architectural and safety analysis assets. The correct management of variability in architecture and safety models in product line domain engineering is required to enable the systematic reuse of product line architecture and safety analysis in application engineering. The reused safety assets contribute to reduce the effort and complexity in performing safety analysis for a specific product variant in application engineering. Integrating compositional safety analysis into product line engineering processes should contribute to reduce the effort in generating expensive safety assessment artefacts such as fault tree analysis and FMEA results required for certifying a given product variant. Achieving the automated traceability between development and assessment artefacts, and the assurance case is also required in safety-critical systems product line engineering processes. Therefore, context and design variation should be traced to architecture, safety analysis and assessment assets, and assurance cases. Changes in product requirements should be further propagated throughout architectural and safety models. This is necessary because ignoring the impact the context and design variation in safety assessment and assurance cases may weaken the confidence or invalidate the assurance case (HABLI, 2009).

8.3.3 Objectives

The objective of the TiribaFC-SPL case study is to evaluate whether the proposed model-based approaches for variability management, safety assessment, assurance case construction in safety-critical product lines provide automated support for: management and resolution of variability into TiribaFC-SPL architectural and safety models in domain engineering, and systematic reuse and generation of safety artefacts in product line application engineering.

8.3.4 Methodology

This section presents the notations, tools, and model-based techniques used to carry out the TiribaFC-SPL case study.

8.3.5 Notations

- Feature-Oriented Domain Analysis (KANG *et al.* 1990) was used to specify product line feature and context models;
- SysML (OMG, 2015) was used to represent the product line architecture model by means of *Block Definition Diagram*;
- HiP-HOPS (PAPADOPOULOS *et al.* 2011) notation was used to specify the product line failure model;
- GSN and its Pattern and Modular extensions were used to support the specification of variant-specific assurance cases.

8.3.6 Tooling

- Feature IDE Eclipse-based modeling tool was used to support product line feature and context modeling;
- Microsoft Visio and its SysML Stencil and GSN plug-ins were used to support the specification of product line architecture model and assurance cases;
- MATLAB/Simulink model-based development environment was used to support the design of the product line architecture;
- HiP-HOPS compositional safety analysis tool was used to support the specification of the product line failure model in domain engineering. In application engineering, HiP-HOPS was used to support variant-specific failure modeling, and automatic synthesis of fault trees and FMEA results. The HiP-HOPS Tabu Search SIL decomposition optimization tool was used to support automatic decomposition of safety integrity requirements in different product variants;

- Product line component SIL decomposition (PL-SILDec) tool developed in the course of this thesis was used to support the decomposition of SILs to product line components early on the design;
- BVR variant management tool was used to support the management and resolution of variability in architecture and safety analysis embedded into MATLAB/Simulink and HiP-HOPS failure models;
- Model-Based Assurance Cases (MBAC) tool (HAWKINS *et al.* 2015) was used to support the generation of variant-specific assurance cases from a diverse set of system models.

8.4 Case Study Phases

The TiribaFC-SPL case study was carried out into three major phases 1) management and resolution of variability into architectural and safety models, 2) product line compositional safety analysis and design optimization, and 3) product line model-based assurance cases. Each major phase encompasses a set of phases performed in product line domain engineering and application engineering.

8.4.1 Management and Resolution of Variability in Architecture and Safety Models

In domain engineering, during the "*SC-PLE-1: Product Line Requirements Elicitation*" phase, by adopting an extractive product line development strategy, product line feature and context models were defined from the analysis of the Tiriba flight control system and its usage context. Feature and context models define the permitted functional and contextual variation for the Tiriba flight control system. In the "*SC-PLE-2: Product Line Architectural Design*" phase, no changes were performed into the original Tiriba flight control system, since its architecture model already provides variability mechanisms for pilot mode functions, represented by MATLAB/Simulink "*Switch*" blocks (BOTTERWECK *et al.* 2010). Still in domain engineering, hazards and their risk associated with Tiriba pilot mode variants in a range of usage context were identified during the "*SC-PLE-3: Product Line*

Safety Analysis" phase. Safety integrity requirements, in terms of ARP 4754a Development Assurance Levels (EUROCAE, 2010), were also allocated minimize the effects of the occurrence of each one of these hazards. Contributing component failures to the occurrence of these hazards were also identified during component failure analysis. Thus, the product line failure model was obtained. In the "*SC-PLE-4: Variability Realization Modeling"* phase, functional and context features were linked to their realization in architectural components and safety analysis data, with the support of the BVR variant management tool integrated to compositional safety analysis techniques (VASILEVSKIY *et al.* 2015). Therefore, the automated traceability between product line feature, architecture, and failure models is achieved in the variability realization model. In application engineering, the resolution model was obtained by selecting functional and context features from the TiribaFC-SPL feature model, and variant-specific flight control architecture and failure models were automatically derived with the support of the BVR toolset. The detailed description of the artefacts produced in this major phase is available in Appendix D.

8.4.2 Product Line Compositional Safety Analysis and Design Optimization

Compositional safety analysis and design optimization into product line domain engineering aim to identify the threats to the TiribaFC-SPL safety (i.e., hazards), their causes, and allocating safety requirements to eliminate or minimize the effects of system-level failures in the "*SC-PLE-3: Product Line Safety Analysis"* phase, as aforementioned in Section 8.4.1. TiribaFC-SPL safety analysis activities were performed from the perspective of the following pilot mode variants: "*Manual and Autonomous Pilot"* (TFC-MAT), "*Manual and Assisted Pilot"* (TFC-MAS), "*Manual and Autopilot"* (TFC-MAP), and "*All Pilot Modes"* (TFC-ALL). TFC-MAT and TFC-MAS system variants were assumed to be deployed into a *Light UAV*, to operate in an "*Uncontrolled Airspace"*, and to be used in "*Environment Monitoring"* applications. TFC-MAP and TFC-ALL system variants were assumed to be deployed into a *Light UAV*, to operate in a "*Controlled Airspace"*, and to be used in "*Defense"* applications. For each one of these scenarios, hazards, their causes, safety integrity requirements, and contributing component failure modes were identified. The analysis was constrained to these variants and context to reduce the complexity and facilitate the understanding of the concepts presented in the product line compositional safety analysis and design optimization approach proposed in this thesis.

In application engineering, after the "*SC-PLE-6 - Product Derivation*" phase, from the generated variant-specific architecture and failure models, fault trees and FMEA results were generated with the support of HiP-HOPS compositional safety analysis tool (PAPADOPOULOS et al. 2011). At the "*SC-PLE-8 - Product SIL Decomposition*" phase, for each system variant, safety integrity requirements allocated to hazards associated with a given TiribaFC-SPL system variant were automatically decomposed throughout contributing component failure modes with the support of the HiP-HOPS Tabu Search DAL decomposition optimization extension (SOROKOS *et al.* 2015). This tool is an extension of HiP-HOPS Tabu Search ASIL decomposition algorithm (AZEVEDO *et al.* 2014) to address aerospace systems. DAL decomposition was performed for each one of the following four TiribaFC-SPL system variants: TFC-MAT, TFC-MAS, TFC-MAP, and TFC-ALL assumed in the usage context mentioned earlier in this section. These multiple DAL decomposition results are input artefacts to the "*Product Line Component SIL Decomposition*" phase.

The extension to HiP-HOPS design optimization tooling to support the allocation of safety integrity requirements to product line components developed in this thesis (see Chapter 5) was used to automate the analysis of multiple DAL decomposition results to derive the DALs to be allocated to TiribaFC-SPL components to address product line process-based certification. The allocated DALs guide product line engineers and safety analysts in establishing safety objectives, and defining development, verification, validation, testing, and safety assessment processes, to achieve product line process-based certification in compliance with ARP 4754a and DO-178C (RTCA, 2011) standards. These activities were allocated to TiribaFC-SPL components according to their integrity. Product line component SIL decomposition allows achieving process-based certification for TiribaFC-SPL components without being expensive or stringent.

8.4.3 Product Line Model-Based Assurance Cases

The integration of Model-Based Assurance Cases approach (HAWKINS *et al.* 2015) into product line engineering processes supports the automated traceability between TiribaFC-SPL variant-specific assurance cases and their respective development and assessment models. Thus, changes in the TiribaFC-SPL product variant are automatically propagated throughout the assurance case. In product line application engineering, the assurance case pattern specification, the weaving model, and product line asset metamodels described in Chapter 6, together with TiribaFC-SPL variant-specific models are input artefacts to configure the model-based assurance cases tooling (HAWKINS *et al.* 2015) to generate an assurance case for the given TiribaFC-SPL system variant. This process was repeated to generate assurance case models for each one of the following TiribaFC-SPL variants: "*Manual and Autonomous Pilot*", "*Manual and Assisted Pilot*", "*Manual and Autopilot*", and "*All Pilot Modes*". The traceability of assurance case elements to variant-specific models provided by model-base assurance cases increased the confidence of the generated assurance case model. Such traceability complies with ARP 4754a and DO-178C guidance related to the traceability between development, assessment assets, and the assurance case. Therefore, variant-specific system models and assurance cases can be used as evidentiary documentation to support goal-based and process-based certification of TiribaFC-SPL system variants against ARP 4754A and DO-178C aerospace safety standards.

8.5 Tiriba UAV Case Study Outcomes

The following deliverables were produced during the management and resolution of variability in architectural and safety models, compositional safety analysis and design optimization, and model-based assurance cases major phases of the TiribaFC-SPL case study:

- Domain Engineering:
    - TiribaFC-SPL Feature and Context Models;
    - TiribaFC-SPL Architecture Model;
    - TiribaFC-SPL Failure Model (i.e., Hazard and Risk, and Component Failure Analysis);
    - Variability Realization Model; and
    - TiribaFC-SPL Component DAL Decomposition Results.
- Application Engineering:
    - TiribaFC-SPL Variant-Specific Feature Models;
    - TiribaFC-SPL Variant-Specific Architecture and Failure models;
    - Variant-Specific Fault Trees and FMEA Models;
    - Variant-Specific DAL Decomposition Results; and

- Variant-Specific Assurance Cases.

Traceability links between product line variation expressed in functional and context features, and their realization into architectural components, and safety analysis were established in the variability realization model produced with the support of BVR variant management toolset. The variability realization model provides support for further resolution of variability in architectural and safety models in the product derivation phase. Therefore, changes into variant-specific feature and context models are automatically propagated throughout the reused architecture and failure models, and generated safety assessment artefacts such as product fault trees, FMEA results, DAL Decomposition results, and the assurance case. The deliverables produced in each phase of the TiribaFC-SPL case study are detailed in Appendix D.

8.6 Analysis of the Thesis Contributions

This section presents the analysis and evaluation of the following research contributions resultant from this thesis:

- Variability Management in Product Line Architecture and Safety Models (Chapter 4);
- Product Line Compositional Safety Analysis and Design Optimization (Chapter 5);
- Product Line Model-Based Assurance Cases (Chapter 6); and
- Pattern Instantiation Constraints and Artefact Patterns in Model-Based Assurance Cases (Chapter 7).

Each research contribution was evaluated against the thesis hypothesis, presented in Section 8.1, using at least two forms of evaluation as shown in Table 8.1. The evaluation results for each thesis contribution are presented in the following sections.

8.6.1 Variability Management in Product Line Architecture and Safety Models

The contributions of this thesis regarding variability management in product line architecture and safety models have shown how context and design variation, defined in feature and context models, can be automatically traced to their realization into product line architectural and safety analysis. This was achieved by means of a domain and tool independent systematic approach to integrate variant management and compositional safety analysis techniques into software product line engineering processes. Applying such approach in automotive and aerospace case studies allowed the systematic reuse of architectural and safety analysis information, and automatic generation of safety assessment artefacts in product line application engineering, contributing to reduce the effort and costs of variant-specific safety assessment. Research contribution in this area has described how to adapt existing variant management techniques/tools to support the management and resolution of variability in safety models. Therefore, adapters for BVR variant management tool have been developed to integrate variant management into MATLAB-Simulink/HiP-HOPS and OSATE AADL/Error Annex model-based development and compositional safety analysis tools. Hephaestus/Simulink has also been adapted to support variability management in Simulink models with HiP-HOPS failure annotations. The contributions in this area were evaluated through: academic peer review, development of tool support by means of adapters to Hephaestus/Simulink and BVR variant management tools supporting variability management in safety models, Hybrid Braking System automotive product line case study, and Tiriba Flight Control product line aerospace case study.

The automated ***traceability*** of variation in functional and usage context features throughout architecture and safety analysis models has been shown by carrying out automotive and aerospace variability management case studies with the support of the BVR toolset and the developed adapters. The ***feasibility*** of the approach is shown by the development of tool support, in the form of adapters to BVR and Hephaestus/Simulink variant management tools. ***Feasibility*** has also been indicated by conducting automotive and aerospace variability management case studies, where it has been shown that variability in architectural and safety models were managed in the variability realization model produced in domain engineering, and variant-specific architecture and failure models were generated in application engineering. The reused architectural and safety analysis artefacts contributed to reduce the effort in performing variant-specific safety assessment to generate fault trees and FMEA artefacts with the support of compositional safety analysis techniques (Chapter 5). Automotive and aerospace case studies have shown that the approach is domain independent. Although this thesis has not presented a case study covering other representations for architectural and failure models, it was verified that the proposed approach is applicable to other model representations and variant management techniques. To evidence this claim, adapters to the BVR toolset were developed to support the management and resolution of variability in architectural and failure models specified in AADL/Error Annex languages with the support of Eclipse-based OSATE development environment (DELANGE and FEILER, 2014). The developed BVR adapters for AADL/Error Annex were tested to manage variation into an AADL/Error Annex version of the Hybrid Braking System product line architecture and failure models discussed throughout this thesis. Finally, peer review analysis of the variability management in automotive and aerospace case studies has shown that the proposed variability management approach can be ***systematically*** applied to other safety-critical product lines. The variability management and resolution phases and their respective activities were executed sequentially and produced equivalent deliverables in both automotive and aerospace case studies.

From the feedback provided by the analysis of the case studies and the developed tool support, the following remarks can be made:

- **Traceability of product line variation throughout system and safety assets** is achieved by adapting and integrating existing variant management techniques to compositional safety analysis techniques to enable support for handling variation in safety models. These adapters were used to support variant management into automotive and aerospace product line case studies, which are instantiations of the proposed approach. Therefore, it shows that variation in design and context identified in domain engineering is traced to their realization in both architectural and failure models. Thus, the proposed approach supports the automated integration between model-based development, compositional safety analysis, and variant management assets as prescribed by Habli's "*Product Line Safety Metamodel*" (HABLI 2009). The systematic reuse of architecture and safety assets and traceability of product requirements throughout these assets is achieved in application engineering. In application engineering, changes in product requirements are automatically propagated throughout architecture, failure models, and safety assessment and assurance cases, which are generated from the reused architecture and failure models;

- **The Safety-Critical Product Line Engineering (SC-PLE) approach is generic and systematic:** the term generic means that the approach presented in this thesis is domain and tool independent. Domain independence means that the approach does not address a specific domain, e.g., not specific to military, automotive, aerospace or nuclear domains. Tool independence means that the approach is not tied to specific model-based development, compositional safety analysis, or variant management tools. Systematic means that the variability management phases, activities and deliverables defined in the approach are repeatable, in which the execution of the approach in different case studies follow the same steps and produce equivalent deliverables. Automotive and aerospace case studies and the tooling support for managing variability into HiP-HOPS and AADL Error Annex safety models provide indications to appeal that the approach is generic and systematic.

The feasibility of the proposed approach is limited to the development of adapters for existing variant management tools to support variability management in safety analysis models. The complexity of adapting a variant management tool is dependent upon the characteristics of the tool and its available documentation. With regard to variability management, the precision in identifying the impact of context and design variation into architectural and safety analysis models in variability realization modeling is limited to product line engineers domain knowledge and experience. In addition, the conduction of automotive and aerospace variability management case studies has shown that the complexity of variability realization modeling may increase as the size of the product line scope increases. Therefore, variability realization modeling may become a time-consuming and error-prone task. Such complexity can be reduced with the adoption of graphical editors, as provided by BVR toolset, to support the variability realization modeling.

8.6.2 Product Line Compositional Safety Analysis and Design Optimization

Regarding product line compositional safety analysis and design optimization, it has been proposed an approach to integrate compositional safety analysis and design optimization into safety-critical software product line engineering processes. This approach supports the systematic reuse of product line safety analysis and automatic generation of safety assessment artefacts in application engineering. In domain engineering, the proposed approach has shown that performing compositional safety analysis aware of the impact of context and design variation supports the systematic reuse of functional hazard assessment and component failure analysis. In application engineering, the proposed approach has shown that integrating compositional safety analysis and design optimization techniques into product line safety processes allows the automatic synthesis of variant-specific fault trees, FMEA, SIL decomposition assessment artefacts from the reused safety analysis.

The product line design optimization approach presented in this thesis comprises the provision of methodological and tooling support for automatic analysis and decomposition of safety integrity requirements to product line components addressing process-based certification early on domain engineering (OLIVEIRA *et al.* 2015a). Product line component safety integrity requirements is useful to guide safety engineers in establishing safety objectives, development and safety assessment processes for product line components in order to achieve process-based certification. The allocation of safety integrity requirements to product line components can contribute to reduce the effort and costs of developing safety-critical product line components since "expensive" safety objectives, in terms of time and budget, are allocated only to highly critical components and not to all product line components. The research contributions in this area were mainly validated through academic peer review, the development of tool support for automated analysis and allocation of product line component safety integrity requirements, and case studies: Hybrid Braking System automotive case study, and the Tiriba Flight Control product line aerospace case study.

The product line compositional safety analysis and design optimization case studies conducted in automotive and aerospace domains have shown the automated and explicit ***traceability*** between variation in design and context, product line architecture and failure models, and variant-specific safety assessment assets. ***Feasibility*** has been indicated by instantiating the product line compositional safety analysis and design optimization approach, with the support of HiP-HOPS compositional safety analysis and design optimization toolset (PAPADOPOULOS *et al.* 2011), in automotive and aerospace case studies. ***Feasibility*** has also been shown by developing an extension to HiP-HOPS Tabu Search ASIL/DAL decomposition optimization tool to support the automated analysis and decomposition of safety integrity requirements to product line components (OLIVEIRA *et al.* 2015a) and its validation in automotive and aerospace case studies. Finally, peer review analysis performed by academic researchers and safety engineers domain experts from the industry (HiP-HOPS tool developers), and automotive and aerospace case studies have shown that the product line compositional safety analysis and design optimization approach can be ***systematically*** applied in different safety-critical domains, with the support of different model-based development, compositional safety analysis and design optimization techniques. From the analysis of case study results against the principles defined in Habli's *Product Line Safety Metamodel*, the following observations can be made:

- **Explicit focus on causality and product line variation** principle is addressed in the product line safety analysis phase, where the identified conditions leading the system to unsafe states, their causes and allocated safety requirements are explicitly traced to the usage scenarios considered to perform functional hazard assessment and component failure analysis. Therefore, the impact of context and design variation in the safety analysis is captured and traced, allowing the systematic reuse and generation of safety artefacts in application engineering;
- **Instantiation of the product line compositional safety analysis and design optimization approach and Safety Metamodels**: automotive and aerospace product line compositional safety analysis and design optimization case studies have shown the effectiveness of the integration of variant management and compositional safety analysis techniques in automating the traceability of context and design variation throughout product line safety analysis, and variant-specific fault trees and FMEA results. It means that traceability links between system and contextual assets, and safety assets defined into "*Functional Failure", "Architectural Failure",* and "*Component Failure"* metamodels can be automated with the support of the integration between variant management and compositional safety analysis techniques;
- **Integration of compositional safety analysis and design optimization within product line domain engineering and application engineering**: in domain engineering, product line safety analysis is performed aware of variation in design and context to support the systematic reuse of safety analysis data in application engineering. In application engineering, the reused variant-specific safety analysis data is the input for compositional safety analysis techniques, e.g., HiP-HOPS, OSATE AADL/Error Annex, generating fault trees, FMEA, and safety integrity requirements decomposition safety assets. Therefore, changes in the product requirements are directly propagated throughout architecture, safety analysis and safety assessment assets;
- **Consideration of Product Line Component Safety Integrity Requirements and Process factors:** in this thesis we have considered the impact of product line

component safety integrity requirements in the allocation of prescriptive safety objectives, development activities, and output artefacts required to achieve product line process-based certification. The proposed method and tool support for automatic analysis and allocation of safety integrity requirements to product line components (see Chapter 5) fills the gap in this area left by earlier research in product line safety assessment (HABLI, 2009). It is important to highlight that the proposed product line component safety requirements allocation tool addresses safety requirements in terms of Safety Integrity Levels.

8.6.3 Product Line Model-Based Assurance Cases

The product line model-based assurance cases approach has shown how to integrate model-based assurance cases into software product line engineering processes to support the automatic generation of assurance cases from variant-specific design and assessment models. Therefore, a model-based approach has been defined to integrate assurance case construction into product line domain engineering and application processes. The research contributions in this area were evaluated through automotive and aerospace case studies. Automated ***traceability*** of variation in product requirements throughout development, safety assessment, assurance case models is shown by carrying out automotive and aerospace assurance case construction case studies. The ***feasibility*** of the approach has also been shown by case studies where the Model-Based Assurance Cases (HAWKINS *et al.* 2015) tool was used to support the generation of assurance cases in product line application engineering. Academic peer review and the conduction of case studies in different domains have shown that the approach defines ***systematic*** guidance and deliverables applicable to other safety-critical domains.

Model-Based Assurance Cases in product line engineering supports the automated traceability of variation in development and assessment models throughout the assurance case. Therefore, changes in product requirements are automatically propagated throughout the assurance case. Model-based assurance cases technique provides the automated generation of the evidence required for certifying product variants by linking assurance case argument elements to system models (evidence).

8.6.4 Pattern Instantiation Constraints and Artefact Patterns in Model-Based Assurance Cases

The SACM-MBAC approach proposed in this thesis has shown that the OMG SACM 2.0 (OMG, 2015a) metamodel abstractions are effective in: specifying instantiation constraints in assurance case patterns, and artefact patterns and their links to assurance case patterns. The specification of instantiation constraints in assurance case patterns contributes to resolve data dependencies between pattern elements during the assurance case pattern instantiation process. The specification of artefact patterns supports tracing assurance claims to the provenance information of artefacts referenced in these claims, contributing to increase the confidence in the assurance case. This area of contribution was evaluated through academic peer review, tool support, and case study. The ***feasibility*** of research contribution in this area has shown by peer review analysis performed by academic members from the OMG SACM 2.0 committee, the development of SACM 2.0 model editor and GSN2SACM 2.0 model transformation tool, and by SACM 2.0 assurance case and artefact patterns for a hybrid braking system variant.

The analysis of the SACM 2.0 assurance case and artefact patterns has shown the effectiveness of SACM 2.0 in specifying instantiation constraints in assurance case patterns, and linking artefact patterns to assurance case patterns. These findings contributed to overcome the limitations related to the specification and weaving of assurance case pattern instantiation constraints from the current model-based assurance cases approach (HAWKINS *et al.* 2015). Although we have defined the structure of an SACM 2.0 Model-Based Assurance Cases approach, more effort is required to fully adapt the current model-based assurance cases tooling (HAWKINS *et al.* 2015) to support the SACM 2.0 metamodel (OMG, 2015a).

8.7 Summary

This chapter presented the evaluation of the four main thesis contributions. The evaluation was performed via peer review, development of tooling support, case studies, and one case study in aerospace domain, which generated the results that supported the thesis hypothesis. The following chapter presents the final conclusions of this thesis.

Chapter 9

CONCLUSIONS

This chapter presents a summary of the conclusions obtained from the research contributions of the thesis. Future research directions are also presented and discussed.

9.1 Thesis Contributions

A model-based approach to support the systematic reuse and generation of safety artefacts in safety-critical product line engineering has been defined in this thesis. The approach provides systematic and holistic guidance to integrate compositional safety analysis, variant management, and model-based development techniques to support safety-critical product line engineering processes. The research contributions in the context of the proposed approach provide answers for research questions defined in the introduction (Section 1.3). The contributions of this thesis are focused in the following areas:

- **Safety Aware Software Product Line Engineering Processes**: the definition of a systematic and holistic approach, which comprises a set of processes, activities, and tasks, to support the management of the impact of variation in design and context in safety models to enable the systematic reuse of these models, and generation of safety assurance artefacts in product line engineering for safety-critical systems (OLIVEIRA *et al.* 2016; OLIVEIRA *et al.* 2014) (***RQ1, RQ2***);
- **Product Line Safety Variability Management:** the provision of a systematic approach to manage the impact of variation in design and context in architecture and safety models, guidance to integrate variant management into compositional safety

analysis to support variability management in the safety analysis model, and the development of adapters for existing variant management tools, e.g., the BVR toolset (***RQ2, RQ3***);

- **Product Line Compositional Safety Analysis:** the definition of a systematic approach to support compositional safety analysis aware of variation in design and context in product line domain engineering. Such approach supports the systematic reuse of the safety model, and automatic generation of safety artefacts in product line application engineering (OLIVEIRA *et al.* 2016; OLIVEIRA *et al.* 2014) (***RQ2, RQ3***);
- **Product Line Design Optimization and Safety Certification:** the definition of a method and tool for automated analysis and allocation of safety integrity requirements to product line components to support cost-effective process-based certification (OLIVEIRA *et al.* 2015a) (***RQ5***); and
- **Product Line Model-Based Assurance Cases:** the definition of a systematic approach to integrate model-based assurance case techniques into product line engineering processes to support the automatic generation of variant-specific assurance cases required to achieve goal-based certification (OLIVEIRA *et al.* 2015) (***RQ4***).

The conclusions in each one of these contributions are presented in the following sections.

9.1.1 Safety Aware Software Product Line Engineering Processes

The research contribution in this area comprises the definition of a holistic and systematic approach to integrate compositional safety analysis, variant management, and model-based development into safety-critical software product line engineering processes. This thesis has shown that applying such approach provides the fully integration of variability management in safety analysis, safety assessment and assurance case construction activities, with the support of model-based techniques, into software product line engineering processes. By using compositional safety analysis, variant management, and model-based assurance case techniques, automated traceability of product line variation throughout hazard and risk analysis, component failure analysis, fault trees analysis, FMEA results and the assurance case defined through a new “Product Line Safety Metamodel” (HABLI, 2009) is achieved.

The proposed approach differs from existing work on product line safety assessment by establishing a clear distinction between reusable safety artefacts, where variability should be managed, and those that can be auto-generated with the support of compositional safety analysis techniques. In domain engineering, product line variation is linked to their realization in architecture and safety analysis models to be systematically reused in application engineering, where safety assessment and assurance case artefacts can be generated from the reused architecture and safety analysis. Thus, changes in a product variant are automatically propagated throughout the reference architecture, safety assessment assets and the assurance case. Whereas traditional software product line engineering processes (SEI, 2016; GOMAA, 2005; POHL *et al.* 2005) do not suite safety processes established in safety standards for the development of safety-critical systems, product line engineering processes can be adapted by following the guidance defined in the proposed approach. By taking a software product line engineering process, and adapting it to suite safety processes, by following the guidance defined in the proposed approach, it results in the thesis contributions detailed in the following sections.

The model-based approach presented in this thesis was built upon, and to comply with, safety assessment processes defined in normative safety standards, e.g., IEC 61508, ISO 26262, SAE ARP 4754A, and SAE ARP 4761. The results obtained from automotive and aerospace case studies and the development of extensions for variant management tools have shown that the approach is domain and tool independent. Domain independence means that the approach is applicable to other domains and safety standards, e.g., to support product line safety assurance in industrial and medical domains. Tool independence means that the approach can be supported by integrating compositional safety analysis, variant management, and model-based development tools other than the chosen in this thesis. The results from the cases studies have shown that the approach is *systematic* (repeatable), which the same activities were performed and the same output artefacts were generated in both case studies. The approach's *feasibility* and *traceability* have been shown through the seamless integration between variant management, compositional safety analysis and model-based development tools.

9.1.2 Product Line Safety Variability Management

The research contributions in this area are: a systematic approach to support the management of the impact of context and design variation in architecture and safety analysis models, and resolution of variability on these models; the provision of guidance to integrate variant management into compositional safety analysis to support variability management in safety models developed in the targeted compositional safety analysis tool; and the development of tooling support. This thesis has shown that the proposed variability management approach supports the automated traceability of product line variation throughout product line assets, and systematic reuse of product line architecture and safety models when variability is resolved for a particular product variant. Thus, changes in a given product variant are automatically propagated throughout variant-specific architecture and safety assets. This is achieved by the provision of seamless integration between variant management, compositional safety analysis, and model-based development tools. Such integration is provided by adapters for variant management tools developed in this thesis to support variability management in MATLAB/HiP-HOPS and OSATE AADL & Error Annex models, which are mature compositional safety analysis tools. The guidance presented in this thesis to adapt variant management tools to support safety models was applied to develop adapters for Hephaestus/Simulink and BVR variant management tools. Specifically, the BVR adapter for OSATE AADL & Error Annex provides seamless integration between OSATE AADL modeling environment and BVR toolset, allowing engineers performing system modeling, failure modeling, and variability modeling in a single platform. It contributes to reduce the effort and learning curve to apply the approach whereas separated training and dedicated support to handle different tools is not required. Finally, the complexity of adapting a particular variant management tool using the guidance proposed in this thesis is dependent upon the characteristics of the tool and its available documentation.

9.1.3 Product Line Compositional Safety Analysis

The research contribution in this area was the provision of a systematic approach to perform, with the support of compositional techniques, safety analysis that recognizes the impact of context and design variation in domain engineering. The approach supports the systematic reuse of architecture and safety models, and automatic generation of safety assessment assets from the reused safety model in the application engineering. This approach automates traceability links between product line variation and safety assessment artefacts defined in the "*Product Line Safety Metamodel*" (HABLI, 2009) with the support of compositional safety analysis and variant management tools. At functional level, context and design variation is linked to hazards and their causes, risk assessment, allocated safety requirements, and component failure logic as defined in a Functional Failure Model. At architectural and component levels, traceability links between product line variation and safety analysis information are automatically propagated throughout variant-specific fault trees and FMEA results, generated from the reused safety model as defined in Architecture and Component Failure Models.

In this thesis we have extended compositional safety analysis by considering the impact of context and design variation in the definition of safety properties as defined in previous research (HABLI, 2009; HABLI et al., 2009). This is important to support the systematic reuse of product line components in different scenarios. The proposed approach has shown that performing safety analysis aware of variation in design and context allows safety analysts to identify how different product variants, usage context, product line evolution and reuse scenarios impact in the definition of system hazards, their causes and associated risks, allocated safety requirements, and component failure logic (BAUMGART *et al.* 2014). Different product variants might have different architectures and components changing safety properties, e.g., redundant and non-redundant architectures might have different hazards, with different causes, allocated functional and integrity safety requirements. Variation in the usage scenarios within which product variants operate may also lead to different safety properties. Therefore, different contexts assumed for a given variant can result in different hazards and causes, allocated safety integrity requirements, and component failure logic. Considering context and design variation in safety analysis also makes it possible to identify the impact of the evolution of the product line architecture on the safety properties. For example, the addition, modification, or removal of a component might change hazards, safety requirements, and component failure logic. It also allows safety analysts to identify the impact of different reuse scenarios for product line features on their safety properties, supporting the reuse of these features across product lines. For example, different hazards and safety goals may arise when a given feature from an aircraft flight control product line is assumed to be reused in another product line.

The adoption of compositional safety analysis techniques to support the product line safety processes allows safety analysis to be performed integrated with the architectural design in a modular fashion. It allows the safety analysis to be performed in a manner that is aware of product line variation, and modular reuse of both components and their safety analysis information is achieved when variability is resolved with support of variant management tools. The resultant product line architecture and failure models are integrated in a single model. Compositional safety analysis techniques use fault modeling languages to integrate failure analysis with architecture modeling, e.g., HiP-HOPS failure modeling can be integrated with MATLAB/Simulink and SimulationX, and the AADL Error Annex can be integrated with AADL. This approach supports the traceability between architectural components and their failure data, allowing the systematic reuse of architectural components and their failure data. Therefore, variant management tools are further used to link design and context variation expressed in the feature model to their realization in the product line architecture and failure models. By performing safety analysis that is aware of context and design variation, it is possible to clearly identify the safety analysis information that can be safely reused when variability is resolved for a given functional variant and its context.

Product line compositional safety analysis approach increases the potential for the reuse of safety analysis artefacts in safety-critical software product line engineering, thereby reducing the effort and complexity of performing safety analysis for a specific variant. Therefore, if no variant-specific component is added to the architecture, safety assessment artefacts such as fault trees and FMEA results, can be auto-generated for a given variant from the reused architecture and failure models. On the other hand, if a new component is added to the variant architecture, variant-specific safety analysis should be performed by considering the impact of the new component on the safety of a given variant, and later, variant-specific fault trees and FMEA can be generated. The proposed approach to support compositional safety analysis in product line processes complies with normative requirements defined in traditional safety standards such as ISO 26262 and SAE ARP 4754A. For example, compositional safety analysis, fault tree analysis, and FMEA activities defined in the proposed approach address the *Functional Hazard Assessment* (FHA), *Preliminary System Safety Assessment* (PSSA), and *System Safety Assessment* (SSA) phases defined in SAE ARP 4754A safety assessment process (EUROCAE, 2010). Considering ISO 26262 (ISO, 2011), *Hazard and Risk Analysis*, *Derivation of Safety Goals from Hazard Analysis*, *System Safety Analysis* (FTA and FMEA), *Safety Assessment*, and *Assurance Case* construction activities defined in its safety lifecycle are addressed by the model-based approach proposed in this thesis.

9.1.4 Product Line Design Optimization and Safety Certification

A method and tool built upon the HiP-HOPS optimization algorithms for decomposition of ASILs and DALs was developed to support the automatic allocation of safety integrity requirements to product line components from the analysis of allocation results for multiple product line instances. Given that allocating stringent safety integrity requirements to all product line components would increase the development effort and costs in terms of verification, validation, and testing, the proposed method supports near-optimal allocation of safety integrity requirements to product line components. Therefore, cost-effective product line process-based certification is achieved by allocating stringent integrity requirements only to highly critical components of the product line architecture and less stringent integrity requirements to less critical components. Although existing design optimization techniques and tools for automatic decomposition of safety integrity requirements were not originally designed to address product lines, this thesis has shown how these techniques can be adapted to the product line design optimization approach defined in the proposed method (Chapter 5).

9.1.5 Product Line Model-Based Assurance Cases

A novel and systematic approach to integrate model-based assurance cases techniques, built upon the model-weaving concept, into product line processes is the main contribution in this area. This approach was instantiated with the support of MBAC tooling to support the automatic generation of variant-specific assurance case from a diverse set of system models (HAWKINS *et al.* 2015). Traceability links between safety assessment models and the assurance case defined in the "*Product Line Safety Metamodel*" are used. Thus, changes in a given product variant and its system models are automatically propagated throughout the assurance case. The contribution in this area also comprises the extension of MBAC to enable the specification of instantiation constraints in assurance case patterns, artefact patterns and their integration with assurance case patterns. Pattern instantiation constraints make it possible to define dependence relationships between assurance case pattern elements. An artefact pattern defines the relationship between the provenance information associated with an evidence item (a development artefact) referenced in the assurance case. Artefact pattern instances integrated with the assurance case contributes to increase the confidence on the evidence items referenced in the assurance case.

9.2 Limitations

This section highlights the limitations of the thesis contributions. Notice that specific limitations have already been discussed in the previous chapters. Here, we present the overall limitations of the thesis contributions. It is worth to highlight that we intend to deal with some of these issues in future work.

Evaluation of the integration of variant management into compositional safety analysis: the integration of variant management into compositional safety analysis presented in Chapter 4 is limited to medium sized case studies and HiP-HOPS (PAPADOPOULOS *et al.* 2011) compositional safety analysis tool. Case studies with different and more complex safety-critical product lines should be conducted to obtain more evidence regarding the scalability of the proposed solution for integrating variant management into compositional safety analysis. In addition, other case studies should be conducted to further evaluate the integration of variant management into OSATE AADL Error Annex compositional safety analysis tool (DELANGE and FEILER, 2014).

Limitations of variant management and product line compositional safety analysis: case studies presented throughout this thesis (Chapters 4, 5, and 8) are limited to managing variability in hazards, safety integrity requirements, and component failure analysis. Probabilistic properties of components and functional safety requirements may change from a product variant to another. Therefore, further work aims to investigate the use of the proposed integration between variant management and compositional safety analysis techniques to manage variability in probabilistic safety assessment and functional safety requirements.

The evaluation of the product line compositional safety analysis approach: presented in Chapters 5 and 8 is limited to case studies in automotive and aerospace domains, and HiP-HOPS compositional safety analysis technique. Further evaluation of the product line compositional safety analysis approach should be done to address product lines from different domains, e.g., medical and nuclear power plant systems, and other compositional safety analysis techniques such as OSATE AADL/Error Annex.

Limitations of product line design optimization approach: the proposed method and tool to support design optimization into product line engineering presented in Chapter 5 is limited to support the allocation and decomposition of safety integrity requirements to product line components. Further research in this area aims to investigate the potential of using design optimization techniques to support architectural decisions in product line development process.

Limitations of the product line MBAC approach: the current product line MBAC approach presented in Chapter 6 is limited to support the generation of product-based arguments, not covering process-based arguments (HABLI and KELLY, 2006). Process-based arguments provide backing support for the justification of the process in which safety evidence referenced in the assurance case have been developed. Therefore, further work aims to adapt the current MBAC approach to support the generation of process-based arguments from the information provided by design optimization (safety integrity requirements) and safety standards.

Limitations in the experimental study: the experiment conducted to evaluate the product line MBAC approach in generating assurance cases for product variants involved 24 participants (see Chapter 6). The obtained results provide evidence of the effectiveness and efficiency of the product line MBAC approach, but the scope is limited to post-graduate students in computer science. Further replications of this experiment involving professionals from the industry could be performed to obtain stronger evidence to generalize the results.

Limitations of the SACM-MBAC approach: the current support for the SACM-MBAC approach presented in Chapter 7 is limited to a prototype tool. Additional effort is required to evolve it towards a more stable version. With regard to evaluation, other case studies using different assurance case and artefact patterns, and safety-critical systems should be performed to obtain more evidence regarding the benefits provided by the SACM-MBAC approach.

The following section details the future research.

9.3 Future Research

The following areas have been identified as subject of future research:

- Probabilistic Safety Assessment and Management of Functional Safety Requirements;
- Product Line Design Optimization;

- Model-Based Assurance Cases and Process-Based Arguments;
- Assurance Case Pattern-Based Approach and Model-Driven Development;
- Variability Management and Compositional Safety Analysis in System of Systems Architectures;

Further research in each one of these areas is detailed in the following sections.

9.3.1 Probabilistic Safety Assessment and Management of Functional Safety Requirements

In this thesis, we have focused on using compositional safety analysis and variant management techniques to manage the impact of design and context variation on qualitative safety properties such as hazards, their causes, allocated safety integrity requirements, and component failure logic. However, product line variation also impacts in the probability of a failure in an architectural component. Therefore, a component failure condition might have the same causes in different variants and contexts, but the failure rate associated with that condition might vary from one variant/context to another. Since failure rates are dynamically calculated based on the instantiation of variation points in the product line architecture and failure model, context and design variation may lead to different failure rates for the same failure mode of a component. Further research in this area is intended to manage the impact of variation in design and context on the probabilistic attributes of components.

Functional safety requirements represent architectural decisions intended to mitigate the effects of failures in the overall safety of the system. Functional safety requirements are strongly connected to safety properties such as hazards and component failure modes. Variation in safety properties may require the allocation of different functional safety requirements. In a safety-critical product line, the addition of newer system functions and their associated failure behavior lead to new functional safety requirements. In addition, a functional safety requirement may be associated with one or more safety conditions, and the mitigation of a given safety condition might be associated with multiple functional safety requirements. Thus, the management of the impact of variation in design and context on the relationships between functional safety requirements and safety conditions provide supports for the systematic reuse of functional safety requirements and it ensures the safety of product variants. Further research in this area intends to explore the potential use of the integration between variant management, compositional safety analysis, and model-based development tools to support the management and systematic reuse of functional safety requirements in product line safety processes.

9.3.2 Product Line Design Optimization

This thesis has shown the benefits in extending design optimization techniques to support the identification of near-optimal solutions for allocation of safety integrity requirements to product line components to achieve cost-effective process-based certification. Further research in this area should investigate the potential of design optimization techniques to support engineers in taking decisions based on optimization objectives such as reliability, safety, and cost, in developing the product line architecture.

9.3.3 Model-Based Assurance Cases and Process-Based Arguments

In this thesis we have focused in supporting the automatic generation of product-based arguments to product line instances. Whereas process-based arguments provide backing support for justifying the confidence on the process under which the evidence referenced in the assurance case have been developed, it would also be useful to consider process-based arguments using an MBAC approach. Process-based arguments can be generated from the information provided by product line design optimization and prescriptive safety standards. Design optimization provides the safety integrity requirements associated with components and safety standards provide the normative requirements to be addressed according to the given integrity level. Further research in this area could be focused on integrating process-based arguments within the product line MBAC approach to support the automated generation of process-based arguments integrated with product-based arguments.

9.3.4 Assurance Case Pattern-Based Approach and Model-Driven Development

Identifying how different assurance case patterns can be generated from a diverse set of design, assessment, and processes models contributes to improve the understanding about assurance case patterns and their instantiation. In addition, reusing relationships between assurance case patterns and system models, usually stored into the weaving model in the MBAC approach, has the potential to reduce the complexity of the assurance case generation process. The definition of patterns that describe how assurance case patterns can be generated from system models in a model weaving operation, as established in the Structured Patterns Metamodel Standard (SPMS) (OMG, 2015b), has the potential to fully automate the specification of the weaving between assurance case pattern and system model elements. SPMS is a standard for the definition and description of patterns used in architecture, design, and implementation of software systems. SPMS provides metamodels to store pattern specifications, pattern instances and their relationships, suitable for use in tooling and pattern repositories (OMG, 2015b). On the other hand, the OMG SACM 2.0 metamodel (OMG, 2015a) defines abstractions that improve the representation of assurance cases patterns and their relationships with development artefacts (evidence items).

Further research in this area aims to extend the traditional assurance case pattern-based approach with the definition of weaving patterns that describe how existing assurance case patterns are instantiated from the information provided by a diverse set of system models. Weaving patterns to establish how assurance case patterns for COTS (YE, 2005), software systems (WEAVER, 2003), and process-based arguments (HABLI and KELLY, 2006) are linked to system models should be created. Further work in this area is focused on investigating how OMG SACM 2.0 (OMG, 2015a) and SPMS (OMG, 2015b) metamodels can be used to integrate weaving patterns and assurance case patterns to evolve the MBAC (HAWKINS *et al.* 2015) approach to support the weaving of assurance cases patterns from a weaving pattern catalogue.

9.3.5 Variability Management and Compositional Safety Analysis in System of Systems

Systems of Systems (SoS) are complex, large scale software systems which operationally and managerially independent systems cooperate to provide new, unique functionalities that cannot be provided by any constituent system separately (DAGLI and KILICAY-ERGIN, 2008). SoS architectures are highly configurable large scale and complex systems that emerge different behaviors in interacting with multiple and different contexts (CAMPBELL *et al.* 2005). In the same way as large-scale safety-critical product lines, Systems of Systems Engineering has to deal with variability and evolution, and software product line concepts can be useful. Bosch (2002) has defined a taxonomy for product line approaches in which "*Program of Product Lines*" goes into the direction of SoS, which product line engineering can be seen as a techniques to develop components in a SoS approach (BOTTERWECK, 2013).

In a SoS architecture, any given constituent software system may interact with multiple actors (systems) in multiple contexts (MENON and KELLY, 2010). Thus, behaviors, which are explicitly required by one actor within the SoS may contribute to hazardous behavior manifested by other actor. In addition, different contexts assumed for an actor may emerge different behaviors that can contribute to different hazardous behaviors manifested by other actors. Performing safety analysis in SoS architectures is complex and challenging. Whereas the SoS approach has similarities with the SPLE approach with regard to variability, further research in this area aims to investigate the potential of the seamless integration between compositional safety analysis, variant management and context aware safety analysis in supporting safety analysis of highly configurable and complex SoS architectures.

REFERENCES

ADELARD. Adelard Safety Case Editor. Available on-line: <www.adelard.co.uk>, Accessed in January, 11th, 2016.

ADLER, R., DOMIS, D., HOEFIG, K., KEMMANN, S., KUHN, T., SCHWINN, J., TRAPP, M. Integration of component fault trees into the UML. In: MODELS IN SOFTWARE ENGINEERING: WORKSHOPS AND SYMPOSIA AT MODELS, 2010, Oslo, Norway, **Proceedings**…, Springer-Heidelberg, p. 312-327.

ALLENBY, K., KELLY, T. Deriving safety requirements using scenarios. In: INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING, 5th, 2001, Toronto, Canada, **Proceedings**…, IEEE, p. 228-235.

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., ANGEL, S. **A Pattern Language: Towns, Buildings, Construction**, Oxford University Press, United Kingdom, 1978.

AMERICA, P., ROMMES, E., OBBINK, H. Multi-view variation modeling for scenario analysis. In: INTERNATIONAL WORKSHOP ON PRODUCT FAMILY ENGINEERING, 5th, Nov. 4-6, 2003, Siena, Italy, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2004, v. 3014, p. 44-65.

ANAC, Certification. Available on-line: <http://www2.anac.gov.br/certificacao/>. Accessed in: January, 16th, 2016.

AZEVEDO, L. S., PARKER, D., WALKER, M., PAPADOPOULOS, Y., ARAUJO, R. Assisted assignment of automotive safety requirements. **IEEE Software**, IEEE, v. 31, i. 1, p. 62-68, 2014.

AZEVEDO, L. S., PARKER, D., WALKER, M., PAPADOPOULOS, Y., ARAUJO, R. E. Automatic decomposition of safety integrity levels: optimization by tabu search. In: WORKSHOP ON CRITICAL AUTOMOTIVE APPLICATIONS: ROBUSTNESS & SAFETY (CARS), 2nd, 2013, Toulouse, France, **Proceedings**…, p. N/A.

BACHMANN, F., GOEDICKE, M., LEITE, J., NORD, R., POHL, K., RAMESH, B., VILBIG, A. A meta-model for representing variability in product family development. In: INTERNATIONAL WORKSHOP ON SOFTWARE PRODUCT FAMILY ENGINEERING, 5th, Nov. 4-6, 2003, Siena, Italy, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2004, v. 3014, p. 66-80.

BACHMANN, F., BASS, L. Managing variability in software architectures. In: 2001 SYMPOSIUM ON SOFTWARE REUSABILITY: PUTTING SOFTWARE REUSE IN CONTEXT (SSR'01), May 18-20, 2001, Ontario, Canada, **Proceedings**…, ACM, New York, USA, p.126-132.

BASIR, N., DENNEY, E. FISCHER, B. Constructing a safety case for automatically generated code from formal program verification information. In: INTERNATIONAL

CONFERENCE ON COMPUTER SAFETY, RELIABILITY, AND SECURITY (SAFECOMP '08), 27th, 2008, Newcastle, United Kingdom, **Proceedings**…, Springer-Verlag, Berlin, Heidelberg, p. 249-262.

BASS, L., BACHMANN, F., KLEIN, M. Making variability decisions during architecture design. In: INTERNATIONAL WORKSHOP ON PRODUCT FAMILY ENGINEERING, 5th, Nov. 4-6, 2003, Siena, Italy, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2004, v. 3014, p. 454-465.

BATE, I. J., KELLY, T. P. Architectural considerations in the certification of modular systems. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY RELIABILITY AND SECURITY, 21st, 2002, Catania, Italy, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2002, v. 2434, p. 321-333.

BATTEUX, M., PROSVIRNOVA, T., RAUZY, A., KLOUL, L. The AltaRica 3.0 project for model-based safety assessment. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL INFORMATICS (INDIN), 11th, July 29-31, 2013, **Proceedings**…, IEEE, p. 741-746.

BAUMGART, S., ZHANG, X., FRÖBERG, J., PUNNEKKAT, S. Variability management in product lines of safety critical embedded systems. In: International Conference on Embedded Systems (ICES), 2014, Coimbatore, **Proceedings**…, p. 98-103.

BECKER, M. Towards a general model of variability in product families. In: WORKSHOP ON SOFTWARE VARIABILITY MANAGEMENT, 1st, 2003, Groningen, Netherlands, **Proceedings**…, p. N/A.

BECKER, M., GEYER, L., GILBERT, A., BECKER, K. Comprehensive variability modelling to facilitate efficient variability treatment. In: INTERNATIONAL WORKSHOP ON PRODUCT FAMILY ENGINEERING, 4th, Oct. 3-5, 2001, Bilbao, Spain, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2002, v. 2290, p. 294-303.

BENAVIDES, D., SEGURA, S., RUIZ-CORTÉS, A. Automated analysis of feature models 20 years later: A literature review. **Information Systems**, v. 35 n. 6, p. 615-636, September, 2010.

BESSLING, S., AND HUHN, M. Formal safety analysis and verification in the model-driven development of a pacemaker product line. In: MODELLBASIERTE ENTWICKLUNG EINGEBETTETER SYSTEME (MBEES), 2012, Dagstuhl, Germany, **Proceedings**…, p. 133-144.

BEUCHE, D. Modeling and building software product lines with pure::variants. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 16th, 2012, Salvador, Brazil, **Proceedings**…, ACM, New York, USA, v. 2, p. 255-255.

BIEBER, P., DELMAS, R., SEGUIN, C. 2011. DALculus theory and tool for development assurance level allocation. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY, AND SECURITY, 30th, Naples, Italy, **Proceedings**…., Springer Berlin Heidelberg, Lecture Notes in Computer Science, p. 43-56.

BIG LEVER, Gears. Available on-line: <http://www.biglever.com/>, Accessed in: January, 18th, 2016.

BISHOP, P, G., BLOOMFIELD, R. E. A methodology for safety case development. In: SAFETY-CRITICAL SYSTEMS SYMPOSIUM, 6th, 1998, Birmingham, United Kingdom, **Proceedings**..., Springer-London, Industrial Perspective of Safety-critical Systems, p. 194-203.

BLOOMFIELD, R., BISHOP, P. 2010. Safety and assurance cases: past, present and possible future - an Adelard perspective. In: SAFETY-CRITICAL SYSTEMS SYMPOSIUM, 18th, 2010, Bristol, United Kingdom, **Proceedings**…, Springer-London, Making System Safer, p. 51-67.

BONIFÁCIO, R., TEIXEIRA, L., BORBA, P. Hephaestus: a tool for managing product line variabilities. In: BRAZILIAN SYMPOSIUM ON COMPONENTS, ARCHITECTURE, AND SOFTWARE REUSE, 3rd, 2009, Natal, Brazil, **Proceedings**…, p. 26-34.

BOSCH, J. Maturity and evolution in software product lines: approaches, artefacts, and organization. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 2nd, 2002, San Diego, CA, USA, **Proceedings**…, Springer-Heidelberg, Software Product Lines, Lecture Notes in Computer Science, v. 2379, p. 257-271.

BOSCH, J. **Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach**, Addison-Wesley, New York, USA, 2000.

BOZZANO M, VILLAFIORITA A. The FSAP/NuSMV-SA safety analysis platform. **International Journal on Software Tools for Technology Transfer**, v. 9, i. 1, p. 9-24, 2007.

BOTTERWECK, G. Variability and evolution in systems of systems. In: WORKSHOP ON ADVANCES IN SYSTEMS OF SYSTEMS (AiSoS), 1st, 2013, Rome, Italy, **Proceedings**…, EPTCS v. 133, p. 8-23.

BOTTERWECK, G., POLZER, A., KOWALEWSKI, S. Using higher-order transformations to derive variability mechanism for embedded systems, In: WORKSHOPS AND SYMPOSIA AT MODELS, 2009, Denver, USA, **Proceedings**…, Springer, Models in Software Engineering, 2010, v. 6002, p. 68-82.

BRAGA, R. T. V., TRINDADE JR, O., CASTELO BRANCO, K. R. L. J., LEE, J. Incorporating certification in feature modeling of na unmanned aerial vehicle product line. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 16th, 2012, Salvador, Brazil, **Proceedings**…, ACM, p. 249-258.

BRANCO, K., PELIZZONI, J., NERIS, L., TRINDADE, O., OSORIO, F., WOLF, D. Tiriba: a new approach of UAV based on model-driven development and multiprocessors. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2011, Shanghai, China, **Proceedings**…, IEEE, p. 1-4.

CAA. Civil Aviation Authority, Available on-line: <https://www.caa.co.uk/home/>, Accessed in: January, 16th, 2016.

CIVIL AVIATION AUTHORITY (CAA). CAP 670 SW 01: Acceptable Means of Compliance to CAP 670 SW 01: Guidance for Producing SW 01 Safety Arguments for COTS Equipment, CAA, issue 2, 2014.

CAMPBELL, J. E., LONGSINE, D. E., SHIRAH, D., ANDERSON, D. J. 2005. System of Systems Modeling and Analysis. Technical report, number: SAND2005-0020, Sandia National Laboratories, Albuquerque, USA.

CAPILLA, R., BOSCH, J., KANG, K. C. **Systems and Software Variability Management: Concepts, Tools and Experiences**, Springer Publishing Company, 2013.

CAVE, C. H. An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006. Technical report, The Stationary Office, 2006.

CERTWARE. Eclipse-based, open source tool for safety, assurance, or dependability cases. Available on-line: <http://nasa.github.io/CertWare/cae.html>, Accessed in: January, 11th, 2016.

CENELEC. European committee for electro-technical standardization: BS EN 50159: 2001 railway applications – communication, signaling and processing systems, CENELEC, 2001.

CLEMENTS, P. On the importance of product line scope. In: INTERNATIONAL WORKSHOP ON PRODUCT FAMILY ENGINEERING, 4th, Oct. 3-5, 2001, Bilbao, Spain, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2002, v. 2290, p. 70-78.

CLEMENTS, P., NORTHROP, L. **Software Product Lines: Practices and Patterns**, Addison-Wesley, 2001.

CARNEGIE MELLON UNIVERSITY (CMU). Software product lines overview. Software Engineering Institute, 2015.

CARNEGIE MELLON UNIVERSITY (CMU). Product Line Hall of Fame, Software Engineering Institute. 2013. Available on-line: <http://splc.net/fame.html>. Accessed in: December, 15th, 2015.

COIT, D. W., SMITH, A. E. Penalty guided genetic search for reliability design optimization. **Computer Industrial Engineering**, v. 30, i. 4, p. 895–904, 1996.

COPI, I. M., COHEN, C, McMAHON, K. **Introduction to Logic**, 14th ed., Pearson Education, 2010.

CUENOT, P., FREY, P., JOHANSSON, R., LÖNN, H., PAPADOPOULOS, Y., REISER, M. O., SANDBERG, A., SERVAT, D., KOLAGARI, R. T., TÖRNGREN, M., WEBER, M. The EAST-ADL Architecture description language for automotive embedded software. In: MODEL-BASED ENGINEERING OF EMBEDDED REAL-TIME SYSTEMS (MBEERTS), 2010, Dagstuhl Castle, Germany, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, v. 6100, p. 297-305.

CZARNECKI, K., HELSEN, S., EISENECKER, U. Staged configuration using feature models. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 3rd, 2004, Boston, MA, USA, **Proceedings**…, Springer-Verlag, Lecture Notes in Computer Science, 2004, v. 3154, p. 266-283.

DAGLI, C. H.; KILICAY-ERGIN, N. **System of systems architecting**, John Wiley & Sons, 2008.

DAVID, P., IDASIAK, V., KRATZ, F. Reliability study of complex physical systems using SysML. **Reliability Engineering & System Safety**, v. 95, i. 4, p. 431-450, 2010.

DEB, K., PRATAP, A., AGARWAL, S., MEYARIVAN, T. A fast and elitist multi-objective genetic algorithm: NSGA-II. **IEEE Transactions on Evolutionary Computing**, v. 6 i. 2, p. 182-197, 2002.

DE CASTRO, R., ARAÚJO, R. E., FREITAS, D. Hybrid ABS with electric motor and friction brakes. In: INTERNATIONAL SYMPOSIUM ON DYNAMICS OF VEHICLES ON ROADS AND TRACKS, 22nd, 2011, Manchester, United Kingdom, **Proceedings**…, p. 1-7.

DEELSTRA, S. SINNEMA, M., BOSCH, J. Product derivation in software product families: a case study. **Journal of Systems and Software**, v. 74, n. 2, 2005.

DEHLINGER, J., LUTZ, R. Evaluating the reusability of product-line software fault tree analysis assets for a safety-critical system. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE (ICSR), 11th, 2009, Falls Church, VA, USA, **Proceedings**…, Springer-Heidelberg, Formal Foundations of Reuse and Domain Engineering, Lecture Notes in Computer Science, 2009, v. 5791, p. 160-169.

DEHLINGER, J., HUMPHREY, M., SUVOROV, L., PADMANABAHN, P., LUTZ, R. Decimal and PLFaultCAT: from product-line requirements to product-line member software fault trees. In: RESEARCH DEMONSTRATION AT INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'07), 29th, 2007, Minneapolis, **Proceedings**…, p. 49-50.

DEHLINGER, J., LUTZ, R. "PLFaultCAT: A product-line software fault tree analysis tool. **Automated Software Engineering**, v. 13, n. 1, p. 169-193, 2006.

DEHLINGER, J., LUTZ, R. Software fault tree analysis for product lines. In: INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEMS ENGINEERING (HASE'04), 8th, 2004, Florida, USA, **Proceedings**…, IEEE, p. 12-21.

DINGDING, L., LUTZ, R. Fault contribution trees for product families. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 13th, 2002, Proceedings…, IEEE, p. 231-242.

DELANGE, J., FEILER, P. Architecture fault modeling with the AADL error-model annex. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS (SEAA), 40th, 2014, Verona, Italy, **Proceedings**…, IEEE, p. 361-368.

DENNEY, E., PAI, G., WHITESIDE, I. Formal foundations for hierarchical safety cases. In: INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEMS ENGINEERING (HASE), 16th, 2015, Daytona Beach, California, **Proceedings**… IEEE, p. 52-59.

DENNEY, E., HABLI, I., PAI, G. Dynamic safety cases for through-life safety assurance. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE 2015), NEW IDEAS AND EMERGING RESULTS (NIER), 37th, May, 2015, Florence, Italy, **Proceedings**…, IEEE, p. 587-590.

DENNEY, E., NAYLOR, D., PAI, G. Querying safety cases. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY (SAFECOMP '14), 33rd, Sep., 2014, Florence, Italy, **Proceedings**…, Springer-Heidelberg, p. 294-309.

DENNEY, E., HABLI, I., PAI, G. Automating the assembly of aviation safety cases. **IEEE Transactions on Reliability**, v. 63, i. 4, p. 830-849, July 2014.

DENNEY, E., PAI, G., WHITESIDE, I. Hierarchical safety cases. In: NASA FORMAL METHODS SYMPOSIUM, 5th, 2013, **Proceedings**…, Springer-Heidelberg, Lecture Notes on Computer Science v. 7871, p. 478-483.

DENNEY, E., PAI, G. A formal basis for safety case patterns. In: International Conference on Computer Safety, Reliability and Security, 32nd, 2013, Toulouse, France, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, p. 21-32.

DENNEY, E., PAI, G. Evidence arguments for using formal methods in software certification. In: INTERNATIONAL WORKSHOP ON SOFTWARE CERTIFICATION (WoSoCer), Nov., 2013, Pasadena, CA, **Proceedings**…, IEEE, p. 375-380.

DENNEY, E.; PAI, G.; HABLI, I. Perspectives on software safety case development for unmanned aircraft. In: ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 42nd, 2012. **Proceedings…** IEEE, 2012, p. 1-8.

DENNEY, E., PAI, G. A lightweight methodology for safety case assembly. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY, 31st, 2012, Magdeburg, Germany, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, v. 7612, p. 1-12.

DENNEY, E., PAI, G., POHL, J. Advocate: an assurance case automation toolset. In: WORKSHOP ON NEXT GENERATION OF SYSTEM ASSURANCE APPROACHES FOR SAFETY CRITICAL SYSTEMS (SASSUR), 2012, Magdeburg, Germany, **Proceedings**…, Springer-Heidelberg, p. 8-21.

DHOUIBI, M. S., PERQUIS, J. M., SAINTIS, L. BARREAU, M. Automatic decomposition and allocation of safety integrity levels using system of linear equations. In: INTERNATIONAL CONFERENCE ON PERFORMANCE, SAFETY AND ROBUSTNESS IN COMPLEX SYSTEMS AND APPLICATIONS, 4th, 2014, Nice, France, **Proceedings**…, p. 1-4.

DEL FABRO, M. D., BÉZIVIN, J., JOUAULT, F., VALDURIEZ, P. Applying generic model management to data mapping. In: BASES DE DONNÉES AVANCÉES (BDA05), 2005, **Proceedings**…, p. N/A.

DEL FABRO, M. D., BÉZIVIN, J., JOUAULT, F., ERWAN, B., GUELTAS, G. AMW: a generic model weaver. In: ÈRES JOURNÉES SUR L'INGÉNIERIE DIRIGÉE PAR LES MODÈLES, 1st, 2005, Paris, **Proceedings**…, p. N/A.

DOMIS, D., ADLER, R. BECKER, M. Integrating variability and safety analysis models using commercial UML-based tools. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINES, 19th, 2015, **Proceedings**…, ACM, New York, NY, USA, p. 225-234.

DOMIS, D., TRAPP, M. Integrating safety analyses and component-based design. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY, AND SECURITY, 27th, 2008, Newcastle, United Kingdom, **Proceedings**…, Lecture Notes on Computer Science, Springer-Verlag, v. 5219, p. 58-71.

DORDOWSKY, F., BRIDGES, R., TSCHOPE, H. Implementing a software product line for a complex avionics system. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 15th, 2011, **Proceedings**…, IEEE, p. 241-250.

DOUGLASS, B. P. **Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems**, 1st ed., Addison-Wesley Professional, 2002.

ECLIPSE, Eclipse Modeling Framework (EMF) Project, 2016. Available on-line: <http://www.eclipse.org/modeling/emf/>. Accessed in: January, 16th, 2016.

EUROCAE. ARP4754A - Guidelines for Development of Civil Aircraft and Systems, EUROCAE, 2010.

EUROCONTROL, Safety Case Development Manual. 2006. Available on-line: <http://www. eurocontrol.int/sites/default/files/article/content/documents/nm/link2000/safety-case-developement-manual-v2.2-ri-13nov06.pdf>, Accessed in: January, 15th, 2016.

FAA, Federal Aviation Administration, Available on-line: <http://www.faa.gov/>. Accessed in: January, 16th, 2016.

FAA, AC 20-148: Reusable Software Components, Federal Aviation Administration (FAA), December, 2004.

FOOD AND DRUG ADMINISTRATION (FDA). Infusion Pumps Total Product Life Cycle: Guidance for Industry and FDA Staff. Technical report, FDA, 2014.

FEILER, P., NIZ, D., ASSIP study of real-time safety-critical embedded software-intensive system engineering practices. Technical report, number: CMU/SEI-2008-SR-001, 2008.

FENELON, P., MCDERMID, J. A. An integrated toolset for software safety analysis. **Journal of Systems and Software**, v. 21, i. 3, p. 279–290, 1993.

FENG, Q., LUTZ, R. Bi-Directional Safety Analysis of Product Lines. **Journal of Systems and Software**, v. 78, n. 2, p. 111-127, 2005.

FENN, J., HAWKINS, R., WILLIAMS, P., KELLY, T., BANNER M. G., OAKSHOTT, Y. The who, where, how, why and when of modular and incremental certification. In: IET INTERNATIONAL CONFERENCE ON SYSTEM SAFETY CONFERENCE, 2nd, 2007, **Proceedings**…, p. 135-140.

FENN, J. HAWKINS, R., KELLY, T. P. WILLIAMS, P. Safety case composition using contracts – refinements based on feedback from an industrial case study. In: SAFETY CRITICAL SYSTEMS SYMPOSIUM (SSS'07), 15th, Feb., 2007, Bristol, United Kingdom, **Proceedings**…, Springer-London, p. 133-146.

FREE SOFTWARE FOUNDATION. R-Software. Available on-line: <http://www.r-project.org/>. Accessed in: February, 1st, 2016.

GAMMA, E., HELM, R., JOHNSON, R., and VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison Wesley, 1995.

GE X., PAIGE R. F., McDERMID J. A. Probabilistic failure propagation and transformation analysis. In INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY, SECURITY, 28th, 2009, Hamburg, Germany, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, v. 5775, p. 5-28.

GEYER, L., BECKER, M. On the influence of variabilities on the application-engineering process of a product family. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINES, 2nd, 2002, San Diego, CA, USA, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, v. 2379, p. 1-14.

GOMAA, H. **Designing Software Product Lines with UML**, Addison-Wesley, 2005.

GÓMEZ, C., PETER, L., SUTOR, A. Variability management of safety and reliability models: an intermediate model towards systematic reuse of component fault trees. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY, AND SECURITY, 29th, 2010, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2010, v. 6351, p. 28-40.

GRUNSKE, L, KAISER, B. An automated dependability analysis method for COTS-based systems. In: INTERNATIONAL CONFERENCE ON COTS-BASED SOFTWARE SYSTEMS, 4th, 2005, Bilbao, Spain, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2005, v. 3412, p. 178–190.

GSN STANDARD. GSN community standard version 1.0. 2011. Available on-line: <http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf>. Accessed in: February, 24th, 2016.

GÜDEMANN, M. ORTMEIER, F. A framework for qualitative and quantitative model-based safety analysis. In: INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEM ENGINEERING (HASE), 12th, 2010, San Jose, CA, USA, **Proceedings**…, IEEE, 2010, p. 132-141.

GULA, R. J. **Nonsense: A Handbook of Logical Fallacies**, Axios Press, 2002.

HABLI, I, KELLY, T. A safety case approach to assuring configurable architectures of safety-critical product lines. In: INTERNATIONAL CONFERENCE ON ARCHITECTING CRITICAL SYSTEMS (ISARCS), 1st, 2010, Prague, Czech Republic, **Proceedings**…, Springer-Verlag, Lecture Notes in Computer Science, 2010, v. 6150, p. 142-160.

HABLI, I. Model-based assurance of safety-critical product lines. PhD Thesis, Department of Computer Science, The University of York, York, United Kingdom, 2009.

HABLI, I., KELLY, T., PAIGE, R. Functional hazard assessment in product lines: a model-based approach. In: MODEL-DRIVEN PRODUCT-LINE ENGINEERING, 1st, 2009, Enschede, Netherlands, **Proceedings**…, 2009, p. N/A.

HABLI, I., KELLY, T., HOPKINS, I. Challenges of establishing a software product line for an aerospace engine monitoring system. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 11th, 2007, Japan, **Proceedings**…, IEEE, 2007, p. 193-202.

HABLI, I., KELLY, T. Process and product certification arguments - getting the balance right. In: ACM SIGBED REVIEW – SPECIAL ISSUE ON WORKSHOP ON INNOVATIVE TECHNIQUES FOR CERTIFICATION OF EMBEDDED SYSTEMS, 12th, 2006, San Jose, CA, USA, **Proceedings**…, ACM, v. 3, i. 4, 2006, p. 1-8.

HALMANS, G. POHL, K., SIKORA, E. Documenting application-specific adaptations in software product line engineering. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 20th, June 16-20, 2008, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2008, v. 5074, p. 109-123.

HANSEN, P., LIH, K. W. Heuristic reliability optimization by tabu search. **Annals of Operations Research**, v. 63, i. 2, p. 321-336, 1996.

HAUGEN, O., OGARD, O. BVR - better variability results. In: INTERNATIONAL CONFERNCE ON SYSTEM ANALYSIS AND MODELING: MODELS AND REUSABILITY, 8th, 2014, Valencia, Spain, **Proceedings**…, Springer International Publishing, Lecture Notes in Computer Science, 2014, v. 8769, p. 1-15.

HAUGEN, O., MOLLER-PEDERSEN, B., OLDEVIK, J., OLSEN, G. K., SVENDSEN, A. Adding standardized variability to domain specific languages. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 12th, 2008, Limerick, Ireland, **Proceedings**…, IEEE, 2008, p. 139-148.

HAWKINS, R., HABLI, I., KOLOVOS, D., PAIGE, R., KELLY, T. Weaving an assurance case from design: a model-based approach. In: INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEMS ENGINEERING (HASE), 16th, 2015, Daytona Beach, USA, **Proceedings**…, IEEE, 2015, p.110-117.

HAWKINS, R., HABLI, I., KELLY, T. The need for a weaving model in assurance case automation. In: Architecture Centric Virtual Integration Workshop, 2015, ADA-Europe, Madrid, Spain, **Proceedings**…, 2015, p. N/A.

HAWKINS, R., HABLI, I. KELLY, T., McDERMID, J. Assurance cases and prescriptive software safety certification: a comparative study. **Safety Science**, v. 59, p. 55-71, 2013.

HEALTH AND SAFETY EXECUTIVE (HSE). ALARP at a glance, Available on-line: <http://www.hse.gov.uk/risk/theory/alarpglance.htm>. Accessed in: January, 20th 2016.

HEUER, A., STRICKER, V., BUDNIK, C. J., KONRAD, S., LAUENROTH, K., POHL, K. Defining variability in activity diagrams and petri nets. **Science of Computer Programming**, v. 78, n. 12, p. 2414-2432, December, 2013.

HiP-HOPS TEAM. HiP-HOPS automated fault tree, fmea and optimization tool user manual, 2013. Available on-line: <http://hip-hops.eu/index.php/the-manual>. Accessed in: February, 26th, 2016.

HOUDEK, H., LOWEN, U. **Advanced Model-based Engineering of Embedded Systems - The SPES XT Modeling Framework and its Applications**, Springer, 2015.

INTERNATIONAL ELECTRO-TECHNICAL COMMISSION (IEC), BS IEC 61508 – Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related System, British Standards Institute/IEC, 2010.

INTERNATIONAL ELECTRO-TECHNICAL COMMISSION (IEC), IEC 61513: 2001 Nuclear power plants – Instrumentation and control for systems important to safety General requirements for systems, IEC, 2001.

INCT-SEC. Instituto Nacional de Ciência e Tecnologia – Sistemas Embarcados Críticos. 2014. Available on-line: < http://www.inct-sec.icmc.usp.br/en/>. Accessed in: February, 19th, 2016.

ISO. ISO 26262: Road Vehicles Functional Safety, ISO, 2011.

ISOGRAPH SOFTWARE. Fault tree + v11 software tool. 2016. Available on-line: <http://www.isograph-software.com/index.htm>. Accessed in: February, 24th, 2016.

ITI GMBH. SimulationX. Available on-line: <http://simulationx.com>. Accessed in: February, 26th, 2016.

JACOBSON, I., GRISS, M., PATRIK, J. **Software Reuse: Architecture, Process, and Organization for Business Success**, Addison-Wesley-Longman, 1997.

JARING, M., BOSCH, J. Representing variability in software product lines: a case study. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINES, 2nd, 2002, San Diego, CA, USA, **Proceedings**…, Springer-Heidelberg, Lecture Notes on Computer Science, 2002, v. 2379, p. 15-36.

JOHANSEN, M. F., HAUGEN, O., FLEUREY, F. An algorithm for generating t-wise covering arrays from large feature models. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 16th, 2012, Salvador, Brazil, **Proceedings**…, ACM, v. 1, p. 46-55.

JOSHI, A., MILLER, S. P., WHALEN, M., HEIMDAL, M. P. E. A proposal for model-based safety analysis. In: DIGITAL AVINICS SYSTEMS CONFERENCE, 24th, 2005, Washingtion, D.C., **Proceedings**…, IEEE, 2005, p. N/A.

KÄßMEYER, M., SCHULZE, M., SCHURIUS, M. A process to support a systematic change impact analysis of variability and safety in automotive functions. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINES (SPLC '15), 19th, 2015, **Proceedings**…, ACM, New York, NY, USA, 2015, p. 235-244.

KÄßMEYER, M., MONCADA, D. S. V., SCHURIUS, M., Evaluation of a systematic approach in variant management for safety-critical systems development. In: INTERNATIONAL CONFERENCE OF EMBEDDED AND UBIQUITOUS COMPUTING (EUC), 13th, 2015, **Proceedings**…, IEEE, 2015, p. 35-43.

KAISER, B, GRAMLICH, C, FORSTER, M. State/event fault trees – a safety analysis model for software-controlled systems. **Reliability Engineering and System Safety**, v. 92, i. 11, p. 1521–1537, 2007.

KANG, K., COHEN, S. HESS J., NOVAK, W., PETERSON, A. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, number: CMU/SEI-90-TR-021, Pittsburgh, Software Engineering Institute, Carnegie Mellon University, 1990.

KASTNER, C., TRUJILLO, S., APEL, S. Visualizing software product line variabilities in source code. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, WORKSHOP ON VISUALIZATION IN SOFTWARE PRODUCT LINE ENGINEERING, 2nd, 2008, Limerick, Ireland, **Proceedings**…, IEEE, 2008, p. N/A .

KELLY, T. P. Can process and product-based approaches to software safety certification be reconciled? In: SAFETY CRITICAL SYSTEMS SYMPOSIUM, 16th, 2008, **Proceedings**…, Springer-London, Improvements in System Safety, 2008, p. 3-12.

KELLY, T. P. Arguing safety – a systematic approach to safety case management. PhD Thesis, Department of Computer Science, University of York, York, United Kingdom, 1998.

KELLY, T., MCDERMID, J. Safety case construction and reuse using patterns. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY, 16th, 1997, **Proceedings**…, Springer-London, LNCS, 1997, p. 55–69.

KLETZ, T., **Hazop and Hazan: Identifying and Assessing Process Industry Hazards**, 3rd ed., Institution of Chemical Engineers, 1992.

KOLOVOS, D., ROSE, L., GARCÍA-DOMÍNGEZ, A., PAIGE, R. **The Epsilon Book**, Eclipse, 2013. Available on-line: <http://www.eclipse.org/epsilon /doc/book/>. Accessed in: November, 14th, 2015.

KRUEGER, C., Variation management for software production lines. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 2nd, 2002, San Diego, CA, USA, **Proceedings**…, Springer-London, Lecture Notes in Computer Science, v. 2379, p. 37–48.

KULTUREL, K. S., SMITH, A. E., COIT, D. W. Efficiently solving the redundancy allocation problem using tabu search. **IIE Transactions**, v. 35, i. 6, p. 515–26, 2003.

LANDUYT, D. V. OP DE BEECK, S., HOVSEPYAN, A., MICHIELS, S. JOOSEN, W., MEYNCKENS, S., DE JONG, G., BARAIS, O., AND ACHER, M. Towards managing variability in the safety design of an automotive hall effect sensor. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 18th, 2014, **Proceedings**…, ACM, p. 304-309.

LEE, J. S., KATTA, V., JEE, E. K., RASPOTNIG, C. Means-ends and whole-part traceability analysis of safety requirements. **Journal of Systems and Software**, v. 83, p. 1612-1621, 2010.

LEE, K., KANG, K., LEE, J. Concepts and guidelines of feature modeling for product line software engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE: METHODS, TECHNIQUES, AND TOOLS, 7th, 2002, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, v. 2319, p. 62-77.

LEVESON, N. G. **Safeware System Safety and Computers**, Addison-Wesley, Reading, MA, 1995.

LEVESON, N. G. Software safety: why, what, and how. **ACM Surveys**, v. 18, n. 2, June, 1986.

LINDEN, F. J., SCHMID, K., AND ROMMES, E., **Software product lines in action: the best industrial practice in product line engineering**. Springer-Verlag New York, Inc., Secaucus, NJ, 2007.

LISAGOR O., MCDERMID J. A., PUMFREY D. J. Towards a practicable process for automated safety analysis. In: INTERNATIONAL SYSTEM SAFETY CONFERENCE (ISSC'06), 24th, 2006, **Proceedings**…, p. 596-607.

LITTLEWOOD, B., STRIGINI, L. Validation of ultrahigh dependability for software-based systems. **Communications of the ACM**, v. 36, n. 11, p. 69-80, 1993.

LIU, J., DEHLINGER, J., LUTZ, R. 2007. Safety analysis of software product lines using stated modeling. **Journal of Systems and Software**, v. 80, n. 11, p. 1879-1892.

MADER, R., ARMENGAUD, E., LEITNER, A., STEGER, C. Automatic and optimal allocation of safety integrity levels. In: ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM (RAMS), 2012. **Proceedings**…, IEEE, p. 1-6.

MARSH, W. SafEty and Risk Evaluation using bayesian NEts: SERENE,The SERENE Method Manual, version 1.0. Technical report, number: SERENE/5.3/CSR/3053/R/1, ERA Technology Ltd, May, 1999.

MARTINS, L. E. G., OLIVEIRA, T., A case study using a protocol to derive safety functional requirements from fault tree analysis, In: INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 22nd, 2014, Karlskrona, Sweden, **Proceedings**…, IEEE, p. 412-419.

MCDERMID, J. A., and PUMFREY, D. J. Software safety: why is there no consensus? In: INTERNATIONAL SYSTEM SAFETY CONFERENCE, 19th, 2001, Huntsville, USA, **Proceedings**…, System Safety Society, p. N/A.

MASON, P. A. J., SAEED, A., RIDDLE, S. On the role of traceability for standards compliance: Tracking requirements to code. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY, 22nd, 2003, Edinburgh, Scotland, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2003, v. 2788, p. 303–316.

MATHWORKS, MATLAB/Simulink. Available on-line: <http://www.mathworks.com /products/simulink/>. Accessed in: February, 26th, 2016.

MENON, C., KELLY, T. Eliciting software safety requirements in complex systems. In: Annual IEEE Systems Conference, 4th, San Diego, CA, **Proceedings**…, IEEE, p. 616-621.

METZGER, A. POHL, K. Software product line engineering and variability management: achievements and challenges. In: FUTURE OF SOFTWARE ENGINEERING (FOSE 2014), 2014, **Proceedings**…, ACM, New York, NY, USA, 2014, p. 70-84.

MoD. DEF-STAN 00-56 Issue 4 Part 1: Safety management requirements for defense systems. Technical report, UK Ministry of Defense, 2007.

MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION (MISRA), Report 6: Verification and Validation, MISRA, February 1995.

MUTHIG, D., ATKINSON, C. Model-driven product line architectures. In: INTERNATIONAL CONFERENCE, 2nd, 2002, San Diego, CA, USA, **Proceedings**…, Springer-Heidelberg, Lecture Notes in Computer Science, 2002, v. 2379, p. 110-129.

NAIR, S., DE LA VARA, J. S., MELZI, A., TAGLIAFERRI, G., DE LA BEAUJARDIERE, L., BELMONTE, F. Safety evidence traceability: problem analysis and model. In: INTERNATIONAL WORKING CONFERENCE ON REQUIREMENTS ENGINEERING: FOUNDATIONS FOR SOFTWARE QUALITY, 20th, Essen, Germany, **Proceedings**…, Springer Int. Publishing, Lecture Notes in Computer Science, v. 8396, pp. 309-324, 2014.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). Fault Tree Analysis with Aerospace Applications. Technical report, an update to NUREG-0492, NASA Office of Safety and Mission Assurance, Washington, DC, USA, August, 2002.

NEJATI, S., SABETZADEH, M., FALESSI, D., BRIAND, L., COQ, T. A SysML-based approach to traceability management and design slicing in support of safety certification: framework, tool support, and case studies. **Information and Software Technology** v. 54, p. 569–590, 2012.

OLIVEIRA, A. L., BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., HABLI, I., KELLY, T. Model-based safety analysis of software product lines. **International Journal of Embedded Systems**, Inderscience Publishers, 2016. *[to be published].*

OLIVEIRA, A. L., BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., HABLI, I., KELLY, T. Supporting the automated generation of modular product line safety cases. In: INTERNATIONAL CONFERENCE ON DEPENDABILITY AND COMPLEX SYSTEMS (DepCoS-RELCOMEX), 10th, June, 2015, Brunów, Poland, **Proceedings**…, Springer Int. Publishing, Advances in Intelligent Systems and Computing, Theory and Engineering of Complex Systems and Dependability, 2015, v. 365, p. 319-330.

OLIVEIRA, A. L., BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., AZEVEDO, L., PARKER, D., HABLI, I., KELLY, T. Automatic allocation of safety requirements to components of a software product line. In: IFAC SYMPOSIUM ON FAULT DETECTION, SUPERVISION AND SAFETY FOR TECHNICAL PROCESSES (SAFEPROCESS'15), 9th, 2015, Paris, France, **Proceedings**…, Elsevier, 2015, v. 48, i. 41, p. 1309-1314.

OLIVEIRA, A. L., BRAGA, R. T. B., MASIERO, P. C., PAPADOPOULOS, Y., HABLI, I., KELLY, T. A model-based approach to support the automatic safety analysis of multiple product line products. In: BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEMS ENGINEERING, 4th, 2014, Manaus, Brazil, **Proceedings**…, IEEE, 2014, p. 7-12.

OLIVEIRA, A. L., BRAGA, R. T. V., MASIERO, P. C., HABLI, I., KELLY, T. A pattern to argue the compliance of the system safety requirements decomposition. In: LATIN AMERICAN CONFERENCE ON PATTERN LANGUAGE OF PROGRAMS (SugarLoaf PLoP), 10th, Nov., 2014, Ilhabela, Brazil, **Proceedings**…, Hilside Group, 2014, p. N/A.

OLIVEIRA, A. L., BRAGA, R. T. V., MASIERO, P. C., HABLI, I. KELLY, T. Impact of feature interaction on the safety analysis for unmanned avionics product lines. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY (SAFECOMP) – FAST ABSTRACTS, 32nd, 2013, Toulouse, France, **Proceedings**…., Available on-line: <https://hal.archives-ouvertes.fr/hal-00926563/document>. Accessed in: December, 15th, 2015.

OBJECT MANAGEMENT GROUP (OMG). Systems modelling language (SysML), version 1.4. Object Management Group, September, 2015.

OBJECT MANAGEMENT GROUP (OMG). Structured assurance case metamodel (SACM), version 2.0. Object Management Group, June, 2015. ***[under revision]***

OBJECT MANAGEMENT GROUP (OMG). Structured patterns metamodel standard (SPMS), version 1.0. Object Management Group, October, 2015. Available on-line: <http://www.omg.org/spec/SPMS/>. Accessed in: February, 26th, 2015.

OBJECT MANAGEMENT GROUP (OMG). Structured assurance case metamodel (SACM), version 1.0. Object Management Group, 2013.

OBJECT MANAGEMENT GROUP (OMG). Software Process Engineering Metamodel (SPEM), version 2.0. Object Management Group, April 2008.

OBECT MANAGEMENT GROUP (OMG). Modeling and analysis of real-time and embedded systems, version beta 2. Object Management Group, June, 2008.

OBJECT MANAGEMENT GROUP (OMG). The meta-object facility (MOF), version 2.0. Object Management Group, 2004. Available on-line: <http://www.uml.org/>. Accessed in January, 19th, 2016.

ORIGIN CONSULTING YORK. GSN community standard version 1. GSN Standard, 2011, Available on-line: <http://www.goalstructuring notation.info/documents/GSN_Standard.pdf>. Accessed in: February, 27th, 2016.

PAPADOPOULOS, Y., WALKER, M., PARKER, D., RÜDE, E., HAMANN. Engineering failure analysis and design optimization with HIP-HOPS. **Journal of Engineering Failure Analysis**, v. 18, i. 2, p. 590-608, 2011.

PAPADOPOULOS, Y., WALKER, M., REISER, M. O., WEBER, M., CHEN, D., TORNGREN, M., SERVAT, D., ABELE, A., STAPPERT, F., LONN, H., BERNTSSON, L., JOHANSSON, R., TAGLIABO, F., TORCHIATO, S., SANDBERG, A. Automatic allocation of safety integrity levels. In: WORKSHOP ON CRITICAL AUTOMOTIVE APPLICATIONS: ROBUSTNESS AND SAFETY (CARS), 1st, 2010, **Proceedings**…, ACM, New York, USA, 2010, p. 7-10.

PAPADOPOULOS Y., MCDERMID, J. A. The potential for a generic approach to certification of safety critical systems in the transportation sector. **Journal of Reliability Engineering and System Safety**, v. 63, p. 47-66, 1999.

PARKER, D., WALKER, M., AZEVEDO, L., PAPADOPOULOS, Y., ARAUJO, R. Automatic decomposition and allocation of safety integrity levels using a penalty-based genetic algorithm. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL, ENGINEERING AND OTHER APPLICATIONS OF APPLIED INTELLIGENT SYSTEMS, 26th, 2013, Amsterdam, Netherlands, **Proceedings**…, Springer-Berlin, Recent Trends in Applied Artificial Intelligence, 2013, v. 7906, p. 449-459.

PARKER D., WALKER M., PAPADOPOULOS Y. Model-based functional safety analysis and architecture optimization. **Embedded Computing Systems: Applications, Optimization, and Advanced Design**, Information Science Reference (IGI Global), USA, p. 79-92, 2013.

POHL, K., BÖCKLE, G., AND VAN DER LINDEN F. J. **Software Product Line Engineering: Foundations, Principles and Techniques**. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

PUMFREY, D. J. The principled design of computer system safety analyses. PhD Thesis, Department of Computer Science, The University of York, York, United Kingdom, 1999.

PURE::SYSTEMS, pure::variants. Available on-line: <http://www.pure-systems.com/>. Accessed in: January, 23th, 2016.

RAATIKAINEN, M., SOININEN, T., MÄNNISTÖ, T., MATTILA, A. Characterizing configurable software product families and their derivation. **Software Process: Improvement and Practice, Special Issue on Software Variability and Process Management, John Wiley & Sons**, v. 10, i. 1, p. 41-60, 2005.

REUYS, A., REIS, S., KAMSTIES, E., POHL, K. The ScenTED method for testing software product lines. **Software Product Lines: Research Issues in Engineering and Management**, Springer-Heidelberg, p. 479- 520, 2006.

RIDDERHOF, W., GROSS, H.-G., DOERR, H. Establishing evidence for safety cases in automotive systems–a case study. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY, 26th, 2007, Nuremberg, Germany, **Proceedings**…, LNCS, Springer-Heidelberg, 2007, v. 4680, p. 1-13.

ROSS, D. T. Structured Analysis (SA): A language for communicating ideas. **IEEE Transactions on Software Engineering**. Piscataway, New Jersey, USA, v.3, n.1, p. 16-34, 1977.

RTCA. DO-178C Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, 2012.

RTCA. DO-331 Model-Based Development and Verification Supplement to DO-178C and DO-278A. Radio Technical Commission for Aeronautics, 2011

SALICKI, S., FARCET, N. Expression and usage of the variability in the software product lines. In: INTERNATIONAL WORKSHOP ON SOFTWARE PRODUCT-FAMILY ENGINEERING, 4th, 2001, Bilbao, Spain, **Proceedings**…, Springer-Heidelberg, LNCS, v. 2290, p. 304-318.

SCHULZE, M., MAUERSBERGER, J., BEUCHE, D. Functional safety and variability: can it be brought together? In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 17th, 2013, **Proceedings**…, ACM, New York, USA, 2013, p. 236-243.

SEI, Framework for Software Product Line Practice. Software Engineering Institute, Available on-line: < http://www.sei.cmu.edu/productlines/tools/framework/index.cfm>. Accessed in: February, 27th, 2016.

___, Assurance cases. Software Engineering Institute, Available on-line: <https://www.sei. cmu.edu/dependability/tools/assurancecase/>. Accessed in: February, 24th, 2016.

SIERLA, S., O'HALLORAN, B. M., NIKULA, H., NIKOLAOS PAPAKONSTANTINOU, N., TUMER, I. Y., Safety analysis of mechatronic product lines, **Mechatronics**, Elsevier, v. 24, i. 3, p. 231-240, 2014.

SHAKER, P., ATLEE, J., WANG, S. A feature-oriented requirements modelling language. In: INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 20th, 2012, Chicago, USA, **Proceedings**…, IEEE, 2012, p. 151-160.

SLJIVO, I., GALLINA, B., CARLSON, J., HANSSON, H., PURI, S. A method to generate reusable safety case fragments from compositional safety analysis. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE (ICSR 2015), 14th, 2015, **Proceedings**…, Springer Int. Publishing, Software Reuse for Dynamic Systems in the Cloud and Beyond, 2015, v. 8919, p. 253-268.

SLJIVO, I., GALLINA, B., CARLSON, J., HANSSON, H. Generation of safety case argument-fragments from safety contracts. In: INTERNATIONAL CONFERENCE ON

COMPUTER SAFETY, RELIABILITY, AND SECURITY (SAFECOMP), 33rd, 2014, Florence, Italy, **Proceedings**…, Springer Int. Publishing, LNCS, 2014, v. 8666, p. 170-185.

SLJIVO, I., GALLINA, B., CARLSON, J., HANSSON, H. Strong and weak contract formalism for third-party component reuse. In: INTERNATIONAL WORKSHOP ON SOFTWARE CERTIFICATION (WoSoCer), 3rd, 2013, Pasadena, USA, **Proceedings**…, IEEE, 2013, p. 359-364.

SPESXT, Software Platform Embedded Systems 2020 XT. Available on-line: <http://spes2020.informatik.tu-muenchen.de/ECl.html>. Accessed in: February, 22th, 2016.

SOCIETY OF AUTOMOTIVE ENGINEERS (SAE). SAE AS5506B: Architecture Analysis & Design Language (AADL). Society of Automotive Engineers, 2012, Available on-line: http://standards.sae.org/as5506b/, Accessed: January, 14th, 2016.

SOCIETY OF AUTOMOTIVE ENGINEERS (SAE), Aerospace Recommended Practice 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems. SAE, November, 1996.

SOCIETY OF AUTOMOTIVE ENGINEERS (SAE). ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. SAE 400 Commonwealth Drive, USA, 1994.

SOROKOS I., PAPADOPOULOS Y., AZEVEDO L., PARKER D., WALKER D. Automating allocation of development assurance levels: an extension to HiP-HOPS. In: IFAC INTERNATIONAL WORKSHOP ON DEPENDABLE CONTROL OF DISCRETE SYSTEMS (DCDS'15), 5th, 2015, **Proceedings**…, Elsevier, 2015, v. 48, i. 7, p. 9-14.

STEINER, E. M., MASIERO, P. C., BONIFÁCIO, R. Managing SPL variabilities in UAV Simulink models with Pure::variants and Hephaestus. **CLEI Electronic Journal**, v. 16, n. 1, 2013.

STEPHENSON, Z. R., DE SOUZA, S. McDERMID, J. A. Product line analysis and the system safety process. In: INTERNATIONAL SYSTEM SAFETY CONFERENCE, 22nd, **Proceedings**…, 2004, p. N/A.

STEPHENSON, A., BUTTLE, D., MCDERMID, J. A. Extending commonality analysis for embedded control system families. In: INTERNATIONAL WORKSHOP ON SOFTWARE ARCHITECTURES FOR PRODUCT FAMILIES (IW-SAPF), 2000, **Proceedings**…, Springer-Verlag London, p. 217-224.

SUN, L. Establishing confidence in safety assessment evidence. PhD Thesis, Department of Computer Science, The University of York, York, United Kingdom, 2013.

SVAHNBERG, M. BOSCH, J. Issues concerning variability in software product lines. In: INTERNATIONAL WORKSHOP ON SOFTWARE ARCHITECTURES FOR PRODUCT FAMILIES, 2000, Proceedings…, Springer-Heidelberg, LNCS, v. 1951, p. 146-157.

THE UNIVERSITY OF YORK. Freeware GSN Vision Add-on. Available on-line: <http://www.cs.york.ac.uk/~tpk/gsn/gsnaddoninstaller.zip>, Accessed in: January, 11th, 2016.

THIEL, S., HEIN, A. Systematic integration of variability into product-line architecture design. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINES, 2nd, 2002, San Diego, CA, USA, **Proceedings**…, Springer-Heidelberg, LCNC, v. 2379, p. 130-153.

TOULMIN, S. **The Uses of Argument**, Cambridge University Press, 1958.

TURNES, L., BONIFACIO, R., ALVES, V., LAMMEL, R. Techniques for developing a product line of product line tools: a comparative study. In: BRAZILIAN SYMPOSIUM ON SOFTWARE COMPONENTS, ARCHITECTURES AND REUSE (SBCARS), 5th, 2011, **Proceedings**…, IEEE, 2011, p. 11-20.

US MILITARY. Procedure for performing a failure mode effect and criticality analysis. United States military procedure. Technical report, number: MIL-STD-1629A, US Military, 1977.

VAN DER LINDEN, F. Software product families in europe: the esaps & café projects. **IEEE Software**, v. 19, n. 4, 2002.

VASILEVSKIY, A., HAUGEN, Ø., CHAUVEL, F., JOHANSEN, M. F., SHIMBARA, D. The BVR tool bundle to support product line engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINE (SPLC '15), 19th, 2015, **Proceedings**…, ACM, New York, NY, USA, p. 380-384.

VOELTER, M., VISSER, E. Product line engineering using domain-specific languages. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 15th, 2011, **Proceedings**…, IEEE, 2011, p.70-79.

WALKER M., REISER M. O., TUCCI-PIERGIOVANNI S., PAPADOPOULOS Y., LÖNN H., MRAIDHA C., PARKER D., CHEN D., SERVAT D. Automatic optimization of system architectures using EAST-ADL. **Journal of Systems and Software**, Elsevier, v. 86, i. 10, p. 2467-2487, October, 2013.

WALLACE, M. Modular architectural representation and analysis of fault propagation and transformation. In: INTERNATIONAL WORKSHOP ON FORMAL FOUNDATIONS OF EMBEDDED SOFTWARE AND COMPONENT-BASED SOFTWARE ARCHITECTURES (FESCA), 2nd, 2005, **Proceedings**…, Elsevier, Electronic Notes in Theoretical Computer Science, v. 141, n. 3, p. 53-71.

WEAVER, R. A. The safety of software – constructing and assuring arguments. PhD Thesis, Department of Computer Science, University of York, York, United Kingdom, 2003.

WEILAND, J. Configuring variant-rich automotive software architecture models. In: IEEE CONFERENCE ON AUTOMOTIVE ELECTRONICS, 2nd, 2006, **Proceedings**…, IEEE, 2006, p. 73-80.

WEILAND, J., MANHART, P. A classification of modeling variability in simulink. In: INTERNATIONAL WORKSHOP ON VARIABILITY MODELING OF SOFTWARE-INTENSIVE SYSTEMS (VAMOS'14), 8th, 2014, Nice, France, **Proceedings**…, ACM, New York, USA, p. N/A.

WEISS, D. M., ROBERT, C. T. **Product-Line Engineering: A Family-Based Software Development Process**, Addison-Wesley Professional, 1999.

WILKINSON, P. J. KELLY, T. K. Functional hazard analysis for highly integrated aerospace systems. In: IEE Seminar on Certification of Ground/Air Systems, 1998, London, United Kingdom, **Proceedings**…, IEEE, 1998, p. 4/1-4/6.

WOHLIN, C, RUNESSON, P., HÖST, M., OHLSSON, M. C., REGNELL, B., WESSLÉN, A. **Experimentation in software engineering: an introduction**, Kluwer Academic Publishers, 2000.

WORRELL, R. B., STACK, D.W., A SETS user manual for the fault tree analyst. Technical report, number: NUREG CR-04651, US Nuclear Regulatory Commission, 1978.

YAMAMOTO, S., Argument algebra: a formalization of assurance case development. In: JOINT CONFERENCE ON KNOWLEDGE-BASED SOFTWARE ENGINEERING, 11th, 2014, Volgograd, Russia, **Proceedings**…, Springer-Heidelberg, LNCS, 2014, v. 466, p. 717-725.

YE, F. Justifying the use of cots components within safety critical applications**.** PhD Thesis, Department of Computer Science, University of York, York, United Kingdom, 2005.

YWORKS. yEd Graph Editor. Available on-line: <https://www.yworks.com/products/yed>, Accessed in: January, 16th, 2016.

ZIADI, T., HÉLOUËT, L., JÉZÉQUEL, J. M. Towards a UML profile for software product lines. In: INTERNATIONAL WORKSHOP ON PRODUCT FAMILY ENGINEERING, 5th, Nov. 4-6, 2003, Siena, Italy, **Proceedings**…, Springer-Heidelberg, Software Product-Family Engineering, 2004, v. 3014, p. 129-139.

Appendix A

HiP-HOPS BVR Adapter

This document presents the source code for the BVR adapter developed for HiP-HOPS compositional safety analysis editor.

```java
package org.icmc.usp.br.labes.bvr.hiphopsdiagram.adapter2.editors;

import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import org.eclipse.draw2d.ColorConstants;
import org.eclipse.draw2d.IFigure;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.gef.EditPart;
import org.eclipse.gmf.runtime.diagram.ui.parts.IDiagramGraphicalViewer;
import org.eclipse.gmf.runtime.diagram.ui.parts.IDiagramWorkbenchPart;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.StructuredSelection;
import org.eclipse.swt.graphics.Color;
import org.eclipse.ui.IEditorPart;
import hiphops.presentation.HiphopsEditor;
import no.sintef.bvr.thirdparty.interfaces.editor.IBVREnabledEditor;

public class HipHopsBVREditor extends HiphopsEditor implements IBVREnabledEditor{
   /** The foreground color. */
    private Map<IFigure,Color> foregroundColor = new HashMap<IFigure,Color>();

    /** The background color. */
    private Map<IFigure,Color> backgroundColor = new HashMap<IFigure,Color>();

    @Override
    public void clearHighlighting() {
        for (Iterator<IFigure> it = foregroundColor.keySet().iterator();
              IFigure figure = (IFigure) it.next();
             figure.setForegroundColor((Color)foregroundColor.get(figure));
            figure.repaint();
        }
        for (Iterator<IFigure> it = backgroundColor.keySet().iterator();
              it.hasNext();) {
```

Figure A.1. HiP-HOPS BVR adapter source code part 1.

```
        IFigure figure = (IFigure) it.next();
        figure.setBackgroundColor((Color)backgroundColor.get(figure));
        figure.repaint();

      }
     foregroundColor.clear();
    backgroundColor.clear();
  }

  @Override
  public List<Object> getSelectedObjects() {
        ISelection selection =
        getSite().getSelectionProvider().getSelection();
        StructuredSelection structuredSelection = (StructuredSelection)
        selection;
        return structuredSelection.toList();
  }
  @Override
  public void highlightObject(Object object, int type) {
        if(!(object instanceof EObject))
              return;

        EObject eObject = (EObject) object;
        Color c = ColorConstants.black;
        switch (type) {
              case IBVREnabledEditor.HL_PLACEMENT :
                    c = IBVREnabledEditor.PLACEMENT; break;
              case IBVREnabledEditor.HL_PLACEMENT_OUT :
                    c = IBVREnabledEditor.PLACEMENT_OUT; break;
              case IBVREnabledEditor.HL_PLACEMENT_IN :
                    c = IBVREnabledEditor.PLACEMENT_IN; break;
              case IBVREnabledEditor.HL_PLACEMENT_IN_OUT :
                    c = IBVREnabledEditor.PLACEMENT_IN_OUT; break;
              case IBVREnabledEditor.HL_REPLACEMENT :
                    c = IBVREnabledEditor.REPLACEMENT; break;
              case IBVREnabledEditor.HL_REPLACEMENT_OUT :
                    c = IBVREnabledEditor.REPLACEMENT_OUT; break;
              case IBVREnabledEditor.HL_REPLACEMENT_IN :
                    c = IBVREnabledEditor.REPLACEMENT_IN; break;
              case IBVREnabledEditor.HL_REPLACEMENT_IN_OUT :
                    c = IBVREnabledEditor.REPLACEMENT_IN_OUT; break;
              default :
                    throw new UnsupportedOperationException("coloring of
                   this type is not supported " + type);
        }
        setColor(eObject, c, getActiveEditor());
  }
  @Override
  public void selectObjects(List<Object> objects) {
        throw new UnsupportedOperationException("not implemented");
  }
   public void setColor(EObject obj, Color fg, IEditorPart editor) {
        IDiagramGraphicalViewer gv =
         ((IDiagramWorkbenchPart)editor).getDiagramGraphicalViewer();
         List<?> editParts =
         gv.findEditPartsForElement(IDProvider.getXMIId(obj), EditPart.class);
```

Figure A.2. HiP-HOPS BVR adapter part 2.

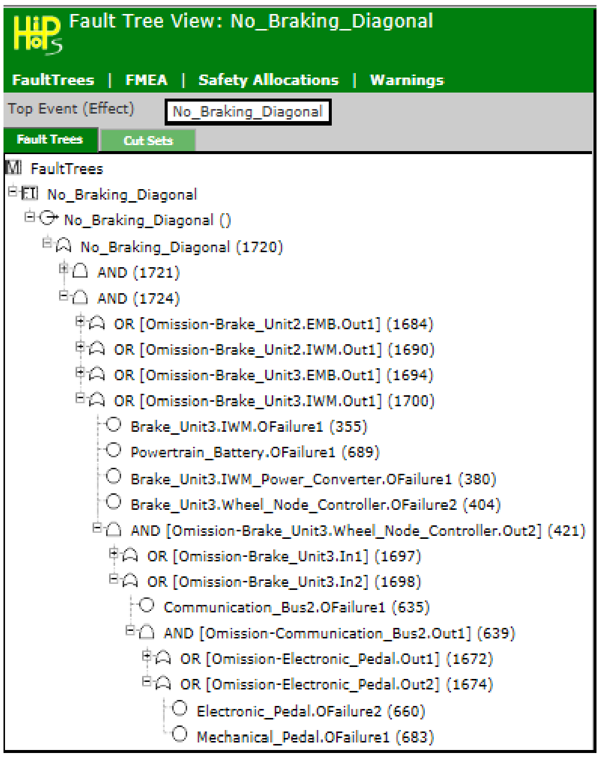
```
        for (Object object : editParts) {
            if(object instanceof IFigure){
               IFigure ep = (IFigure) object;
                 if (!foregroundColor.containsKey(ep)){
                     foregroundColor.put(ep, ep.getForegroundColor());
                 }
                 ep.setForegroundColor(fg);
                 ep.repaint();
            }
        }
    }

    @Override
    public List<EObject> getModelObjects(List<Object> objects) {
          return null;
    }

    @Override
    public List<Object> getGraphicalObjects(List<EObject> objects) {
          return null;
    }
}
```

Figure A.3. HiP-HOPS BVR adapter part 3.

Appendix B

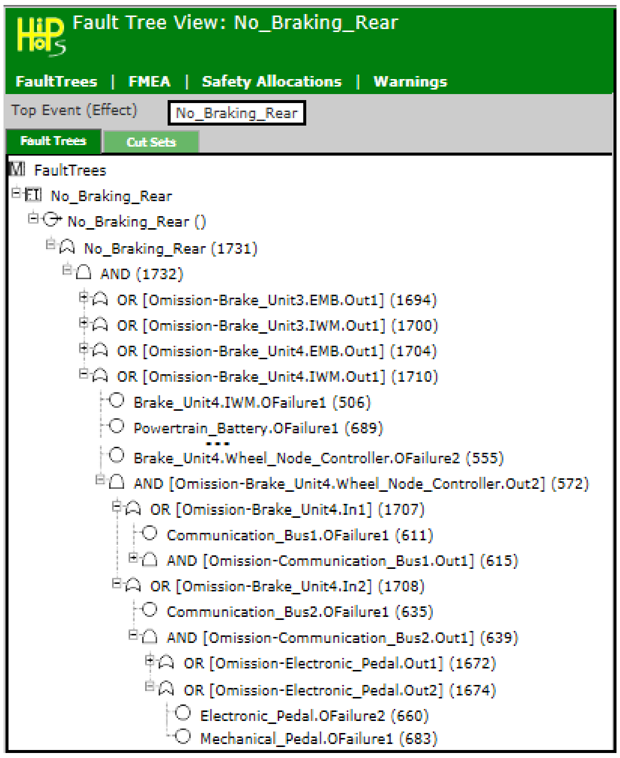
HBS-SPL Case Study Outcomes

This document presents the fault trees and FMEA results generated for the four wheel braking system variant with the support of HiP-HOPS compositional safety analysis tool. Figures B.1 and B.2 shows excerpts of the six fault trees generated for the four wheel braking (4WB) variant. Figures B.3, B.4, and B.5 shows the FMEA results for the 4WB system variant.


Fault Tree View: No_Braking_Diagonal
FaultTrees | FMEA | Safety Allocations | Warnings
Fault Trees
Cut Sets

```
Top Event (Effect)   No_Braking_Diagonal

FaultTrees
 No_Braking_Diagonal
   No_Braking_Diagonal ()
     No_Braking_Diagonal (1720)
       AND (1721)
       AND (1724)
         OR [Omission-Brake_Unit2.EMB.Out1] (1684)
         OR [Omission-Brake_Unit2.IWM.Out1] (1690)
         OR [Omission-Brake_Unit3.EMB.Out1] (1694)
         OR [Omission-Brake_Unit3.IWM.Out1] (1700)
           Brake_Unit3.IWM.OFailure1 (355)
           Powertrain_Battery.OFailure1 (689)
           Brake_Unit3.IWM_Power_Converter.OFailure1 (380)
           Brake_Unit3.Wheel_Node_Controller.OFailure2 (404)
           AND [Omission-Brake_Unit3.Wheel_Node_Controller.Out2] (421)
             OR [Omission-Brake_Unit3.In1] (1697)
             OR [Omission-Brake_Unit3.In2] (1698)
               Communication_Bus2.OFailure1 (635)
               AND [Omission-Communication_Bus2.Out1] (639)
                 OR [Omission-Electronic_Pedal.Out1] (1672)
                 OR [Omission-Electronic_Pedal.Out2] (1674)
                   Electronic_Pedal.OFailure2 (660)
                   Mechanical_Pedal.OFailure1 (683)
```

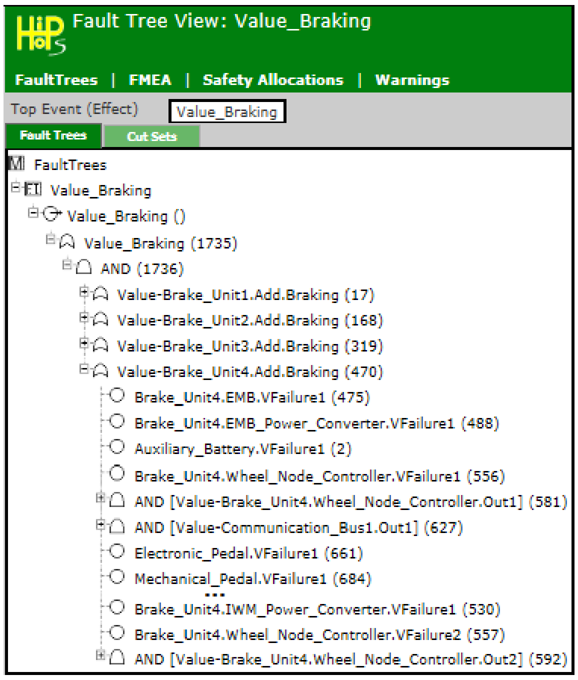
(a) 4WB no braking diagonal fault tree


Fault Tree View: No_Braking_Rear
FaultTrees | FMEA | Safety Allocations | Warnings
Fault Trees
Cut Sets

```
Top Event (Effect)   No_Braking_Rear

FaultTrees
 No_Braking_Rear
   No_Braking_Rear ()
     No_Braking_Rear (1731)
       AND (1732)
         OR [Omission-Brake_Unit3.EMB.Out1] (1694)
         OR [Omission-Brake_Unit3.IWM.Out1] (1700)
         OR [Omission-Brake_Unit4.EMB.Out1] (1704)
         OR [Omission-Brake_Unit4.IWM.Out1] (1710)
           Brake_Unit4.IWM.OFailure1 (506)
           Powertrain_Battery.OFailure1 (689)
           ...
           Brake_Unit4.Wheel_Node_Controller.OFailure2 (555)
           AND [Omission-Brake_Unit4.Wheel_Node_Controller.Out2] (572)
             OR [Omission-Brake_Unit4.In1] (1707)
               Communication_Bus1.OFailure1 (611)
               AND [Omission-Communication_Bus1.Out1] (615)
             OR [Omission-Brake_Unit4.In2] (1708)
               Communication_Bus2.OFailure1 (635)
               AND [Omission-Communication_Bus2.Out1] (639)
                 OR [Omission-Electronic_Pedal.Out1] (1672)
                 OR [Omission-Electronic_Pedal.Out2] (1674)
                   Electronic_Pedal.OFailure2 (660)
                   Mechanical_Pedal.OFailure1 (683)
```

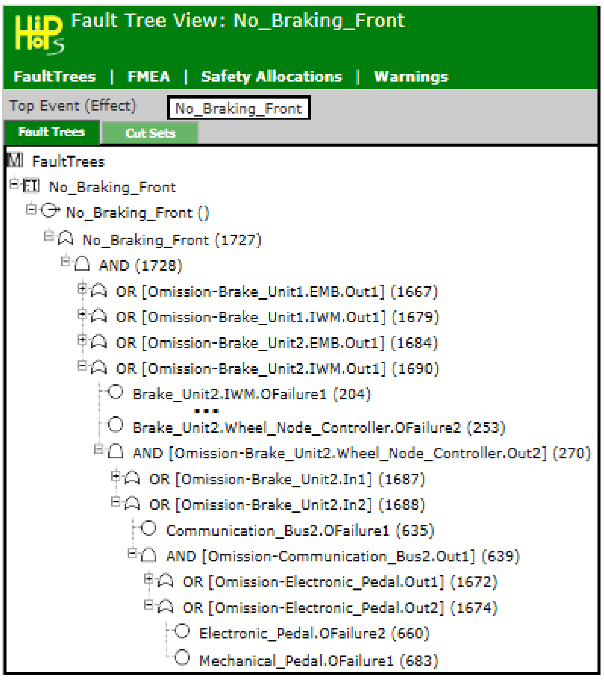
(b) 4WB no braking rear fault tree

Figure B.1. 4WB fault trees part 1.


Fault Tree View: Value_Braking
FaultTrees | FMEA | Safety Allocations | Warnings
Top Event (Effect)
Fault Trees
Cut Sets

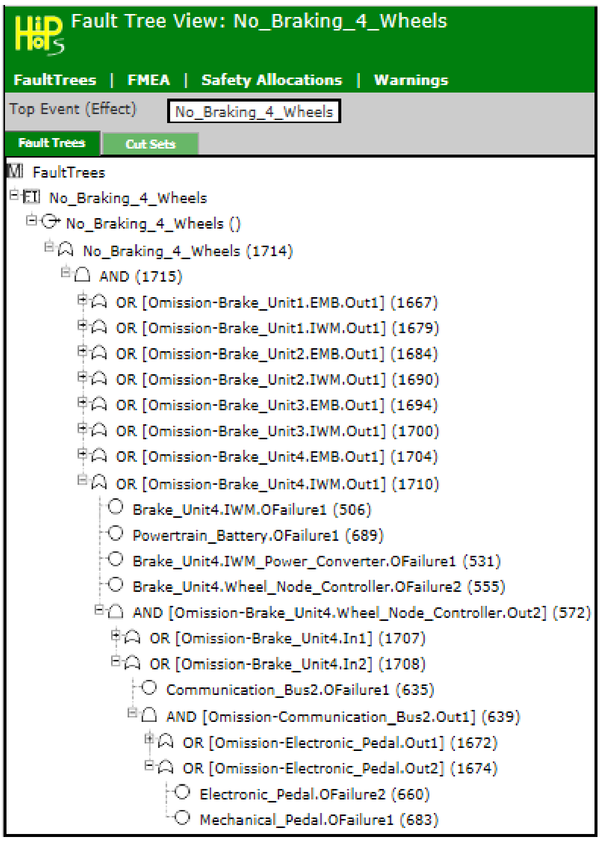
```
FaultTrees
Value_Braking
  Value_Braking ()
    Value_Braking (1735)
      AND (1736)
        Value-Brake_Unit1.Add.Braking (17)
        Value-Brake_Unit2.Add.Braking (168)
        Value-Brake_Unit3.Add.Braking (319)
        Value-Brake_Unit4.Add.Braking (470)
          Brake_Unit4.EMB.VFailure1 (475)
          Brake_Unit4.EMB_Power_Converter.VFailure1 (488)
          Auxiliary_Battery.VFailure1 (2)
          Brake_Unit4.Wheel_Node_Controller.VFailure1 (556)
          AND [Value-Brake_Unit4.Wheel_Node_Controller.Out1] (581)
          AND [Value-Communication_Bus1.Out1] (627)
          Electronic_Pedal.VFailure1 (661)
          Mechanical_Pedal.VFailure1 (684)
          ...
          Brake_Unit4.IWM_Power_Converter.VFailure1 (530)
          Brake_Unit4.Wheel_Node_Controller.VFailure2 (557)
          AND [Value-Brake_Unit4.Wheel_Node_Controller.Out2] (592)
```

(c) 4WB value braking fault tree


Fault Tree View: No_Braking_Front
FaultTrees | FMEA | Safety Allocations | Warnings
Top Event (Effect)
Fault Trees
Cut Sets

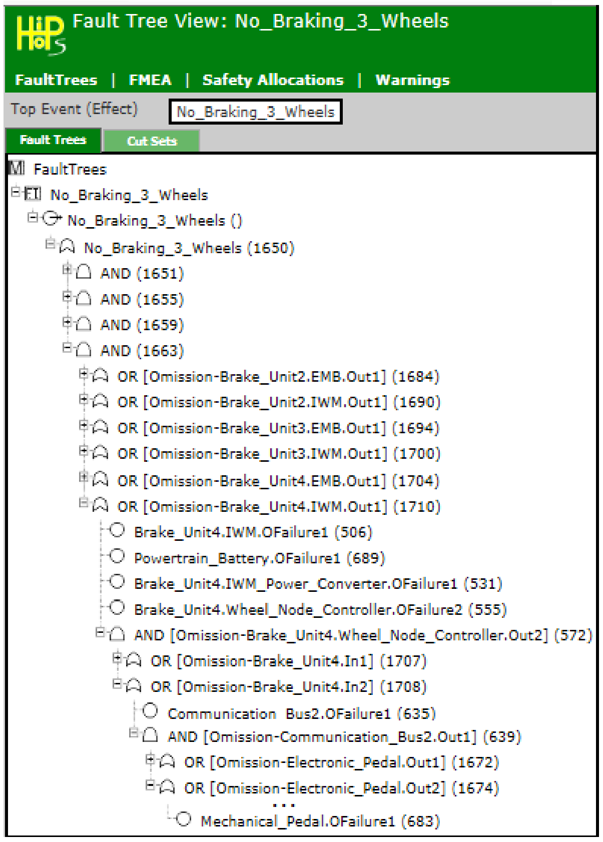
```
FaultTrees
No_Braking_Front
  No_Braking_Front ()
    No_Braking_Front (1727)
      AND (1728)
        OR [Omission-Brake_Unit1.EMB.Out1] (1667)
        OR [Omission-Brake_Unit1.IWM.Out1] (1679)
        OR [Omission-Brake_Unit2.EMB.Out1] (1684)
        OR [Omission-Brake_Unit2.IWM.Out1] (1690)
          Brake_Unit2.IWM.OFailure1 (204)
          ...
          Brake_Unit2.Wheel_Node_Controller.OFailure2 (253)
          AND [Omission-Brake_Unit2.Wheel_Node_Controller.Out2] (270)
            OR [Omission-Brake_Unit2.In1] (1687)
            OR [Omission-Brake_Unit2.In2] (1688)
              Communication_Bus2.OFailure1 (635)
              AND [Omission-Communication_Bus2.Out1] (639)
                OR [Omission-Electronic_Pedal.Out1] (1672)
                OR [Omission-Electronic_Pedal.Out2] (1674)
                  Electronic_Pedal.OFailure2 (660)
                  Mechanical_Pedal.OFailure1 (683)
```

(d) 4WB no braking front fault tree


Fault Tree View: No_Braking_4_Wheels
FaultTrees | FMEA | Safety Allocations | Warnings
Top Event (Effect)
Fault Trees
Cut Sets

```
FaultTrees
No_Braking_4_Wheels
  No_Braking_4_Wheels ()
    No_Braking_4_Wheels (1714)
      AND (1715)
        OR [Omission-Brake_Unit1.EMB.Out1] (1667)
        OR [Omission-Brake_Unit1.IWM.Out1] (1679)
        OR [Omission-Brake_Unit2.EMB.Out1] (1684)
        OR [Omission-Brake_Unit2.IWM.Out1] (1690)
        OR [Omission-Brake_Unit3.EMB.Out1] (1694)
        OR [Omission-Brake_Unit3.IWM.Out1] (1700)
        OR [Omission-Brake_Unit4.EMB.Out1] (1704)
        OR [Omission-Brake_Unit4.IWM.Out1] (1710)
          Brake_Unit4.IWM.OFailure1 (506)
          Powertrain_Battery.OFailure1 (689)
          Brake_Unit4.IWM_Power_Converter.OFailure1 (531)
          Brake_Unit4.Wheel_Node_Controller.OFailure2 (555)
          AND [Omission-Brake_Unit4.Wheel_Node_Controller.Out2] (572)
            OR [Omission-Brake_Unit4.In1] (1707)
            OR [Omission-Brake_Unit4.In2] (1708)
              Communication_Bus2.OFailure1 (635)
              AND [Omission-Communication_Bus2.Out1] (639)
                OR [Omission-Electronic_Pedal.Out1] (1672)
                OR [Omission-Electronic_Pedal.Out2] (1674)
                  Electronic_Pedal.OFailure2 (660)
                  Mechanical_Pedal.OFailure1 (683)
```

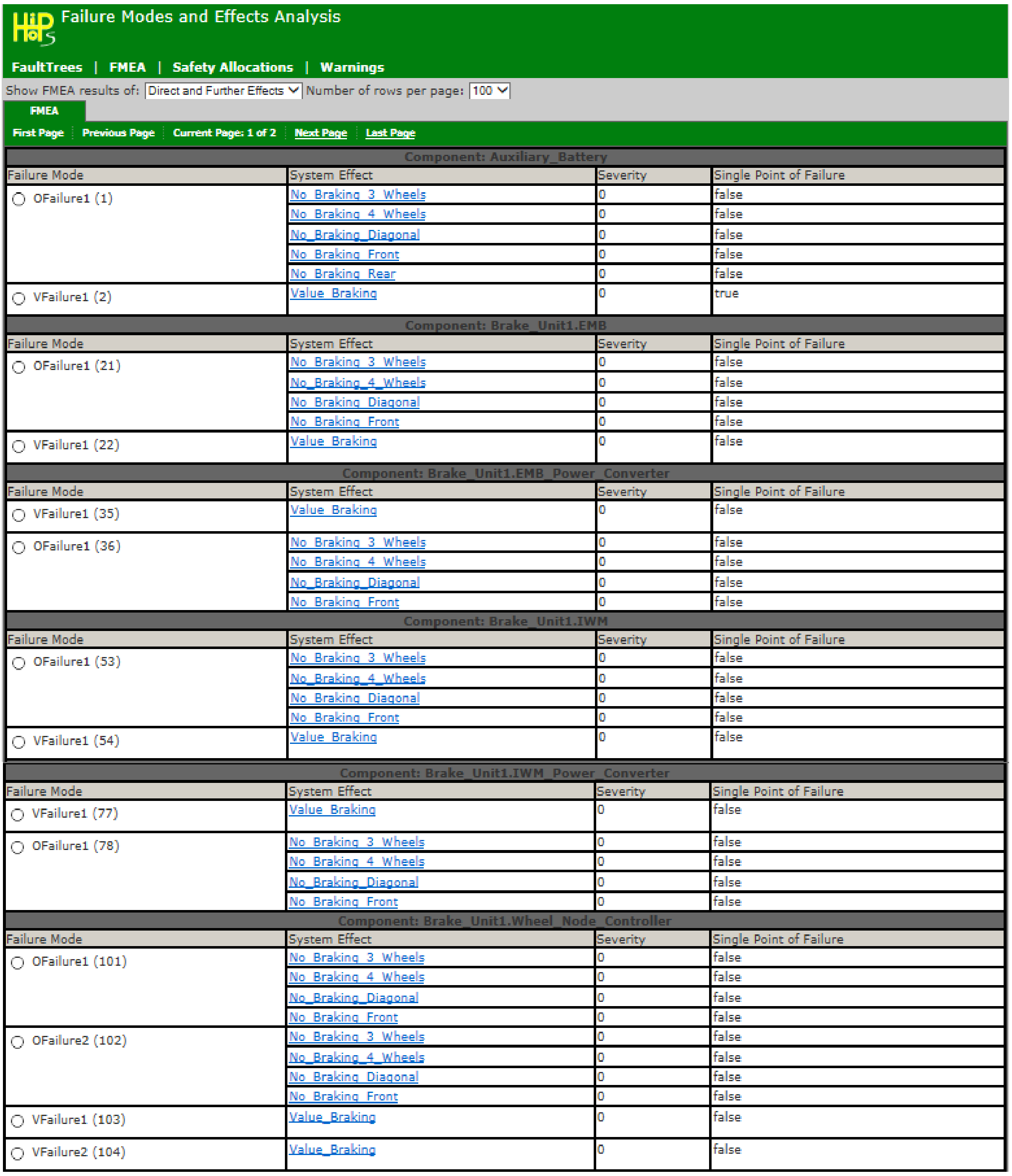
(e) 4WB no braking four wheels fault tree


Fault Tree View: No_Braking_3_Wheels
FaultTrees | FMEA | Safety Allocations | Warnings
Top Event (Effect)
Fault Trees
Cut Sets

```
FaultTrees
No_Braking_3_Wheels
  No_Braking_3_Wheels ()
    No_Braking_3_Wheels (1650)
      AND (1651)
      AND (1655)
      AND (1659)
      AND (1663)
        OR [Omission-Brake_Unit2.EMB.Out1] (1684)
        OR [Omission-Brake_Unit2.IWM.Out1] (1690)
        OR [Omission-Brake_Unit3.EMB.Out1] (1694)
        OR [Omission-Brake_Unit3.IWM.Out1] (1700)
        OR [Omission-Brake_Unit4.EMB.Out1] (1704)
        OR [Omission-Brake_Unit4.IWM.Out1] (1710)
          Brake_Unit4.IWM.OFailure1 (506)
          Powertrain_Battery.OFailure1 (689)
          Brake_Unit4.IWM_Power_Converter.OFailure1 (531)
          Brake_Unit4.Wheel_Node_Controller.OFailure2 (555)
          AND [Omission-Brake_Unit4.Wheel_Node_Controller.Out2] (572)
            OR [Omission-Brake_Unit4.In1] (1707)
            OR [Omission-Brake_Unit4.In2] (1708)
              Communication_Bus2.OFailure1 (635)
              AND [Omission-Communication_Bus2.Out1] (639)
                OR [Omission-Electronic_Pedal.Out1] (1672)
                OR [Omission-Electronic_Pedal.Out2] (1674)
                  ...
                  Mechanical_Pedal.OFailure1 (683)
```

(f) 4WB no braking three wheels fault tree

Figure B.2. 4WB fault trees part 2.


Failure Modes and Effects Analysis
FaultTrees | FMEA | Safety Allocations | Warnings
FMEA
First Page
Previous Page
Current Page: 1 of 2
Next Page
Last Page

Show FMEA results of: Direct and Further Effects Number of rows per page: 100

Component: Auxiliary_Battery

Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (1)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
	No_Braking_Rear	0	false
VFailure1 (2)	Value_Braking	0	true

Component: Brake_Unit1.EMB

Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (21)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
VFailure1 (22)	Value_Braking	0	false

Component: Brake_Unit1.EMB_Power_Converter

Failure Mode	System Effect	Severity	Single Point of Failure
VFailure1 (35)	Value_Braking	0	false
OFailure1 (36)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false

Component: Brake_Unit1.IWM

Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (53)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
VFailure1 (54)	Value_Braking	0	false

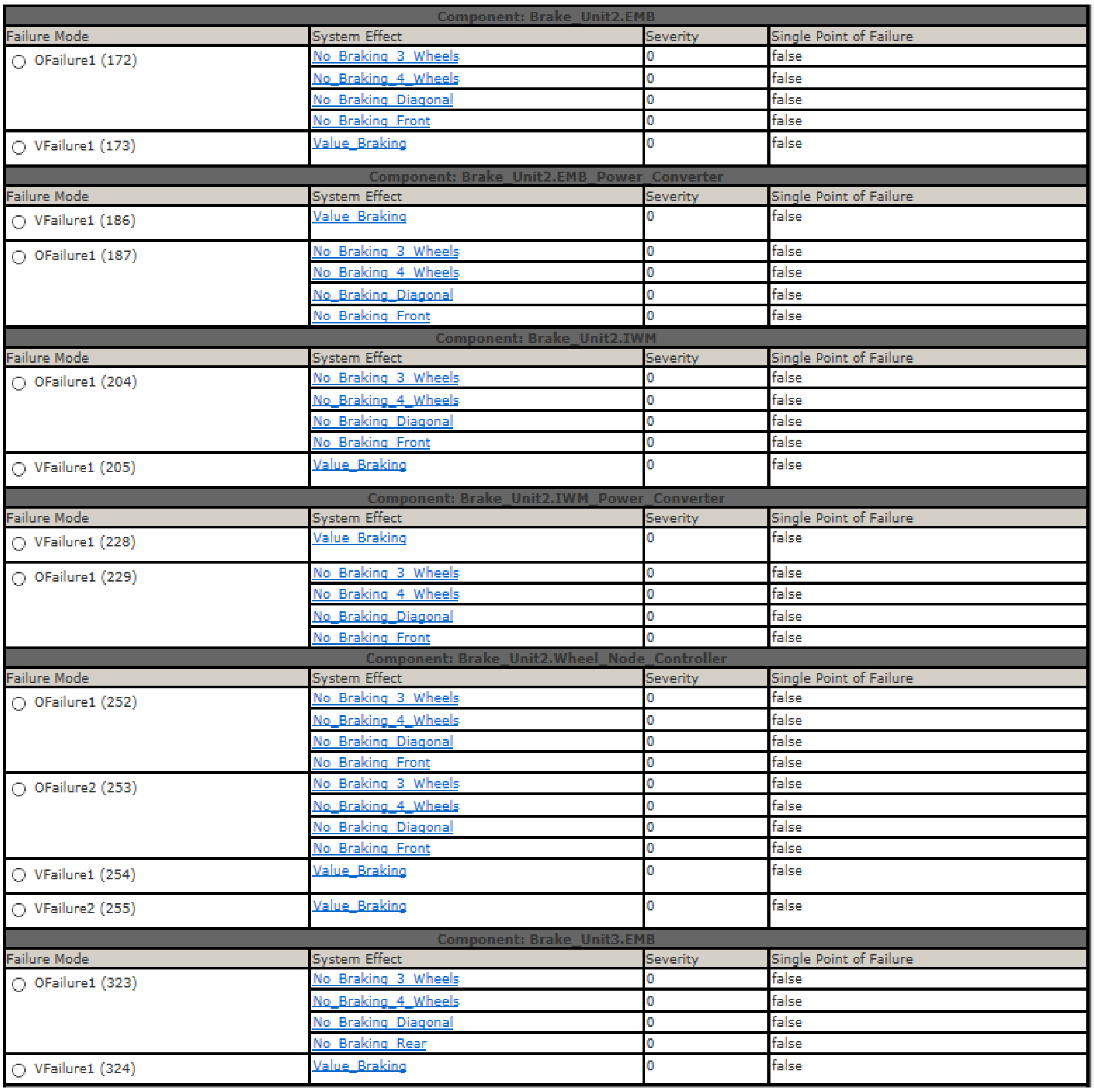
Component: Brake_Unit1.IWM_Power_Converter

Failure Mode	System Effect	Severity	Single Point of Failure
VFailure1 (77)	Value_Braking	0	false
OFailure1 (78)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false

Component: Brake_Unit1.Wheel_Node_Controller

Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (101)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
OFailure2 (102)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
VFailure1 (103)	Value_Braking	0	false
VFailure2 (104)	Value_Braking	0	false

Figure B.3. 4WB failure modes and effects analysis results part 1.



Component: Brake_Unit2.EMB

Failure Mode	System Effect	Severity	Single Point of Failure
○ OFailure1 (172)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
○ VFailure1 (173)	Value_Braking	0	false

Component: Brake_Unit2.EMB_Power_Converter

Failure Mode	System Effect	Severity	Single Point of Failure
○ VFailure1 (186)	Value_Braking	0	false
○ OFailure1 (187)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false

Component: Brake_Unit2.IWM

Failure Mode	System Effect	Severity	Single Point of Failure
○ OFailure1 (204)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
○ VFailure1 (205)	Value_Braking	0	false

Component: Brake_Unit2.IWM_Power_Converter

Failure Mode	System Effect	Severity	Single Point of Failure
○ VFailure1 (228)	Value_Braking	0	false
○ OFailure1 (229)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false

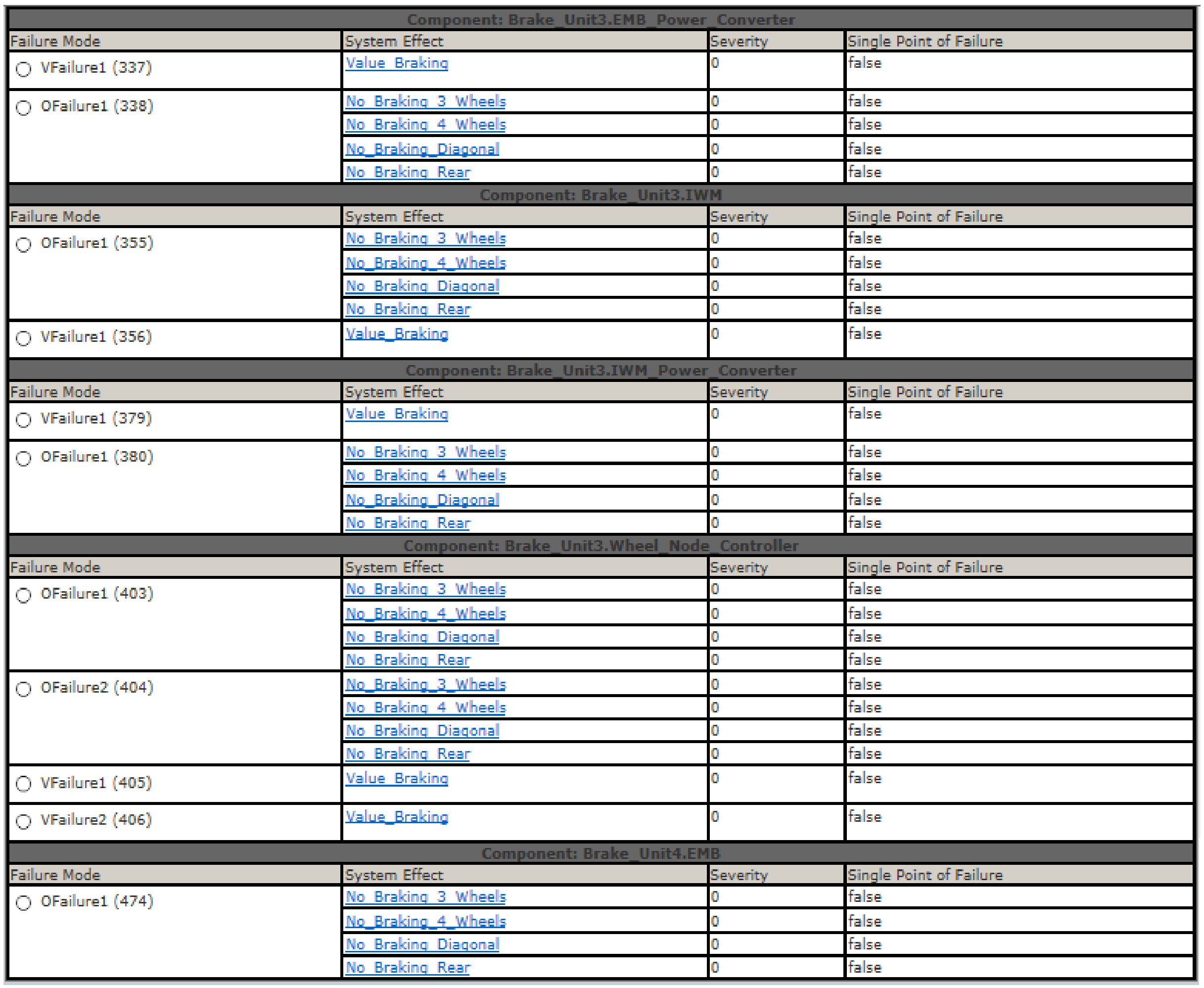
Component: Brake_Unit2.Wheel_Node_Controller

Failure Mode	System Effect	Severity	Single Point of Failure
○ OFailure1 (252)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
○ OFailure2 (253)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Front	0	false
○ VFailure1 (254)	Value_Braking	0	false
○ VFailure2 (255)	Value_Braking	0	false

Component: Brake_Unit3.EMB

Failure Mode	System Effect	Severity	Single Point of Failure
○ OFailure1 (323)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false
○ VFailure1 (324)	Value_Braking	0	false

Figure B.4. 4WB failure modes and effects analysis results part 2.



Component: Brake_Unit3.EMB_Power_Converter			
Failure Mode	System Effect	Severity	Single Point of Failure
VFailure1 (337)	Value_Braking	0	false
OFailure1 (338)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false
Component: Brake_Unit3.IWM			
Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (355)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false
VFailure1 (356)	Value_Braking	0	false
Component: Brake_Unit3.IWM_Power_Converter			
Failure Mode	System Effect	Severity	Single Point of Failure
VFailure1 (379)	Value_Braking	0	false
OFailure1 (380)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false
Component: Brake_Unit3.Wheel_Node_Controller			
Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (403)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false
OFailure2 (404)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false
VFailure1 (405)	Value_Braking	0	false
VFailure2 (406)	Value_Braking	0	false
Component: Brake_Unit4.EMB			
Failure Mode	System Effect	Severity	Single Point of Failure
OFailure1 (474)	No_Braking_3_Wheels	0	false
	No_Braking_4_Wheels	0	false
	No_Braking_Diagonal	0	false
	No_Braking_Rear	0	false

Figure B.5. 4WB failure modes and effects analysis results part 3.

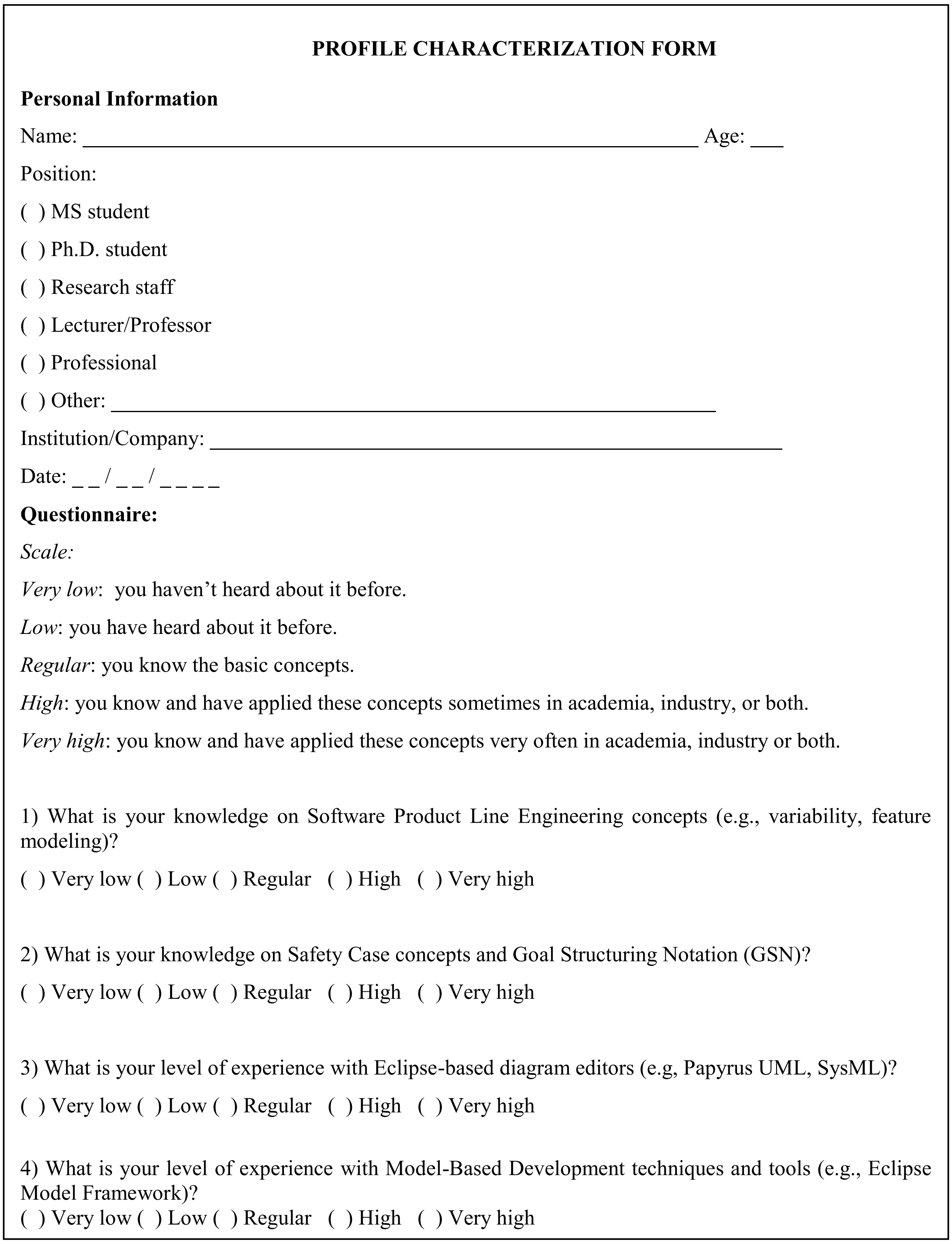
Appendix C

EXPERIMENTAL STUDY DOCUMENTS

The documents used throughout the preparation and operation of the experimental study presented in Chapter 6 are detailed in this appendix.

C.1 Preparation

Figure C.1 shows the profile characterization, and Figure C.2 shows the consent form provided to the participants during the preparation of the experimental study.



PROFILE CHARACTERIZATION FORM

Personal Information

Name: ____ Age: ___

Position:

( ) MS student

( ) Ph.D. student

( ) Research staff

( ) Lecturer/Professor

( ) Professional

( ) Other: ____

Institution/Company: ____

Date: _ _ / _ _ / _ _ _ _

Questionnaire:

Scale:

Very low: you haven't heard about it before.

Low: you have heard about it before.

Regular: you know the basic concepts.

High: you know and have applied these concepts sometimes in academia, industry, or both.

Very high: you know and have applied these concepts very often in academia, industry or both.

1) What is your knowledge on Software Product Line Engineering concepts (e.g., variability, feature modeling)?

( ) Very low ( ) Low ( ) Regular ( ) High ( ) Very high

2) What is your knowledge on Safety Case concepts and Goal Structuring Notation (GSN)?

( ) Very low ( ) Low ( ) Regular ( ) High ( ) Very high

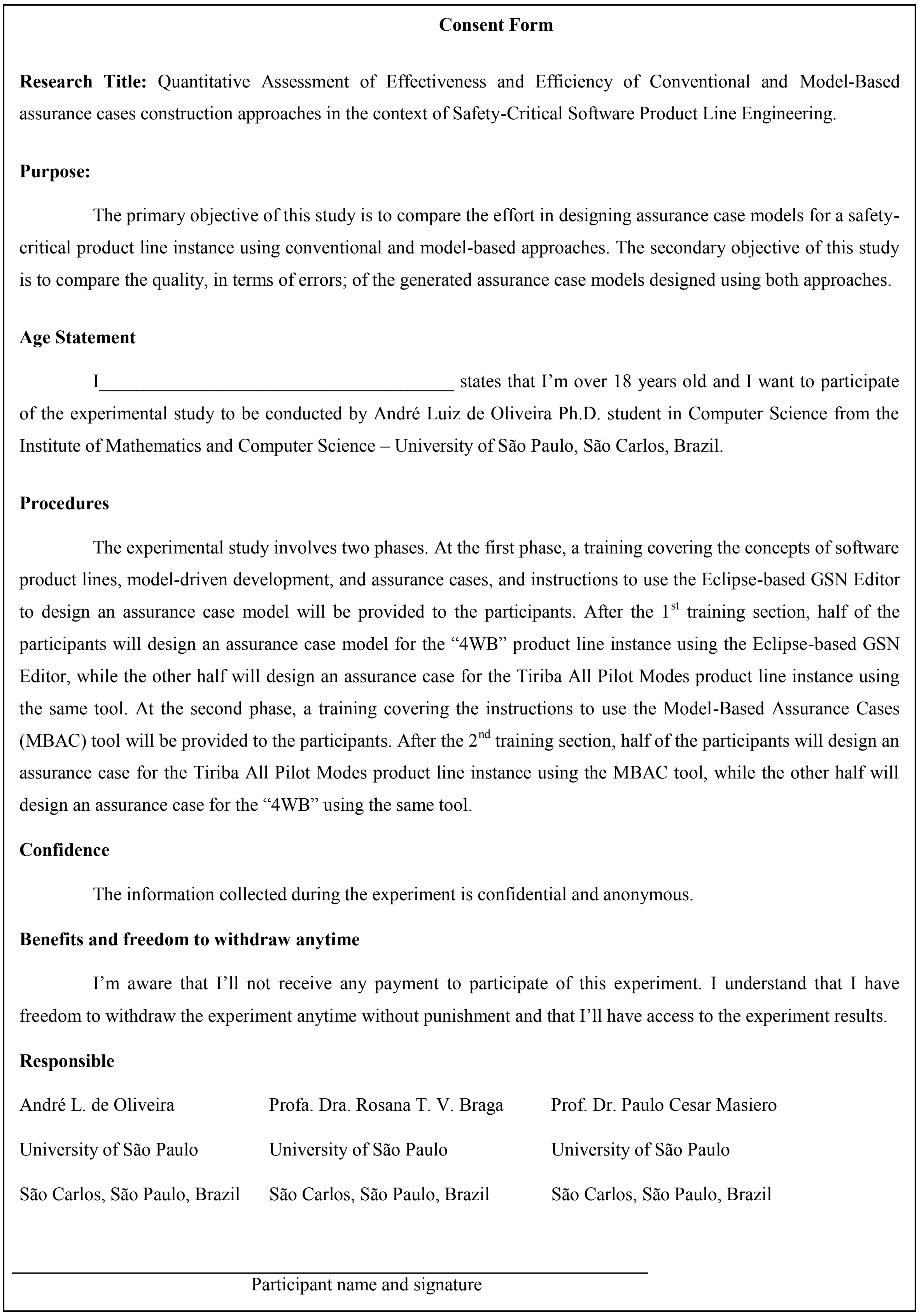
3) What is your level of experience with Eclipse-based diagram editors (e.g, Papyrus UML, SysML)?

( ) Very low ( ) Low ( ) Regular ( ) High ( ) Very high

4) What is your level of experience with Model-Based Development techniques and tools (e.g., Eclipse Model Framework)?

( ) Very low ( ) Low ( ) Regular ( ) High ( ) Very high

Figure C.1 Profile characterization form.



Consent Form

Research Title: Quantitative Assessment of Effectiveness and Efficiency of Conventional and Model-Based assurance cases construction approaches in the context of Safety-Critical Software Product Line Engineering.

Purpose:

The primary objective of this study is to compare the effort in designing assurance case models for a safety-critical product line instance using conventional and model-based approaches. The secondary objective of this study is to compare the quality, in terms of errors; of the generated assurance case models designed using both approaches.

Age Statement

I______________________________________ states that I'm over 18 years old and I want to participate of the experimental study to be conducted by André Luiz de Oliveira Ph.D. student in Computer Science from the Institute of Mathematics and Computer Science – University of São Paulo, São Carlos, Brazil.

Procedures

The experimental study involves two phases. At the first phase, a training covering the concepts of software product lines, model-driven development, and assurance cases, and instructions to use the Eclipse-based GSN Editor to design an assurance case model will be provided to the participants. After the 1st training section, half of the participants will design an assurance case model for the "4WB" product line instance using the Eclipse-based GSN Editor, while the other half will design an assurance case for the Tiriba All Pilot Modes product line instance using the same tool. At the second phase, a training covering the instructions to use the Model-Based Assurance Cases (MBAC) tool will be provided to the participants. After the 2nd training section, half of the participants will design an assurance case for the Tiriba All Pilot Modes product line instance using the MBAC tool, while the other half will design an assurance case for the "4WB" using the same tool.

Confidence

The information collected during the experiment is confidential and anonymous.

Benefits and freedom to withdraw anytime

I'm aware that I'll not receive any payment to participate of this experiment. I understand that I have freedom to withdraw the experiment anytime without punishment and that I'll have access to the experiment results.

Responsible

André L. de Oliveira	Profa. Dra. Rosana T. V. Braga	Prof. Dr. Paulo Cesar Masiero
University of São Paulo	University of São Paulo	University of São Paulo
São Carlos, São Paulo, Brazil	São Carlos, São Paulo, Brazil	São Carlos, São Paulo, Brazil

____

Participant name and signature

Figure C.2. Consent form.

C.2 Operation

The instructions to perform the assurance case construction tasks (Figures C.3, C.4 and C.6), the data collection forms (Figures C.5 and C.7), the supporting manuals for the Eclipse-based GSN editor (Figures C.8 and C.9) and MBAC tool (Figures C.10, C.11 and C.12), and the post-experiment questionnaire (Figures C.13 and C.14) used by the participants during the execution of the experiment, are presented in the following.

CONVENTIONAL APPROACH: ASSURANCE CASE CONSTRUCTION TASK

Group: ___

Phase: ___

Participant ID: ____

Task: Based on the design and safety assessment models from the targeted product variant, ( ) "*All Pilot Modes*" Tiriba flight control variant or ( ) "Four Wheel Braking" hybrid braking system variant, and a fragment of "*Hazard Avoidance*" assurance case pattern (KELLY and McDERMID, 1997), create an assurance case that argues the safety of the targeted product variant using the Conventional Approach with the support of the **Eclipse-based GSN Editor**.

Instructions to perform the assurance case construction task described above:

The fragment of "*Hazard Avoidance*" pattern shown in Figure 1 provides the template structure to design an assurance case model for the "*All Pilot Modes*" variant.

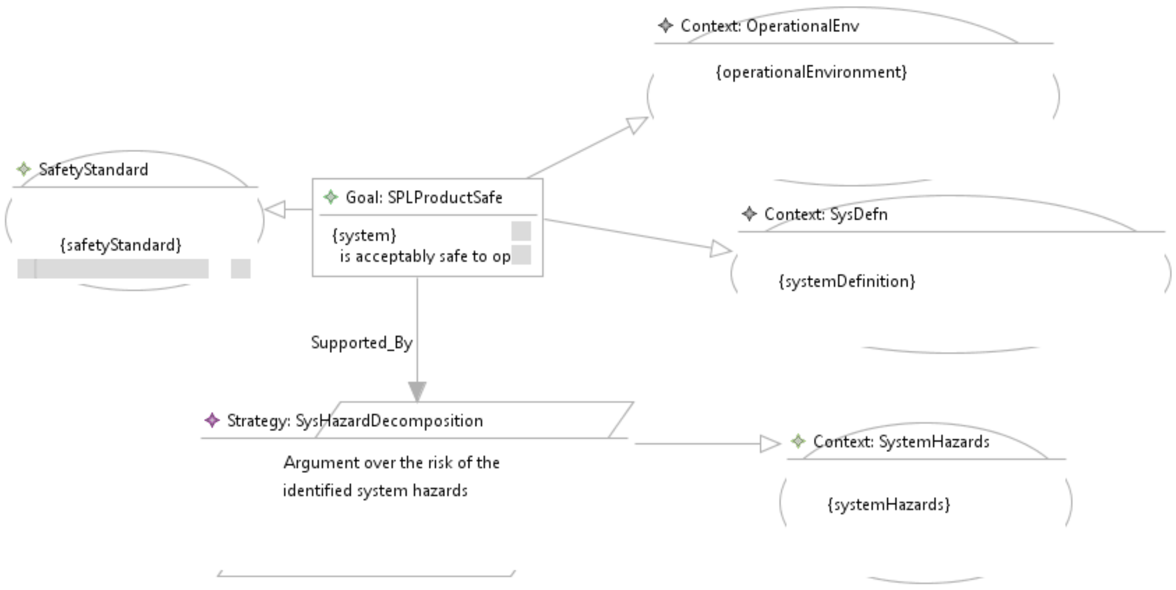


Figure 1. Hazard avoidance pattern in GSN.

Figure C.3. Instructions to perform the assurance case construction task using the conventional approach Part 1.

The elements between "{}" represent for the data required from the system models to instantiate the placeholders/abstract terms from the pattern. This data can be found in feature (.featureModel file extension), context (.context file extension), architecture (i.e., .core file extension), and failure (.hiphops file extension) models associated to the targeted product line instance. These models are located on the targeted project, e.g., "*GSNEditor_TiribaAllPilotModes*" project or "*GSN_EditorHBS4WB"* project, in the experiment workspace.

Table B.1 shows the mapping links between assurance case pattern abstract terms, the data required for their instantiation, and the system model name where the data can be found. This information should be considered for the assurance case construction task using the Eclipse-based GSN editor tooling support.

Table B.1 – Mapping links between abstract terms and system models.

Pattern Role	Data	Model
system	AadlSpec.name	aadlModel (.core)
safetyStandard	Model.description	errorModel (.hiphops)
systemDefinition	A set of Feature.name	featureModel (.featureModel)
operationalEnvironment	A set of ContextFeature.name	contextModel (.context)
systemHazards	A set of Hazard.Name	errorModel (.hiphops)

Figure C.4. Instructions to perform the assurance case construction task using the conventional approach Part 2.

CONVENTIONAL APPROACH: DATA COLLECTION FORM

Pattern Element	***Start Time (min:sec)***	***End Time (min:sec)***	***Total Time (min:sec)***
system			
safetyStandard			
systemDefinition			
operationalEnvironment			
systemHazards			
		Total Time:	

Figure C.5. Data collection form for the conventional approach.

MBAC APPROACH: ASSURANCE CASE CONSTRUCTION TASK

Group: ___

Phase: ___

Participant ID: ____

Task: Based on the design and safety assessment models from the "*Four Wheel Braking*" hybrid braking system variant, a fragment of the "*Hazard Avoidance*" assurance case pattern (KELLY and McDERMID, 1997), and the weaving model; configure the **MBAC Eclipse-based tool** to generate an assurance case that argues the safety of the "*Four Wheel Braking*" product variant.

Instructions to configure the MBAC instantiation program to perform the task described above:

Table 1 shows the variables with their respective values that should be created to configure the MBAC instantiation program to generate an assurance case arguing the safety of the "Four Wheels Braking" variant.

Table 1 – Configuration settings for the instantiation program.

Type of Model	**Variable Name**	**Aliases**	**Model File Path**	**Metamodel**
GraphML Muddle	Weaving	-	/SPL/weaving_hbs.graphml	-
EMF Model	errorModel	design	/SPL/models/HBS_4WB.hiphops	http://hiphops
EMF Model	contextModel	design	/SPL/models/HBS_4WB.context	http://context
EMF Model	featureModel	design	/SPL/models/HBS_4WB.featuremodel	http://featuremodel
EMF Model	aadlModel	design	/SPL/models/HBS_4WB.core	http://core
Plaim XML	patternInstantiation	pattern	/SPL/patterns/Sys_Safe.gsnml	-

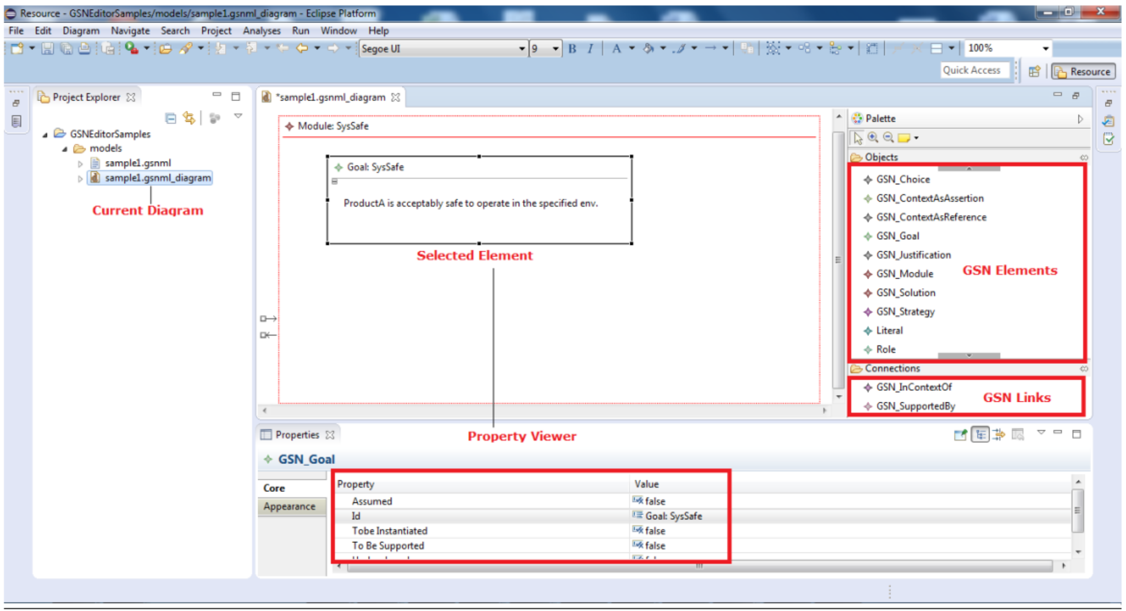
Figure C6. Instructions to perform the assurance case construction task using the MBAC approach.

MBAC APPROACH: DATA COLLECTION FORM

Variable Name	***Start Time (min:sec)***	***End Time (min:sec)***	***Total Time (min:sec)***
weaving			
errorModel			
contextModel			
featureModel			
aadlModel			
patternInstantiation			
		Total Time:	

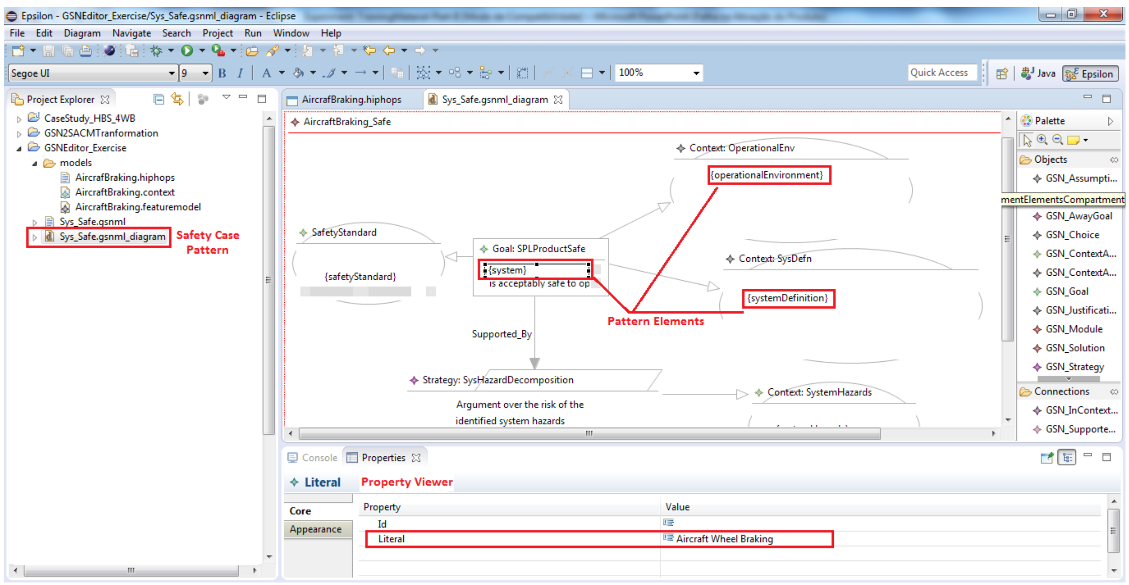
Figure C.7. Data collection form for the MBAC.

ECLIPSE-BASED GSN EDITOR USER MANUAL



Property	Value
Assumed	false
Id	Goal: SysSafe
Tobe Instantiated	false
To Be Supported	false

Figure 1. GSN editor main window, objects, and connections.



Property	Value
Id	
Literal	Aircraft Wheel Braking

Figure 2. Assurance case pattern in the GSN editor.

Figure C.8. Eclipse-based GSN editor manual part 1.

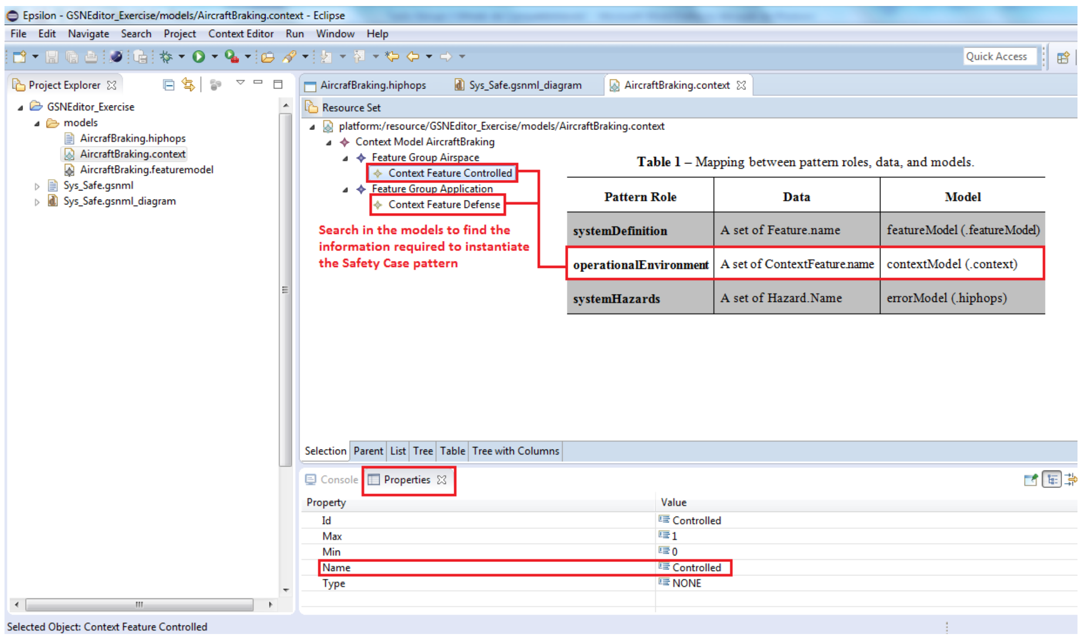


Table 1 – Mapping between pattern roles, data, and models.

Pattern Role	Data	Model
systemDefinition	A set of Feature.name	featureModel (.featureModel)
operationalEnvironment	A set of ContextFeature.name	contextModel (.context)
systemHazards	A set of Hazard.Name	errorModel (.hiphops)

Search in the models to find the information required to instantiate the Safety Case pattern

Property	Value
Id	Controlled
Max	1
Min	0
Name	Controlled
Type	NONE

Selected Object: Context Feature Controlled

Epsilon - GSNEditor_Exercise/models/AircraftBraking.context - Eclipse
Project Explorer
GSNEditor_Exercise
models
AircrafBraking.hiphops
AircraftBraking.context
AircraftBraking.featuremodel
Sys_Safe.gsnml
Sys_Safe.gsnml_diagram
Resource Set
platform:/resource/GSNEditor_Exercise/models/AircraftBraking.context
Context Model AircraftBraking
Feature Group Airspace
Context Feature Controlled
Feature Group Application
Context Feature Defense
Console
Properties

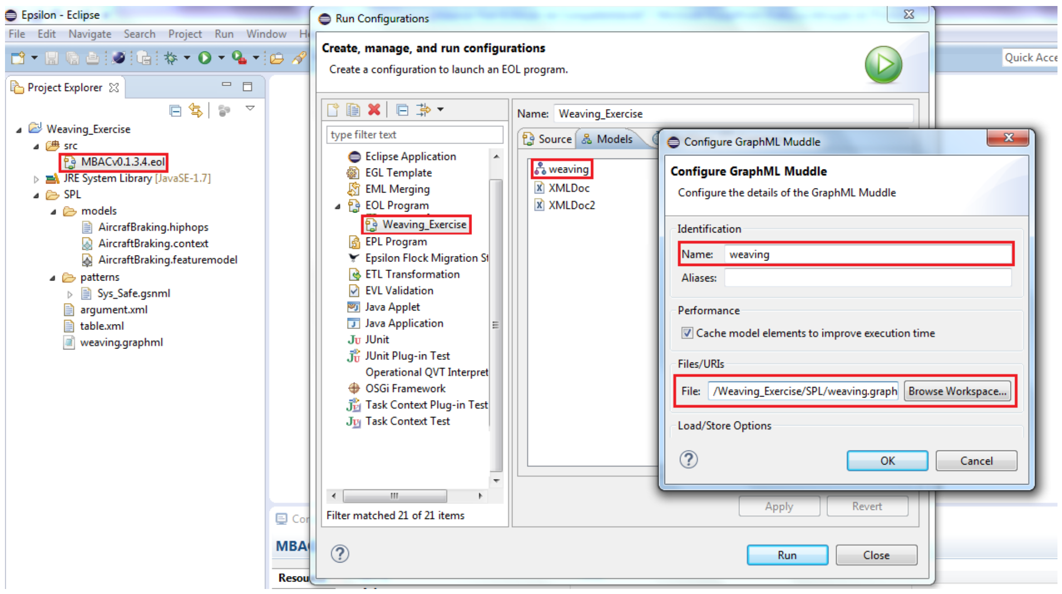
Figure 3. Mapping table and search in the system model.

Figure C.9. Eclipse-based GSN editor manual part 2.

MBAC TOOL USER MANUAL

Design and safety assessment models and their respective metamodels, the "*Hazard Avoidance"* assurance case pattern model specified in the GSN editor, and the weaving model are the inputs to configure the MBAC (HAWKINS et al. 2015) instantiation program. The "*argument.xml*" and "*table.xml*" are empty files used by the instantiation program output the results. Since the instantiation program uses variables to manipulate the input models to output the assurance case, its configuration requires performing the following steps:

1) Create a variable for referencing the weaving model: the variable should be named "weaving" and it refers to the weaving model file, i.e., a *".graphml"* extension file, as illustrated in Figure 1.



Run Configurations

Create, manage, and run configurations

Create a configuration to launch an EOL program.

Name: Weaving_Exercise

Configure GraphML Muddle

Configure the details of the GraphML Muddle

Name: weaving

Aliases:

[✓] Cache model elements to improve execution time

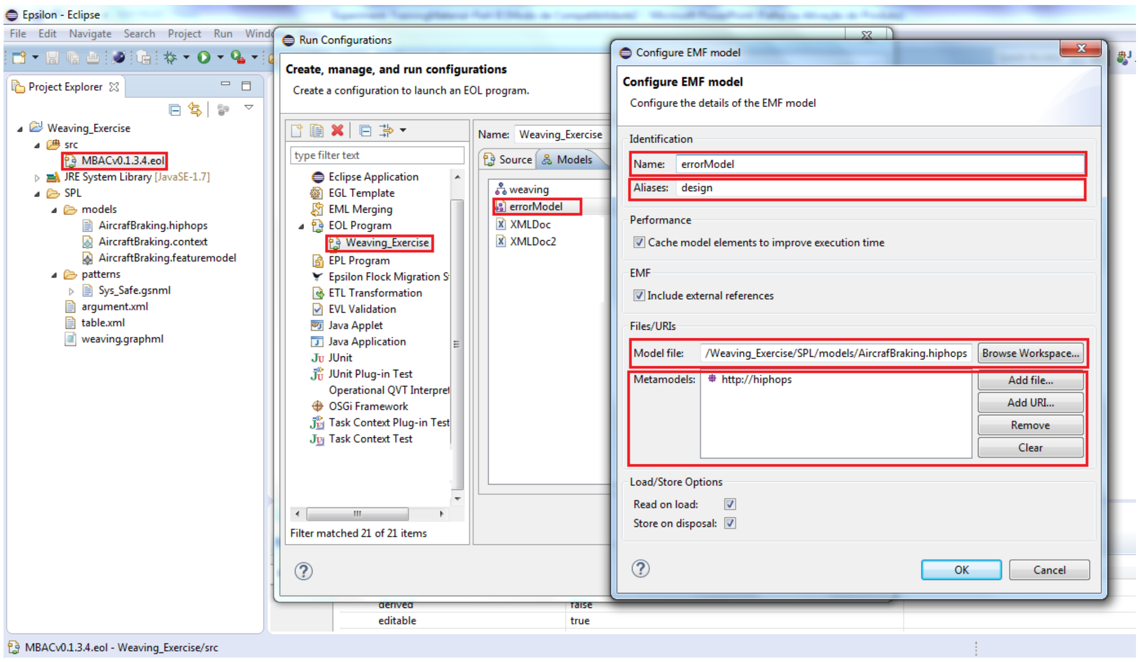
File: /Weaving_Exercise/SPL/weaving.graph

Filter matched 21 of 21 items

Figure 1. Referencing the weaving model in the instantiation program.

Figure C.10. MBAC tool manual part 1.

2) For each system model, create a variable linking the model and its respective metamodel. Figure 2 shows the variable *"errorModel"* referencing the *"AircraftBraking.hiphops"* EMF file, which conforms to the error metamodel. It is important to note that the alias "design" should be assigned to each one of these variables to allow the MBAC instantiation program manipulating the systems models to generate the assurance case; and



Configure EMF model

Configure EMF model

Configure the details of the EMF model

Identification

Name: errorModel

Aliases: design

Performance

[✓] Cache model elements to improve execution time

EMF

[✓] Include external references

Files/URIs

Model file: /Weaving_Exercise/SPL/models/AircrafBraking.hiphops

Metamodels: http://hiphops

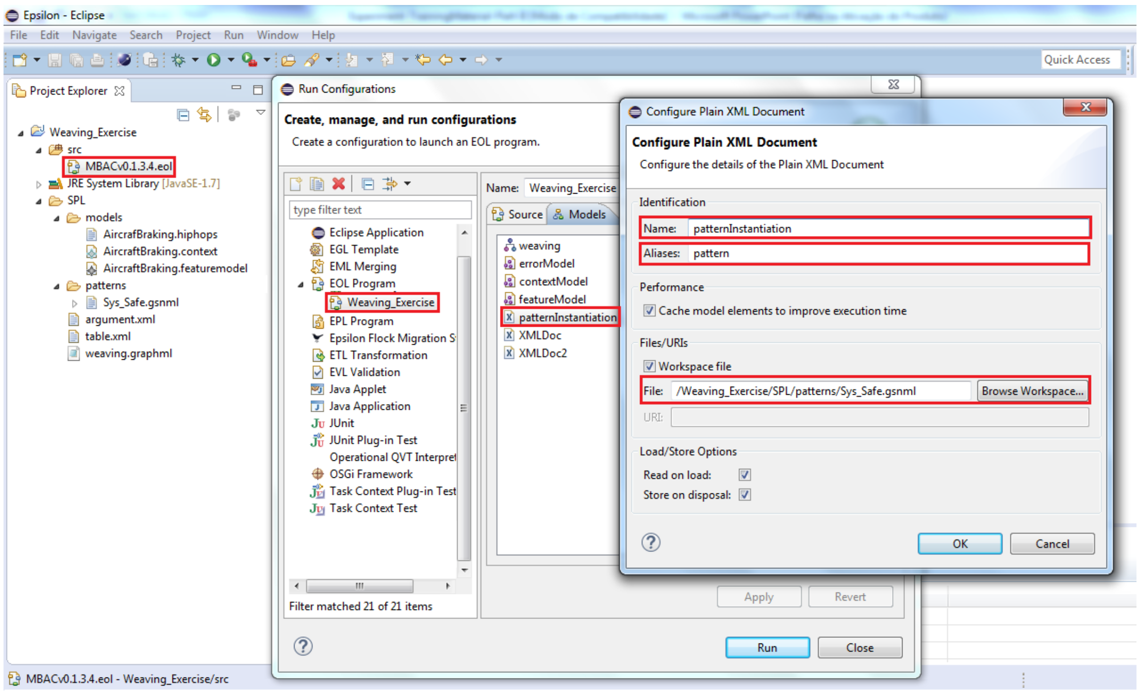
Load/Store Options

Read on load: [✓]

Store on disposal: [✓]

Figure 2. Referencing an artefact model in the instantiation program.

3) Create a variable to refer the assurance case pattern specified in the *".gsnml"* file. Figure 3 shows the variable *"patternInstantiation"* with the alias "pattern" referencing *"Sys*_Safe*.gsnml"* file.



Configure Plain XML Document

Configure Plain XML Document

Configure the details of the Plain XML Document

Identification

Name: patternInstantiation

Aliases: pattern

Performance

[✓] Cache model elements to improve execution time

Files/URIs

[✓] Workspace file

File: /Weaving_Exercise/SPL/patterns/Sys_Safe.gsnml

URI:

Load/Store Options

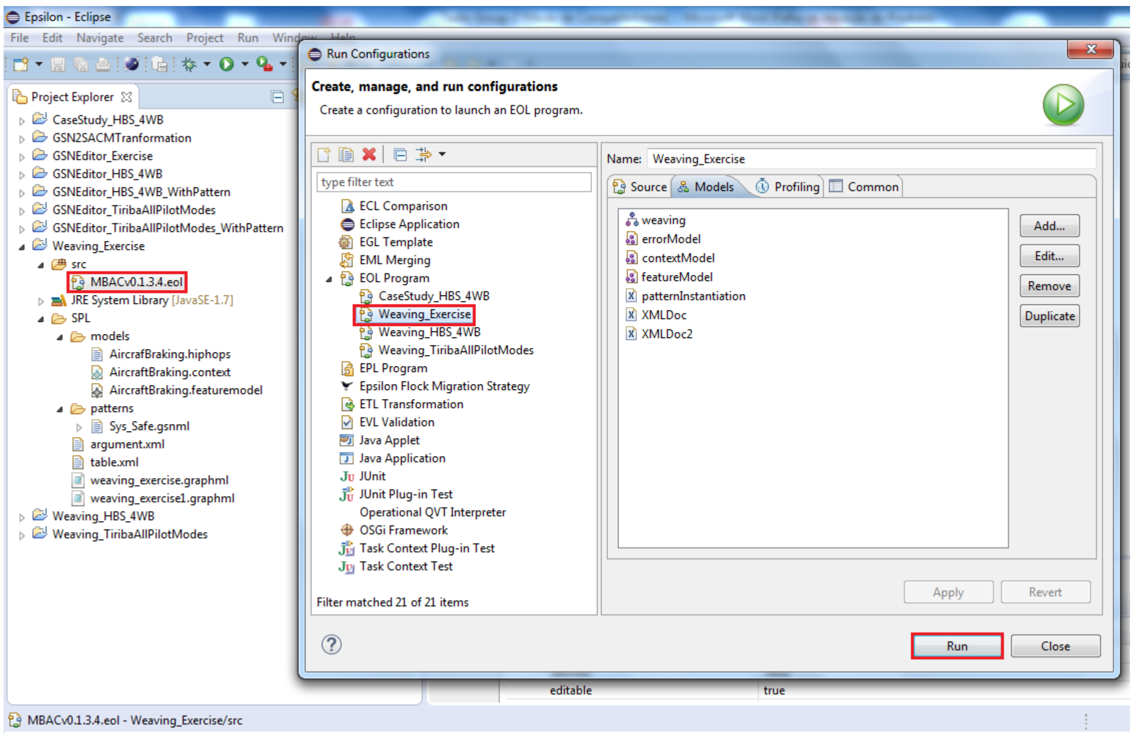
Read on load: [✓]

Store on disposal: [✓]

Figure 3. Referencing an assurance case pattern specification in the instantiation program.

Figure C.11. MBAC tool manual part 2.

4) After configuring the MBAC instantiation program with the input models for a given product line instance, the program is executed, as illustrated in Figure 4.



Epsilon - Eclipse

File Edit Navigate Search Project Run Window Help

Project Explorer

- CaseStudy_HBS_4WB
- GSN2SACMTranformation
- GSNEditor_Exercise
- GSNEditor_HBS_4WB
- GSNEditor_HBS_4WB_WithPattern
- GSNEditor_TiribaAllPilotModes
- GSNEditor_TiribaAllPilotModes_WithPattern
- Weaving_Exercise
  - src
    - MBACv0.1.3.4.eol
  - JRE System Library [JavaSE-1.7]
  - SPL
    - models
      - AircrafBraking.hiphops
      - AircraftBraking.context
      - AircraftBraking.featuremodel
    - patterns
      - Sys_Safe.gsnml
    - argument.xml
    - table.xml
    - weaving_exercise.graphml
    - weaving_exercise1.graphml
- Weaving_HBS_4WB
- Weaving_TiribaAllPilotModes

Run Configurations

Create, manage, and run configurations

Create a configuration to launch an EOL program.

type filter text

- ECL Comparison
- Eclipse Application
- EGL Template
- EML Merging
- EOL Program
  - CaseStudy_HBS_4WB
  - Weaving_Exercise
  - Weaving_HBS_4WB
  - Weaving_TiribaAllPilotModes
- EPL Program
- Epsilon Flock Migration Strategy
- ETL Transformation
- EVL Validation
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Operational QVT Interpreter
- OSGi Framework
- Task Context Plug-in Test
- Task Context Test

Filter matched 21 of 21 items

Name: Weaving_Exercise

Source | Models | Profiling | Common

- weaving
- errorModel
- contextModel
- featureModel
- patternInstantiation
- XMLDoc
- XMLDoc2

Add... Edit... Remove Duplicate

Apply Revert

Run Close

editable true

MBACv0.1.3.4.eol - Weaving_Exercise/src

Figure 4. Executing the instantiation program.

Figure C.12. MBAC tool manual part 3.

Post-Experiment Questionnaire – Part I

Participant ID: ___
Technique/Tool: GSN Editor

1) How easy was to use the GSN editor to instantiate assurance case patterns?
( ) Very easy
( ) Easy
( ) Average
( ) Not easy
( ) Very difficult

2) How easy was to search the required information to instantiate the assurance case pattern in different source models?
( ) Very easy
( ) Easy
( ) Average
( ) Not easy
( ) Very difficult

3) With regard to the **usability attribute**, how do you classify the user interface provided by GSN editor tool?
( ) Highly usable
( ) Usable
( ) Average
( ) Low usable
( ) Not usable

4) With regard to the **usability attribute**, how do you classify the user interface provided by Eclipse-based model editors (feature, context, error, architecture models)?
( ) Highly usable
( ) Usable
( ) Average
( ) Low usable
( ) Not usable

Figure C.13. Post-experiment questionnaire part I.

Post-Experiment Questionnaire – Part II

Technique/Tool: Model-Based Assurance Cases (MBAC)

1) How easy was to understand how MBAC tool works and to use it to instantiate assurance case patterns?

( ) Very easy

( ) Easy

( ) Average

( ) Not easy

( ) Very difficult

2) How easy was to configure MBAC tool with the input models and execute it to generate an assurance case for a given product line instance?

( ) Very easy

( ) Easy

( ) Average

( ) Not easy

( ) Very difficult

3) With regard to the **usability attribute**, how do you classify the user interface provided by MBAC tool?

( ) Highly usable

( ) Usable

( ) Average

( ) Low usable

( ) Not usable

Figure C.14. Post-experiment questionnaire part II.

Appendix D

TIRIBA FLIGHT CONTROL CASE STUDY OUTCOMES

This document details the management, development, safety assessment, and assurance case deliverables produced during the Tiriba Flight Control Product Line (TiribaFC-SPL) case study. Section D.1 presents the TiribaFC-SPL feature and context models. Section D.2 shows the TiribaFC-SPL architecture model. Section D.3 presents the functional hazard assessment and component failure analysis using HiP-HOPS compositional safety analysis technique and tool. Section D.4 presents the mapping links between functional and context features and their realization in architecture and safety analysis models in the variability realization model created with the support of BVR toolset and its adapters developed in this thesis. Section D.5 shows the variability resolution and product derivation processes in TiribaFC-SPL. Section D.6 illustrates the fault trees and FMEA results generated for four Tiriba flight control system variants with the support of HiP-HOPS compositional safety analysis tool. Section D.8 shows the results of applying HiP-HOPS Tabu Search design optimization technique to support the automatic decomposition of safety integrity requirements in each TiribaFC-SPL system variant. Section D.9 presents the results applying the Product Line component SIL Decomposition (PL-SILDec) method and tool (OLIVEIRA *et al.* 2015) proposed in this thesis to support the automatic analysis and decomposition of safety integrity requirements to TiribaFC-SPL components. Finally, Section D.10 shows the generated variant-specific assurance cases with the support of Model-Based Assurance Cases tool.

D.1 SC-PLE Phase 1: TiribaFC-SPL Requirements Elicitation

The Tiriba Flight Control System (TiribaFC) is part of Tiriba UAV system, which comprises four subsystems: Navigation, Control, Inertial, and Barometric Units, where each unit is assigned to a dedicated microprocessor (microcontrollers PIC32 80 MHz). The "*Navigation*" unit subsystem guides the aircraft along a route, according to the rules defined by the mission planner, e.g., the accomplishment of tasks associated with each waypoint such as taking pictures from a specific GPS location. The "*Control*" unit subsystem, i.e., the flight control system, is intended to start the flight mode (direct, stabilized, and autonomous), processing and setup flight commands, keeping the flight conditions and executing commands sent by the "*Navigation*" subsystem. The "*Inertial*" unit subsystem estimates the real-time aircraft position based on the information received from inertial sensors and Global Navigation Satellite System. Finally, the "*Pressure*" unit subsystem provides altitude, vertical speed, and aero-dynamical speed based on air pressure (BRANCO *et al.* 2011). The TiribaFC system was developed in compliance with ARP 4754A (EUROCAE, 2010) and DO-178C (RTCA, 2012) aerospace standards.

From the analysis of TiribaFC system MATLAB/Simulink model, common and variable flight control system functions were identified by adopting an extractive product line development strategy. Figure D.1 shows the TiribaFC feature model that comprises the "*Manual Pilot*" mandatory feature, and three optional pilot mode features. The "*Manual Pilot*" core feature consists in a human operator sending commands to the unmanned aircraft from a ground control station. The "*Autopilot*" feature executes a pre-defined route. The "*Assisted Mode*" feature allows the operator sending commands to the UAV configured with "*Autopilot*" feature. The "*Autonomous Pilot*" mode feature allows the UAV to perform actions according to its current environmental conditions captured by pressure sensors. "*Assisted Pilot*", "*Autonomous Pilot*", and "*Autopilot*" features can be combined into several different ways, allowing the derivation of seven different flight control system variants. These variants are specified in the "*Mode*" feature group from the TiribaFC-SPL context model shown in Figure D.2. These variants differ from each other in the number of pilot modes that provide the flight control capability and the way as they are combined. For example, "*Manual and Autopilot*" variant differs from "*Manual*", *"Autopilot",* and "*Autonomous*" variant by the absence of autonomous pilot mode.

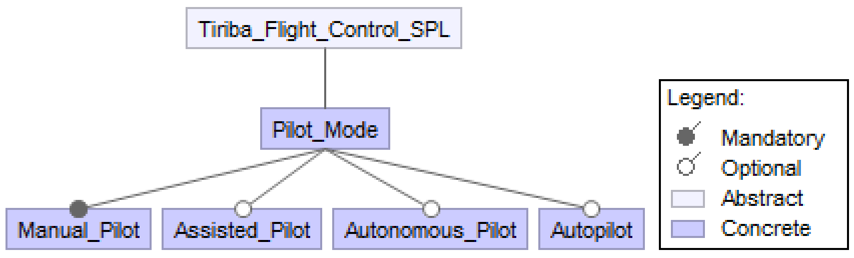


Figure D1. TiribaFC-SPL feature model.

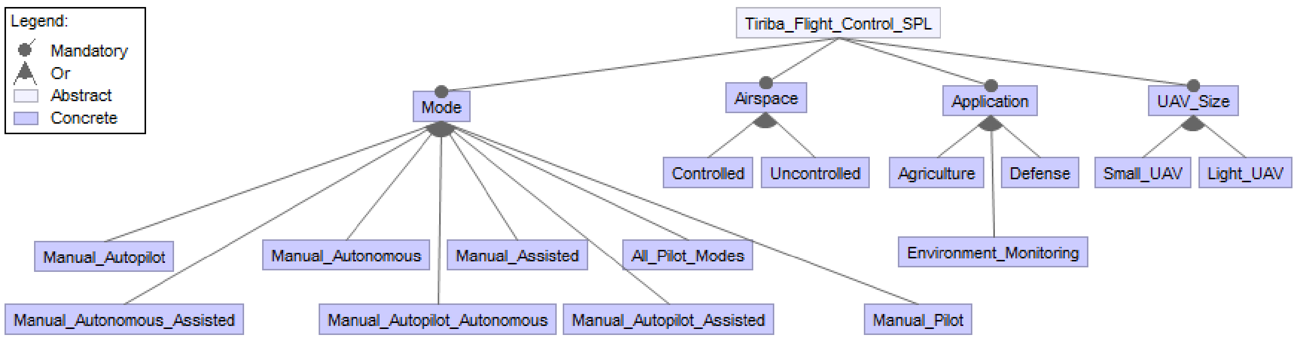


Figure D.2. TiribaFC-SPL context model.

Tiriba flight control variants can operate in a range of different contexts determined by combinations between "*Airspace*", "*Application*", and "*UAV Size*" contextual elements. A Tiriba flight control product variant can operate in controlled or uncontrolled "*Airspace*", it can be deployed into a '*Light*" or a "*Small*" UAV, and it can be used in "*Agriculture*", "*Environment Monitoring*", or military "*Defense*" applications. Therefore, the composition of pilot mode and other contextual variants lead to 84 Tiriba flight control product variants. Pilot *mode* variants together with "*Application*", "*UAV Size*", and "*Airspace*" contextual features directly impact in the definition of safety properties, changing hazards, their causes, and allocated safety requirements. For example, different safety requirements can be allocated to mitigate the effects of a given hazard associated with "*Manual and Autopilot*" variant assumed to operate in controlled or uncontrolled airspaces. Therefore, context and design variation is considered in performing the TiribaFC-SPL safety analysis.

D.2 SC-PLE Phase 2: TiribaFC-SPL Architectural Design

Figure D.3 illustrates an excerpt of the TiribaFC-SPL architecture model in a SysML block diagram. TiribaFC-SPL mainly comprises 4 subsystems, and 14 components. These subsystems and components are composed by 252 model blocks and subcomponents. The "*Switch block*" variability mechanism (BOTTERWECK *et al.* 2010) was used for implementing the pilot mode variation point in the original Tiriba flight control system model.

The "*Mode Switch*" block encapsulates the pilot mode variation point, whilst the "*Command Switch*" block encapsulates the variation point inherent to the source of the pilot commands, e.g., autopilot, manual pilot subsystems, sent to the Tiriba UAV. The "*PWM*[23] *Decoder*" component output port connected to the "*Command Switch*" block is the realization of the "*Manual Pilot*" feature. The "*Autopilot*" subsystem, the "*Flight Controls*" output port from the "*PWM Decoder*" component connected to: the "*Pilot Commands*" input port from the "*Flight Stabilizer*" subsystem, and the "*Command Switch*" block, and the *Flight Stabilizer* component output port (i.e., *Autopilot Settings*) connected to the "*Assisted Mode Switch*" block represent the realization of the "*Autopilot*" feature. The "*Basic Command Processor*" subsystem output port connected to the "*Assisted Mode Switch*' block represents the realization of the "*Autonomous Pilot*" feature. Finally, the "*Control Mode*" output port from the "*Mode Switcher*" component connected to both "*Assisted Mode Switch*" and "*Command Switch*" blocks represents the realization of the "*Assisted Pilot*" feature.

The "*Inertial Measure Unit*" (IMU), "*Pressure*" sensors, and the "*Navigation*" system provide input data for "*Basic Command Processor*" and "*Flight Stabilizer*" sub systems. The "*Altitude Sensor*" also provides input data for the "*Flight Stabilizer*" subsystem. The "*Pilot Joystick*" component provides input data for the "*PWM Decoder*" component, which represents the manual pilot mode. The "*Command Switch*" is a decision block that receives flight commands from different pilot modes and it sends them to the "*PWM Encoder*" component. The "*PWM Encoder*" component performs the processing and outputs the flight commands that are further decoded and executed in flight.

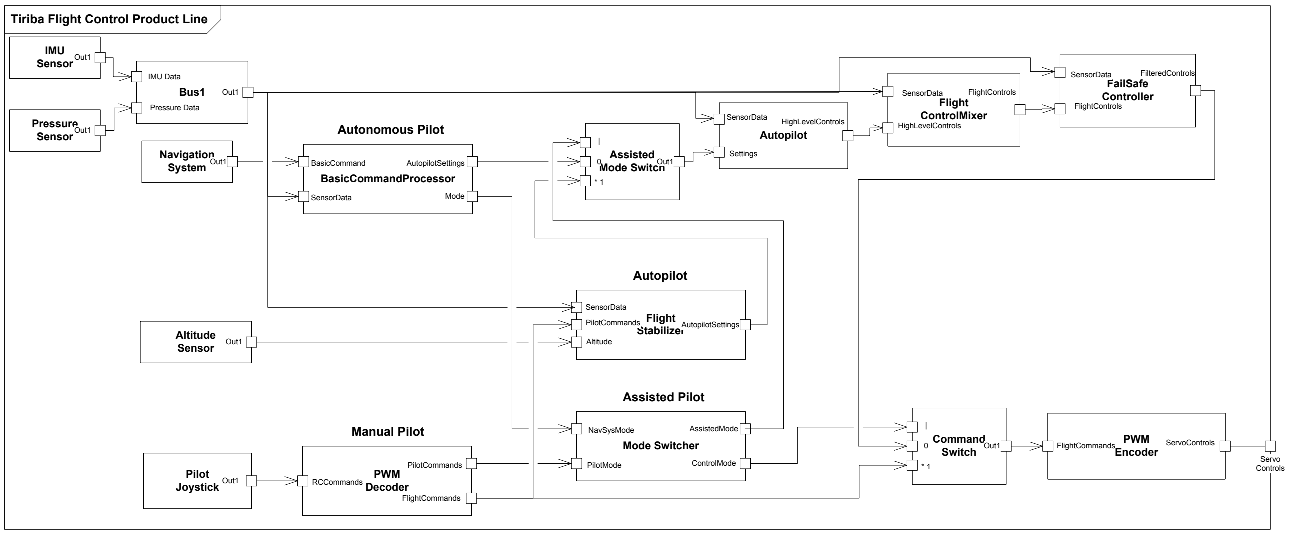


Figure D.3 Tiriba flight control product line architecture model.

[23]Pulse Width Modulation

D.3 SC-PLE Phase 3: TiribaFC-SPL Safety Analysis

In an ARP 4761 Functional Failure Assessment fashion, product line functional hazard assessment and component failure analysis were performed by considering the following four Tiriba flight control variants: "*Manual and Autonomous Pilot*" (TFC-MAT), "*Manual and Assisted Pilot*" (TFC-MAS), "*Manual and Autopilot*" (TFC-MAP), and "*All Pilot Modes*" (TFC-ALL). TFC-MAT and TFC-MAS variants were assumed in the following usage context: deployment in a "*Light UAV*", to operate in an "*Uncontrolled Airspace*", and to be used in "*Environment Monitoring*" applications. TFC-MAP and TFC-ALL variants were assumed to be deployed in a "*Light UAV*", to operate in a "*Controlled Airspace*", and to be used in military "*Defense*" applications. The safety analysis was constrained to these variants and usage context to reduce its complexity, since performing such analysis by considering all potential Tiriba flight control variants would be prohibitive. These variants were analyzed from the perspective of the SAE ARP 4754A risk assessment process. The extended HAZOP analysis technique supported by HiP-HOPS compositional safety analysis tool integrated to MATLAB/Simulink was used to support the TiribaFC-SPL safety analysis phase.

Firstly, by considering each system variant and its associated context, functional hazard assessment started by identifying interactions between product line architectural components that may lead to hazardous events. From the analysis of these interactions, hazards and their potential causes have been identified for each flight control system variant. In the following, during risk assessment, these hazards were classified with regard to their severity and probability of occurrence, and safety integrity requirements, in terms of DALs, were allocated to mitigate the hazard effects. Table D.1 shows the identified hazards, their potential causes, and allocated DALs associated with each product variant. The "*No pilot commands*" and "*Value pilot commands*" hazards were identified from the analysis of these variants. It has been considered that the "*No pilot commands*" is being produced in a Tiriba flight control variant whenever both pilot modes, e.g., manual pilot and autopilot, are omitting their outputs. An incorrect value for the pilot commands happens when at least one of the pilot modes provide wrong flight commands, e.g., wrong coordinates.

Table D.1 – Variability in TiribaFC-SPL functional hazard assessment.

Variant	*Usage Context*	*Hazard ID*	*Failure Conditions*	*DAL*
TFC-MAT	Env. Monitoring, Light UAV, Uncontrolled airspace	MAT_No_pilot_commands	Omission of FailSafeController.FilteredControls AND PWMDecoder.FlightCommands outputs.	A(4)
		MAT_Value_pilot_commands	Incorrect Value of FailSafeController. FilteredControls AND PWMDecoder. FlightCommands outputs.	C (2)
TFC-MAS	Env. Monitoring, Light UAV, Uncontrolled airspace	MAS_No_pilot_commands	Omission of PWMDecoder.FlightCommands AND ModeSwitcher.ControlMode AND Fail SafeController.FilteredControls outputs.	A(4)
		MAS_Value_pilot_commands	Incorrect Value of PWMDecoder. FlightCommands AND ModeSwitcher. ControlMode AND FailSafeController. FilteredControls outputs.	B (3)
TFC-MAP	Defense, Light UAV, Controlled airspace	MAP_No_pilot_commands	Omission of FailSafeController.FilteredControls AND PWMDecoder.FlightCommands outputs.	A (4)
		MAP_Value_pilot_commands	Incorrect Value of FailSafeController. FilteredControls AND PWMDecoder. FlightCommands outputs.	A (4)
TFC-ALL	Defense, Light UAV, Controlled airspace	ALL_No_pilot_commands	Omission of PWMDecoder.Flight Commands AND ModeSwitcher.Control Mode AND FailSafeController.FilteredControls outputs.	B(3)
		ALL_Value_pilot_commands	Incorrect Value of PWMDecoder.FlightCommands AND ModeSwitcher.Control Mode AND FailSafeController.FilteredControls outputs.	B(3)

Hazard causes and allocated safety integrity requirements may change according to the TiribaFC-SPL product variant. For example, the causes for the "*No pilot commands*" hazard in TFC-ALL variant is the omission of flight command outputs provided by "*PWM Decoder"* (i.e., manual pilot), "*Mode Switcher"* (i.e., assisted pilot), and "*Fail Safe Controller'* (i.e., autonomous or autopilot) components. On the other hand, omission failures in autonomous and manual pilot flight commands are the causes for this hazard in the TFC-MAT variant. The causes for "*Value pilot commands*" in the TFC-ALL variant are different from the causes for this hazard in TFC-MAT and TFC-MAP variants, as shown in Table D.1. With regard to variation in the allocated safety integrity requirements, DALs from different stringencies were assigned to the identified hazards in different flight control system variants. For example, DAL "A", i.e., the most stringent, was allocated to the "*No pilot commands*" hazard in the

TFC-MAT, TFC-MAS, and TFC-MAP variants, and DAL "B", less stringent, was assigned to this hazard in the TFC-ALL variant. A less stringent DAL "C" was assigned to the "*Value pilot commands*" in the TFC-MAT variant, whereas most stringent DALs "A", "A", and "B" were respectively assigned to mitigate this hazard in the TFC-MAS, TFC-MAP, and TFC-ALL system variants.

Whereas different component failures may contribute to the occurrence of system hazards in different TiribaFC-SPL variants, from analysis of the identified hazards in different variants and context, component failure analysis was carried out. From such analysis, 106 failure logic expressions were added to 47 components and model blocks. Table D.2 illustrates the failure logic for three components by considering the TFC-MAT, TFC-MAS, and TFC-ALL variants and their associated usage context. The "*Usage Scenario/Current*" column denotes the variant/usage context failure logic implementation and whether it is assumed as the current one for a given component. This table presents two failure logic expressions associated with "*Basic Command Processor"* component addressing TFC-MAT and TFC-ALL variants and their respective usage context. TFC-MAT failure logic states that one omission failure in the "*AutopilotSettings*" output port from the "*Basic Command Processor"* component caused by an internal failure or omissions in one of its input ports, contributes to the occurrence of hazards. On the other hand, the TFC-ALL failure logic states that two omission failures in "*Basic Command Processor"* component outputs caused by different failures in component inputs contribute to the occurrence of hazards. Therefore, the causes of an omission failure in the "*AutopilotSettings*" output port from the "*Basic Command Processor"* component are different in TFC-MAT and TFC-ALL variants, as shown in the *"Failure Expression"* column from Table D.2. When variability in the safety model is resolved for the TFC-ALL variant, the TFC-ALL failure logic is set as the current one in the product failure model. At the end of this phase, the TiribaFC-SPL failure model, in the form of HiP-HOPS failure annotations in the Tiriba Simulink model, is delivered.

Variation in the DALs allocated to Tiriba flight control hazards is further propagated throughout the DALs allocated to mitigate the effects of contributing failure modes of components. Additionally, different component failure data lead to different fault propagations (fault trees), i.e., different combinations of component failures leading to the occurrence of hazards in different TiribaFC-SPL product variants. Therefore, variation identified in functional hazard assessment and component failure analysis is then, propagated throughout Fault Trees and FMEA results. Thus, different TiribaFC-SPL variants have fault trees with different fault propagations and FMEA results. In order to support the systematic the reuse of TiribaFC-SPL safety analysis data specified in the failure model, the impact of context and design variation into safety properties is managed in the variability realization modeling phase.

Table D.2 – Variability in TiribaFC-SPL component failure analysis.

Component	*Impl/Current*	*Output Deviation*	*Failure Expression*
Basic Command Processor	TFC-MAT + Env. Monitoring, Light UAV, Uncontrolled airspace/No	Omission-AutopilotSettings	OFailure1 *or* (Omission-BasicCommand *or* Omission-SensorData)
	TFC-ALL + Defense, Light UAV, Controlled airspace/Yes	Omission-AutopilotSettings	OFailure1 *or* (Omission-BasicCommand *and* Omission-SensorData)
		Omission-Mode	OFailure2 *or* (Omission-BasicCommand *and* Omission-SensorData)
Mode Switcher	TFC-MAT + Env. Monitoring, Light UAV, Uncontrolled airspace/No	Omission-ControlMode	OFailure1 *or* Omission-PilotMode
		Value-ControlMode	VFailure1 *or* Value-PilotMode
	TFC-ALL + Defense, Light UAV, Controlled airspace/Yes	Omission-ControlMode	OFailure1 *or* Omission-PilotMode
		Value-ControlMode	VFailure1 *or* Value-PilotMode
		Omission-AssistedMode	OFailure2 *or* Omission-NavSysModde
		Value-AssistedMode	VFailure2 *or* Value-NavSysMode
PWM Decoder	TFC-MAT + Env. Monitoring, Light UAV, Uncontrolled airspace/No	Omission-FlightCommands	OFailure1 *or* Omission-RCCommands
		Value-FlightCommands	VFailure1 *or* (Omission-RCCommands *or* Value-RCCommands)
	TFC-MAS + Env. Monitoring, Light UAV, Uncontrolled airspace/No	Omission-FlightCommands	OFailure1 *or* Omission-RCCommands
		Value-FlightCommands	VFailure1 *or* (Omission-RCCommands *or* Value-RCCommands)
		Omission-PilotCommands	OFailure2 *or* Omission-RCCommands
		Value-PilotCommands	VFailure2 *or* (Omission-RCCommands *or* Value-RCCommands)

D.4 SC-PLE Phase 4: TiribaFC-SPL Variability Realization Modeling

The BVR toolset (VASILEVISKIY *et al.* 2015), and BVR adapters for MATLAB/Simulink and HiP-HOPS developed in the course of this thesis (see Chapter 4, Section 4.3.1) were used to support the specification of the variability realization model for the TiribaFC-SPL. In this phase, mapping links between functional and usage context features and their realization in MATLAB/Simulink architectural model components, and failure model elements were defined. From the analysis of the product line feature and context models, the following variation points were identified: "*Pilot Mode*", a functional variation point that defines different ways in which pilot mode functions can be combined to derive a product variant; "*Mode*" defines the Tiriba flight controls variants that impact the safety properties, and "*Airspace", "Application",* and "*UAV Size*" contextual variation points, encapsulate the variation in the usage context where "*Pilot Mode*" functional variants can operate. In this case study, mapping links between features and their realization in architectural and safety analysis were specified by considering the TFC-MAT, TFC-MAS, TFC-MAP, and TFC-ALL variants in their respective usage context, as described in Section D.3.

The definition of the variability realization model was performed in two steps: the variability specification model was defined based on the TiribaFC-SPL feature and context models, followed by the definition of the variability realization model, where features representing functional and contextual variants were linked to placement and replacement fragments referencing TiribaFC-SPL architecture and failure model elements, via fragment substitutions. Fragment substitutions define architectural and failure model elements to be included and excluded when a particular system variant is resolved during product derivation. Details about how to specify placement, replacement, and fragment substitutions into BVR variability realization model can be found elsewhere (VASILEVSKYI *et al.* 2015).

Fragment substitutions were created for each functional and safety-related variant specified in the TiribaFC-SPL variability specification model (feature model). Table D.3 shows the placement, replacement, and fragment substitutions associated with three TiribaFC-SPL variants. The placement fragment associated with each flight control system variant, represents the variability mechanisms in the TiribaFC-SPL architecture model, i.e., model elements that may change according to different product variants. The-

Table D.3 – Pilot mode functional variants and associated fragments.

Variation Point	***Variant***	***Fragment***	***Architectural Model Elements***
Pilot Mode	*Manual_Autopilot*	Placement	Basic Command Processor, Mode Switcher components, their ports and connections.
		Replacement	PWM Decoder, Mode Switcher, FlightStabilizer, Autopilot, FlightControlMixer, FailSafeController, Command Switch, and PWM Encoder components, their ports and connections.
	Manual_Autonomous_Pilot	Placement	Basic Command Processor, Mode Switcher components, their ports and connections.
		Replacement	Basic Command Processor, Mode Switcher, PWM Decoder, Mode Switcher, FlightStabilizer, Autopilot, FlightControl Mixer, FailSafeController, Command Switch, and PWM Encoder components, their ports and connections.
	Manual_Assisted_Pilot	Placement	Basic Command Processor, Mode Switcher components, their ports and connections.
		Replacement	Basic Command Processor, Mode Switcher, PWM Decoder, Mode Switcher, FlightStabilizer, Autopilot, FlightControlMixer, FailSafeController, Command Switch, and PWM Encoder components, their ports and connections.

refore, "*Basic Command Processor*" and "*Mode Switcher*" components, their ports and connections are variability mechanisms in the TiribaFC-SPL architecture model, which should be removed when variability is resolved for a given system variant. The replacement fragment associated with "*Manual_Autopilot*" variant indicates that "*PWM Decoder*", "*Mode Switcher*", "*FlightStabilizer*", "*Autopilot*", "*FlightControlMixer*", "*FailSafeController*", "*CommandSwitch*", and "*PWM Encoder*" components, their ports and connections should be included in the product model when variability in the architecture model is resolved. On the other hand, "*Basic Command Processor*" subsystem, their connections, "*NavSysMode*" and "*AssistedMode*" ports from "*Mode Switcher*" component should be removed from the final product when architectural variability is resolved for the "*Manual_Autopilot*" variant, as shown in Table D.3.

Fragment substitutions were also specified to define how variability in the TiribaFC-SPL failure model is resolved when "*MAT_Env_Light_Uncontrolled*", "*MAS_Env_Light_Uncontrolled*", "*MAP_Defense_Light_Controlled*", and "*ALL_Defen se_Light_Controlled*" safety-related variants are chosen in the resolution model. Table

D.4 shows an excerpt of the mapping links between safety-related variants and their realization in the failure model. When "*MAT_Env_Light_Uncontrolled*" variant is chosen, functional hazard assessment and component failure data associated with "*MAS_Env_Light_Uncontrolled*", "*MAP_Defense_Light_Controlled*", and "*ALL_Defense_Light_Controlled*" variants specified as variability mechanisms in a placement fragment, should be removed. In the following, variant-specific failure data specified in the replacement fragment should be included in the resolved failure model. Placement and replacement fragments for "*MAS_Env_Light_Uncontrolled*" and "*MAP_Defense_ Light_Controlled*" variants show the failure model elements to be included and excluded when variability in the failure model is resolved for these variants.

Table 8.4 – Pilot Mode/usage context safety-related variants and associated fragments.

Variation Point	***Variant***	***Fragment***	***Failure Model Elements***	
			FHA Data	***Failure Data***
Mode/Application, Airspace, and UAV Size	*MAT_Env_ Light_ Uncontrolled*	Placement	MAS_No_pilot_commands, MAS_Value_pilot_commands, MAP_No_pilot_commands, MAP_Value_pilot_commands, ALL_No_pilot_commands, ALL_Value_pilot_commands	TFC-ALL-BCPImpl, TFC-ALL-MSImpl, TFC-MAS-PWDImpl, …
		Replacement	MAT_No_pilot_commands, MAT_Value_pilot_commands	TFC-MAT-BCPImpl, TFC-MAT-MSImpl, TFC-MAT-PWDImpl, …
	MAS_Env_ Light_ Uncontrolled	Placement	MAT_No_pilot_commands, MAT_Value_pilot_commands, MAP_No_pilot_commands, MAP_Value_pilot_commands, ALL_No_pilot_commands, ALL_Value_pilot_commands	TFC-MAT-BCPImpl, TFC-MAT-MSImpl, TFC-ALL-PWDImpl, …
		Replacement	MAS_No_pilot_commands, MAS_Value_pilot_commands	TFC-ALL-BCPImpl, TFC-ALL-MSImpl, TFC-MAS-PWDImpl, …
	MAP_Env_ Light_ Controlled	Placement	MAT_No_pilot_commands, MAT_Value_pilot_commands, MAS_No_pilot_commands, MAS_Value_pilot_commands, ALL_No_pilot_commands, ALL_Value_pilot_commands	TFC-MAT-BCPImpl, TFC-ALL-MSImpl, TFC-MAS-PWDImpl, …
		Replacement	MAP_No_pilot_commands, MAP_Value_pilot_commands	TFC-ALL-BCPImpl, TFC-MAT-MSImpl, TFC-ALL-PWDImpl, …

D.5 SC-PLE Phases 5 and 6: Resolution Modeling and Product Derivation

The delivered TiribaFC-SPL variability realization model comprises 8 fragment substitutions, 2 placements and 8 replacements fragments. 4 fragments substitutions are associated with variability in architectural model elements, and 4 fragment substitutions are associated with variability in failure model elements.

In application engineering, resolution models were created for four Tiriba flight control system variants with the support of the BVR resolution model editor. For this case study, resolution models were defined for: TFC-MAT, TFC-MAS, TFC-MAP, and TFC-ALL variants as shown in Figure D.5. These variants differ from each other in the number of pilot mode subsystems and components, functional hazard assessment and component failure data. The TFC-MAT (Figure D.5a) variant comprises manual and autonomous pilot features, and TFC-MAS (Figure D.5b) variant comprises manual pilot and assisted pilot features. These variants are considered to be deployed in a light UAV, to be used in environment monitoring applications, and operating in an uncontrolled airspace. The TFC-MAP (Figure D.5c) variant comprises manual and assisted pilot features, and TFC-ALL (Figure D.5d) variant comprises

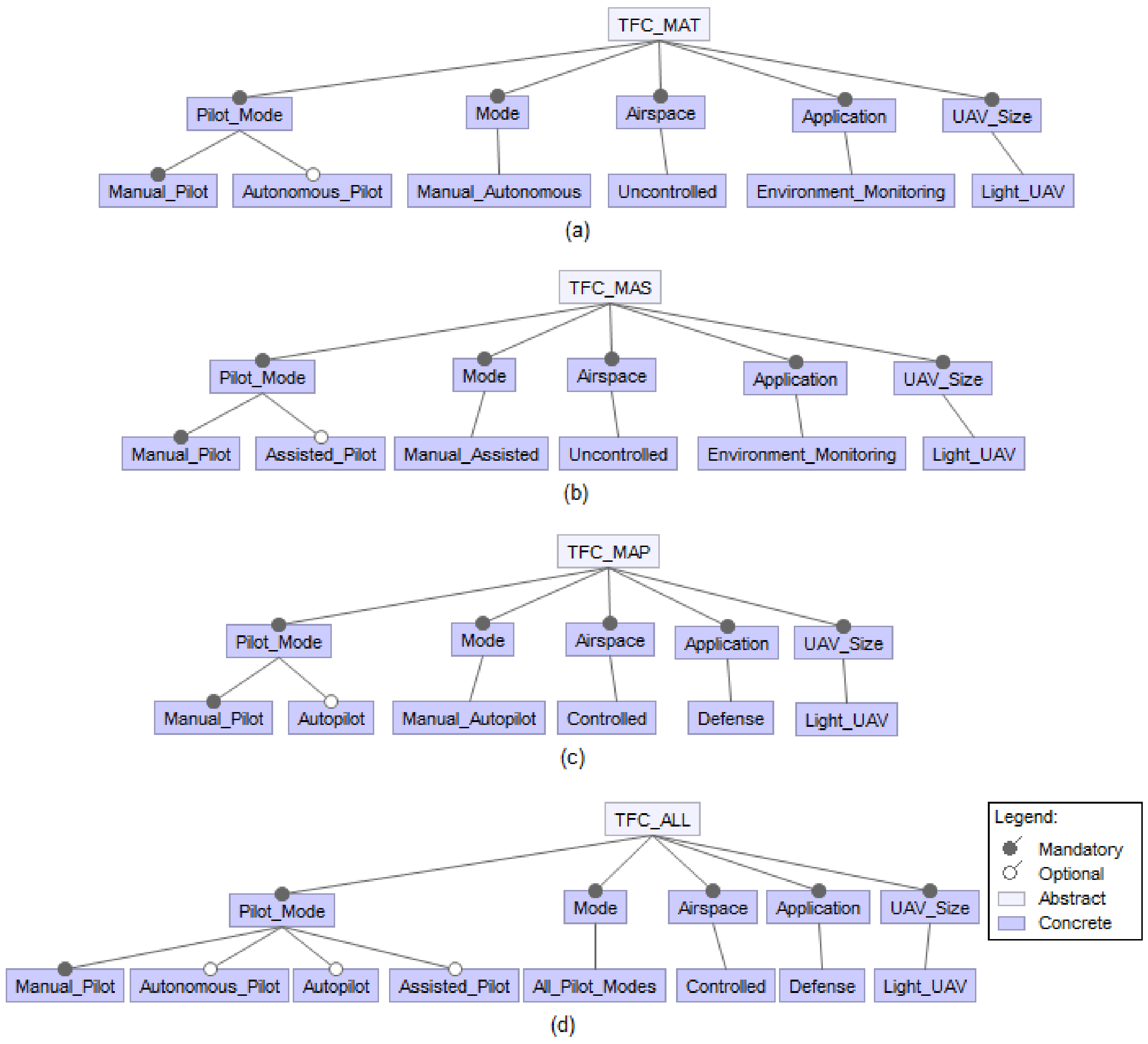


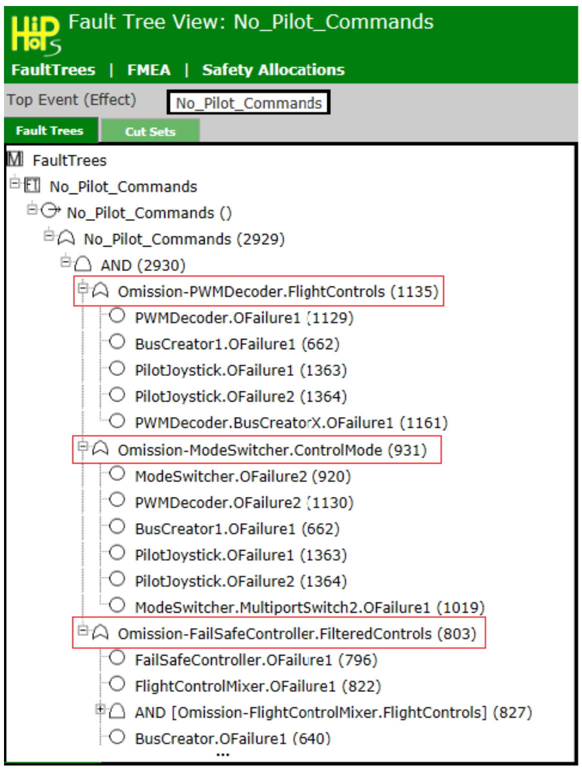
Figure D.5. TiribaFC-SPL resolution models.

manual, assisted, autonomous, and autopilot features. TFC-MAP and TFC-ALL systems variants are considered to be deployed in a light UAV, to be used in military defense applications operating in a controlled airspace.

For each system variant shown in Figure D.5, the following models were input to the BVR variant management tool (VASILEVSKYI *et al.* 2015) resolving variability in architectural and safety models during "*Product Derivation*" phase: BVR variability specification, resolution, and realization models, and the TiribaFC-SPL MATLAB/Simulink architecture and HiP-HOPS failure models. Thus, variant-specific MATLAB/Simulink architecture models with HiP-HOPS failure annotations were generated. The TFC-ALL variant-specific architecture model comprises all pilot mode subsystems and components shown in Figure D.3. The TFC-MAT architecture differs from TFC-ALL by the absence of autopilot and assisted pilot components. The TFC-MAS architecture comprises assisted pilot component and it does not contains autonomous and autopilot components. The TFC-MAP architecture comprises autopilot mode components. Tiriba flight control variant-specific models also comprise variant-specific safety analysis data. These models were further input to HiP-HOPS (PAPADOPOULOS *et al.* 2011) compositional safety analysis tool performing the automatic synthesis of fault trees, FMEA results, and DAL decomposition for each Tiriba flight control system variant.

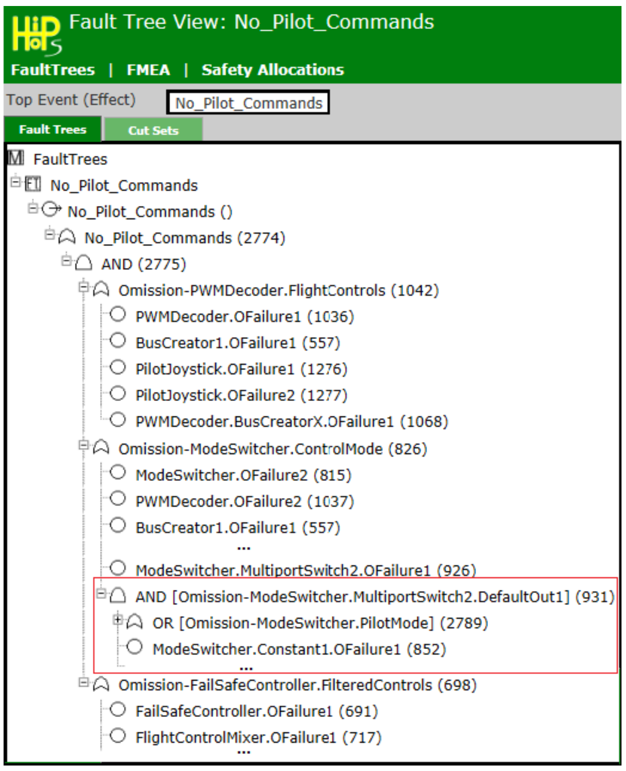
D.6 SC-PLE Phase 7: Variant-Specific Fault Tree Analysis and FMEA

This section presents the fault trees and FMEA results generated for each TiribaFC-SPL variant with the support of HiP-HOPS compositional safety analysis technique/tool. Fault trees were generated for each one of the 8 variant-specific hazards shown in Table D.1, and variant-specific FMEA results were synthesized from the fault trees. Figure D.6 shows excerpts of the "*No pilot commands*" fault trees generated for each variant. The analysis of these fault trees shows the impact of TiribaFC-SPL variation on hazard causes. Therefore, failure conditions that directly contribute to the occurrence of "*No pilot commands*" hazard in TFC-ALL and TFC-MAS fault trees (Figures D.6a and D.6b) are omission of flight commands in *PWM Decoder*, *Mode Switcher*, and *FailSafeController* output ports. On the other hand, the causes for this hazard in TFC-MAT and TFC-MAP fault trees (Figures D.6c and D.6d) are omission of flight commands in "*PWM Decoder"* and "*Fail Safe Controller"* output ports. Such variation is further propagated throughout component failure modes that indirectly contribute to the occurrence of hazards. For example, different component failures contribute to cause the omission of “*HighLevelControls*” output port from the “*Autopilot*” component in TFC-



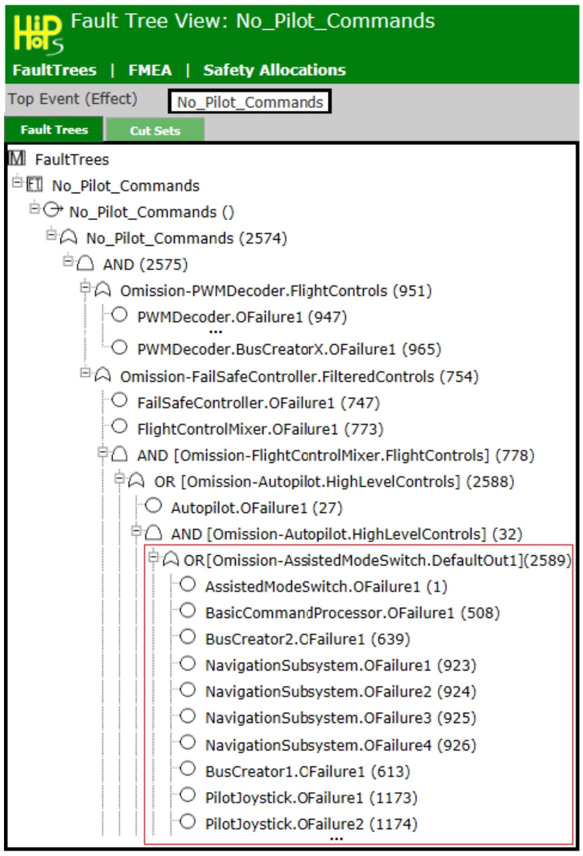
```
Fault Tree View: No_Pilot_Commands
FaultTrees | FMEA | Safety Allocations
Top Event (Effect)  No_Pilot_Commands
Fault Trees  Cut Sets
FaultTrees
  No_Pilot_Commands
    No_Pilot_Commands ()
      No_Pilot_Commands (2929)
        AND (2930)
          Omission-PWMDecoder.FlightControls (1135)
            PWMDecoder.OFailure1 (1129)
            BusCreator1.OFailure1 (662)
            PilotJoystick.OFailure1 (1363)
            PilotJoystick.OFailure2 (1364)
            PWMDecoder.BusCreatorX.OFailure1 (1161)
          Omission-ModeSwitcher.ControlMode (931)
            ModeSwitcher.OFailure2 (920)
            PWMDecoder.OFailure2 (1130)
            BusCreator1.OFailure1 (662)
            PilotJoystick.OFailure1 (1363)
            PilotJoystick.OFailure2 (1364)
            ModeSwitcher.MultiportSwitch2.OFailure1 (1019)
          Omission-FailSafeController.FilteredControls (803)
            FailSafeController.OFailure1 (796)
            FlightControlMixer.OFailure1 (822)
            AND [Omission-FlightControlMixer.FlightControls] (827)
            BusCreator.OFailure1 (640)
            ...
```

(a) TFC-ALL No pilot commands fault tree



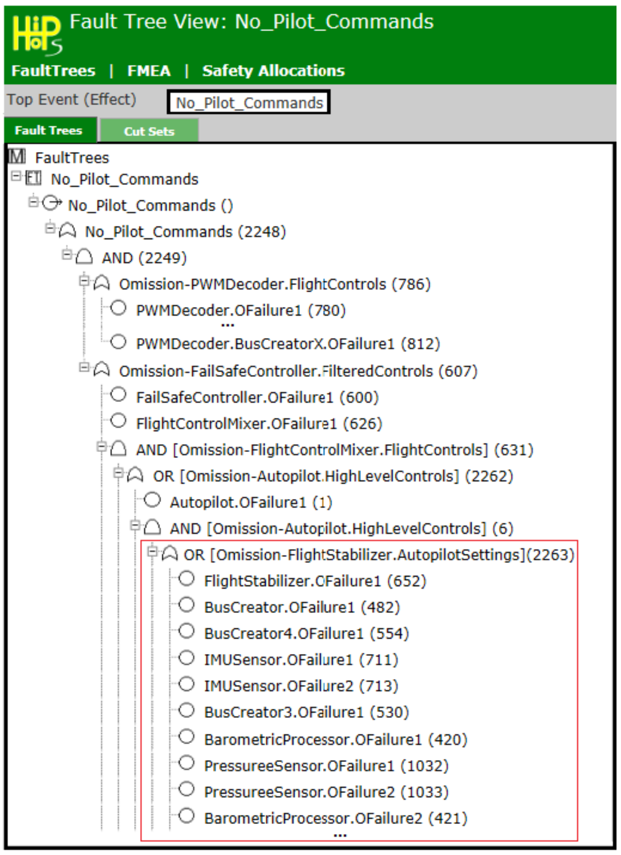
```
Fault Tree View: No_Pilot_Commands
FaultTrees | FMEA | Safety Allocations
Top Event (Effect)  No_Pilot_Commands
Fault Trees  Cut Sets
FaultTrees
  No_Pilot_Commands
    No_Pilot_Commands ()
      No_Pilot_Commands (2774)
        AND (2775)
          Omission-PWMDecoder.FlightControls (1042)
            PWMDecoder.OFailure1 (1036)
            BusCreator1.OFailure1 (557)
            PilotJoystick.OFailure1 (1276)
            PilotJoystick.OFailure2 (1277)
            PWMDecoder.BusCreatorX.OFailure1 (1068)
          Omission-ModeSwitcher.ControlMode (826)
            ModeSwitcher.OFailure2 (815)
            PWMDecoder.OFailure2 (1037)
            BusCreator1.OFailure1 (557)
            ...
            ModeSwitcher.MultiportSwitch2.OFailure1 (926)
            AND [Omission-ModeSwitcher.MultiportSwitch2.DefaultOut1] (931)
              OR [Omission-ModeSwitcher.PilotMode] (2789)
              ModeSwitcher.Constant1.OFailure1 (852)
              ...
          Omission-FailSafeController.FilteredControls (698)
            FailSafeController.OFailure1 (691)
            FlightControlMixer.OFailure1 (717)
            ...
```

(b) TFC-MAS No pilot commands fault tree



```
Fault Tree View: No_Pilot_Commands
FaultTrees | FMEA | Safety Allocations
Top Event (Effect)  No_Pilot_Commands
Fault Trees  Cut Sets
FaultTrees
  No_Pilot_Commands
    No_Pilot_Commands ()
      No_Pilot_Commands (2574)
        AND (2575)
          Omission-PWMDecoder.FlightControls (951)
            PWMDecoder.OFailure1 (947)
            ...
            PWMDecoder.BusCreatorX.OFailure1 (965)
          Omission-FailSafeController.FilteredControls (754)
            FailSafeController.OFailure1 (747)
            FlightControlMixer.OFailure1 (773)
            AND [Omission-FlightControlMixer.FlightControls] (778)
              OR [Omission-Autopilot.HighLevelControls] (2588)
                Autopilot.OFailure1 (27)
                AND [Omission-Autopilot.HighLevelControls] (32)
                  OR[Omission-AssistedModeSwitch.DefaultOut1](2589)
                    AssistedModeSwitch.OFailure1 (1)
                    BasicCommandProcessor.OFailure1 (508)
                    BusCreator2.OFailure1 (639)
                    NavigationSubsystem.OFailure1 (923)
                    NavigationSubsystem.OFailure2 (924)
                    NavigationSubsystem.OFailure3 (925)
                    NavigationSubsystem.OFailure4 (926)
                    BusCreator1.OFailure1 (613)
                    PilotJoystick.OFailure1 (1173)
                    PilotJoystick.OFailure2 (1174)
                    ...
```

(c) TFC-MAT No pilot commands fault tree



```
Fault Tree View: No_Pilot_Commands
FaultTrees | FMEA | Safety Allocations
Top Event (Effect)  No_Pilot_Commands
Fault Trees  Cut Sets
FaultTrees
  No_Pilot_Commands
    No_Pilot_Commands ()
      No_Pilot_Commands (2248)
        AND (2249)
          Omission-PWMDecoder.FlightControls (786)
            PWMDecoder.OFailure1 (780)
            ...
            PWMDecoder.BusCreatorX.OFailure1 (812)
          Omission-FailSafeController.FilteredControls (607)
            FailSafeController.OFailure1 (600)
            FlightControlMixer.OFailure1 (626)
            AND [Omission-FlightControlMixer.FlightControls] (631)
              OR [Omission-Autopilot.HighLevelControls] (2262)
                Autopilot.OFailure1 (1)
                AND [Omission-Autopilot.HighLevelControls] (6)
                  OR [Omission-FlightStabilizer.AutopilotSettings](2263)
                    FlightStabilizer.OFailure1 (652)
                    BusCreator.OFailure1 (482)
                    BusCreator4.OFailure1 (554)
                    IMUSensor.OFailure1 (711)
                    IMUSensor.OFailure2 (713)
                    BusCreator3.OFailure1 (530)
                    BarometricProcessor.OFailure1 (420)
                    PressureeSensor.OFailure1 (1032)
                    PressureeSensor.OFailure2 (1033)
                    BarometricProcessor.OFailure2 (421)
                    ...
```

(d) TFC-MAP No pilot commands fault tree

Figure D.6. No pilot commands fault trees for Tiriba flight control variants.

MAT and TFC-MAP variants, as shown in Figures D.6c and D.6d. An omission failure into "*DefaultOut1*" output port from the "*AssistedModeSwitch*" component and its associated causal chain contribute to the occurrence of an omission failure in the "*HighLevelControls*" output port from the "*Autopilot*" component in the TFC-MAT variant. On the other hand, the omission of the "*AutopilotSettings*" output port from the "*Flight Stabilizer*" component and its associated causal chain contribute to the occurrence of omission of "*HighLevelControls*" output port from the "*Autopilot*" component in the TFC-MAP system variant.

Variability was also found in the number of failure cut sets associated with each TiribaFC-SPL product variant, as shown in Table D.5. A cut set is a combination of basic events that can cause the top event of a fault tree, i.e., a system failure (NASA, 2002). For example, "*No pilot commands*" and "*Value pilot commands*" fault trees associated with TFC-ALL variant have respectively 108 and 154 cut sets, whereas the same fault trees associated with TFC-MAT variant contain respectively 60 and 88 cut sets. Such variation is further propagated throughout variant-specific FMEA results where different component failure modes contribute directly or indirectly to the occurrence of system hazards in each one of these variants. FMEA results show the relationships between the occurrence of component failures and their effects into the overall system safety.

Table D.5 – Variability in Tiriba flight control variant-specific fault trees.

Product Variant	***Fault Tree***	***Cut Sets***
TFC-ALL	No pilot commands	108
	Value pilot commands	154
TFC-MAT	No pilot commands	60
	Value pilot commands	88
TFC-MAS	No pilot commands	108
	Value pilot commands	154
TFC-MAP	No pilot commands	60
	Value pilot commands	88

Tables D.6 and D.7 show fragments of FMEA results for the TFC-ALL and TFC-MAT variants, considering the "*PWM Decoder*" component failures. These tables show the effects of the occurrence of component failures and whether each given component failure is a single point of failure or not. For example, the occurrence of "*OFailure1*" and "*OFailure2*" omission failure modes, and "*VFailure1*" *and* "*VFailure2*" value failure modes in the TFC-ALL variant (see Table D.6) indirectly contribute to the occurrence of

"*No_pilot_commands*" and *"Value_pilot_commands"* hazards in conjunction with other component failure modes. On the other hand, in the FMEA results for the TFC-MAT variant shown in Table D.7, only one omission failure (i.e., "*OFailure1"*), and one value failure (i.e., "*VFailure1"*) in the "*PWM Decoder"* component outputs indirectly contribute to the occurrence of "*No_pilot_commands"* and *"Value_pilot_commands"* hazards in conjunction with other component failure modes. The analysis of multiple variant-specific FMEA results provides the evidence of traceability of context and design variation throughout cause-effect relationships between component failures and the occurrence of system level hazards.

Table D.6 – Excerpt of FMEA results for the TFC-ALL variant.

Component	*Failure Mode*	*System Effect*	*Single Point of Failure*
PWM Decoder	OFailure1 (1029)	No_pilot_commands	False
	OFailure2 (1030)	No_pilot_commands	False
		Value_pilot_commands	False
	VFailure1 (1031)	Value_pilot_commands	False
	VFailure2 (1032)	Value_pilot_commands	False

Table D.7 – Excerpt of FMEA results for the TFC-MAT variant.

Component	*Failure Mode*	*System Effect*	*Single Point of Failure*
PWM Decoder	OFailure1 (947)	No_pilot_commands	False
	VFailure1 (948)	Value_pilot_commands	False

From the analysis of the complete FMEA results associated with each TiribaFC-SPL variant it has been identified that: 30 component failure modes indirectly contribute to the occurrence of hazards in the TFC-ALL system variant, 24 component failure modes were identified in the TFC-MAT variant, 30 component failure modes were identified in TFC-MAS variant, and 24 component failure modes, which indirectly contribute to the occurrence of hazards, were identified in the TFC-MAP system variant. Such analysis provides insights about how TiribaFC-SPL components can fail and contributing to the occurrence of hazards across different system variants, providing feedback for product line engineers to address safety of components across the product line.

D.7 SC-PLE Phase 8: TiribaFC Variant-Specific DAL Decomposition

This section presents the results of using design optimization techniques to automatically decompose the DALs allocated to system hazards associated with each TiribaFC product variant, throughout their contributing component failure modes. Such analysis allows safety analysts in establishing safety objectives to mitigate these component failures, and then, addressing the safety requirements of a particular product variant. The HiP-HOPS Tabu Search DAL decomposition optimization tool (SOROKOS *et al.* 2015) has been used to support the "*Product SIL Decomposition*" phase from the variant-specific fault tree analysis and FMEA results. Variant-specific failure cut sets stored into fault trees and FMEA results (i.e., an xml file) generated with the support of HiP-HOPS are input artefacts to the Tabu Search DAL decomposition algorithm performing the automatic decomposition of DALs allocated to variant-specific hazards throughout the contributing failure modes. The DAL decomposition was performed for each one of the following TiribaFC-SPL variant: TFC-ALL, TFC-MAT, TFC-MAS, and TFC-MAP. The DAL decomposition was performed based on the following cost heuristic that expresses the relative cost jumps of developing Tiriba flight control components according to different Development Assurance Levels (DALs): 0 (DAL E), 10 (DAL D A), 20 (DAL C), 40 (DAL B), and 50 (DAL A). This cost heuristic was used for illustrative purposes, but any other that safety analysts find more suitable can be used instead. Additional information on the DAL decomposition process can be found in the SAE ARP 4754A – Guidelines for Development of Civil Aircraft and Systems (EUROCAE, 2010).

Table D.8 shows an excerpt of the DAL decomposition results for TFC-MAT, TFC-MAS, TFC-MAP, and TFC-ALL system variants. These results correspond to the best DAL decomposition solution found and its total DAL cost found for each TiribaFC-SPL variant. DALs were allocated to 127 component failure modes into four different product variants. The failure modes are stated in terms of omission (*OFailureID*) and value (*VFailureID*) failure types. The DAL allocated to a given component failure mode may change according to the product variant. For example, the DAL allocated to a "*value*" failure mode of the "*PWM Decoder*" component is *"E"* (0) in the TFC-MAT

Table D.8 - DALs allocated to component failure modes in different TiribaFC-SPL variants.

TFC-SPL Component	*Failure Mode*	**TFC-MAT DAL**	**TFC-MAS DAL**	**TFC-MAP DAL**	**TFC-ALL DAL**
Barometric Processor	OFailure1	C(2)	C(2)	C(2)	C(2)
	OFailure2	C(2)	C(2)	C(2)	C(2)
PWM Decoder	OFailure1	A(4)	A(4)	A(4)	C(2)
	OFailure2	-	A(4)	E(0)	C(2)
	VFailure1	E(0)	C(2)	A(4)	C(2)
	VFailure2	-	C(2)	E(0)	C(2)
Pilot Joystick	OFailure1	A(4)	A(4)	A(4)	C(2)
	OFailure2	A(4)	A(4)	A(4)	C(2)
…	…	…	…	…	…
Pressure Sensor	OFailure1	C(2)	C(2)	C(2)	B(3)
	OFailure2	C(2)	C(2)	C(2)	C(2)

variant and *"A"* (4) in the TFC-MAP variant. Such difference emphasizes that different ways in which the TiribaFC-SPL components are combined in a product variant may change the stringency of the component DAL. Such difference also impact in changes in the total DAL cost associated with a given variant-specific system architecture. The HiP-HOPS Tabu Search DAL decomposition optimization algorithm has calculated the total DAL cost associated with each system variant according to the cost heuristic previously explained at the beginning of this subsection. The final DAL cost was: 620 for TFC-ALL, 530 for TFC-MAT, 860 for TFC-MAS, and 780 for TFC-MAP.

The analysis of multiple DAL decomposition results generated with the support of HiP-HOPS DAL decomposition optimization tool allows safety analysts to allocate safety integrity requirements to components, and establishing safety objectives to address safety across a range of product variants without being unnecessarily expensive or stringent. It allow safety analysts to assign expensive, in terms of costs and effort, safety objectives, development and safety assessment activities such verification, validation, testing only to highest critical components instead of all product line components.

D.8 SC-PLE Phase 9: TiribaFC-SPL Component DAL Decomposition

Variant-specific DAL decomposition results generated with the support of HiP-HOPS DAL decomposition optimization tool are input artefacts for the product line component SIL decomposition tool (OLIVEIRA *et al.* 2015), performing the automatic allocation of DALs to TiribaFC-SPL components. The tool performs such analysis on the basis of the following principle: the most stringent DAL allocated to a given failure mode of a component across multiple system variants is the DAL required to guarantee the safe usage of that component across the analyzed system variants. Firstly, the tool has derived the DALs that should be allocated to the product line components in each individual system variant. For each component associated with the given variant, the most stringent DAL allocated to a failure mode associated with that component is then, identified. For example, considering the TFC-ALL system variant, the analysis of the DALs allocated to the "*Pressure_Sensor*" component failure modes (DAL B for the "*OFailure1*" and DAL C for "*OFailure2*") shows that the component DAL for that variant should be DAL B. Finally, for each TiribaFC-SPL component, the tool analyses the DALs allocated the given component in each system variant to extract the most stringent DAL allocated to such component across the analyzed variants. As a result, DALs were allocated to 47 TiribaFC-SPL components.

Table D.9 shows the DALs allocated to 5 TiribaFC-SPL components into four different system variants analyzed from the perspective of two different usage scenarios: TFC-MAT and TFC-MAS system variants used in environment monitoring applications, deployed in a light UAV, operating in an uncontrolled airspace; and TFC-ALL and TFC-MAP variants used for defense applications, deployed a light UAV and operating in a controlled airspace. Analogously to the DALs allocated to component failure modes, the TiribaFC-SPL component DALs may change according to the system variant. For example, the DALs allocated to the "PWM Decoder", "*Pilot Joystick*", and "*Pressure Sensor*" components are respectively "A", "A", and "C" in the TFC-MAT variant, and "C", "C", and "B" in the TFC-ALL variant. From such analysis, the tool has generated the DALs to be allocated to TiribaFC-SPL components (column MAX DAL in Table D.9) to achieve product line process-based certification in compliance with ARP 4754A and DO-178C safety standards.

The analysis of the DAL decomposition results and their implications on the safety integrity requirements of potential usage of components provides useful feedback to the product line development process, contributing to meeting safety requirements without incurring unnecessary costs. The TiribaFC-SPL component DAL allocation results may guide product line engineers to take design decisions in order to address safety of product line components across the analyzed variants. It may also guide safety analysts in establishing safety objectives and structuring development and safety assessment processes for product line components in order to achieve process-based certification without being stringent or expressive. Therefore, by considering "*PWM Decoder*" and "*Fail Safe Controller*" components and their safety integrity requirements, Table D.10 shows a subset of safety objectives and their respective activities required for these components

Table 8.9 – TiribaFC-SPL component DAL decomposition results.

TFC-SPL Component	*TFC-MAT DAL*	*TFC-MAS DAL*	*TFC-MAP DAL*	*TFC-ALL DAL*	*MAX DAL*
Barometric Processor	C(2)	C(2)	C(2)	C(2)	C
Basic Command Processor	E(0)	E(0)	E(0)	-	E
Fail Safe Controller	C(2)	C(2)	C(2)	C(2)	C
Mode Switcher	E(0)	A(4)	E(0)	C(2)	A
PWM Decoder	A(4)	A(4)	A(4)	C(2)	A
…	…	…	…	…	…
Pressure Sensor	C(2)	C(2)	C(2)	B(3)	B

achieving process-based certification in compliance with SAE ARP 4754A and DO-178C safety standards.

Whereas "*PWM Decoder*" is a highly critical component, safety objectives with independence such as: "*aircraft functional hazard assessment*" (SAE ARP 4754A), and "*verification of additional code is achieved*" (DO-178C) should be addressed. Less critical components, such as "*Fail Safe Controller*", requires less stringent safety objectives that do not require independence. Instead of allocating highly stringent safety objectives to all TiribaFC-SPL components, product line component DAL decomposition allows safety analysts to assign safety objectives to a given component or a subset of components according to their integrity. Therefore, less stringent safety objectives assigned to less critical components means less development and safety assessment effort and costs to be considered for developing product line components in order to achieve process-based certification. Such analysis contributes to reduce the effort and costs of developing Tiriba flight control components, addressing safety requirements without compromise safety.

Table 8.10 – Component DALs and process-based certification safety objectives.

Component	***DAL***	***Safety Objectives***	***Activities***
PWM Decoder	A	The aircraft/system functional hazard assessment is performed with independence.	SAE ARP 4754A sections: 5.1.1, 5.2.3, 5.2.4.
		…	…
		High level requirements should comply with system requirements with independence.	DO-178C section 6.3.1: Analysis of compliance with system requirements.
		Verification of additional code that cannot be traced to the source code is achieved.	DO-178C section 6.4.4d: analysis to confirm that all the test cases are traceable to requirements.
Fail Safe Controller	C	The aircraft/system functional hazard assessment is performed.	SAE ARP 4754A sections: 5.1.1, 5.2.3, 5.2.4.
		…	…
		High level requirements should comply with system requirements.	DO-178C section 6.3.1: Analysis of compliance with system requirements.

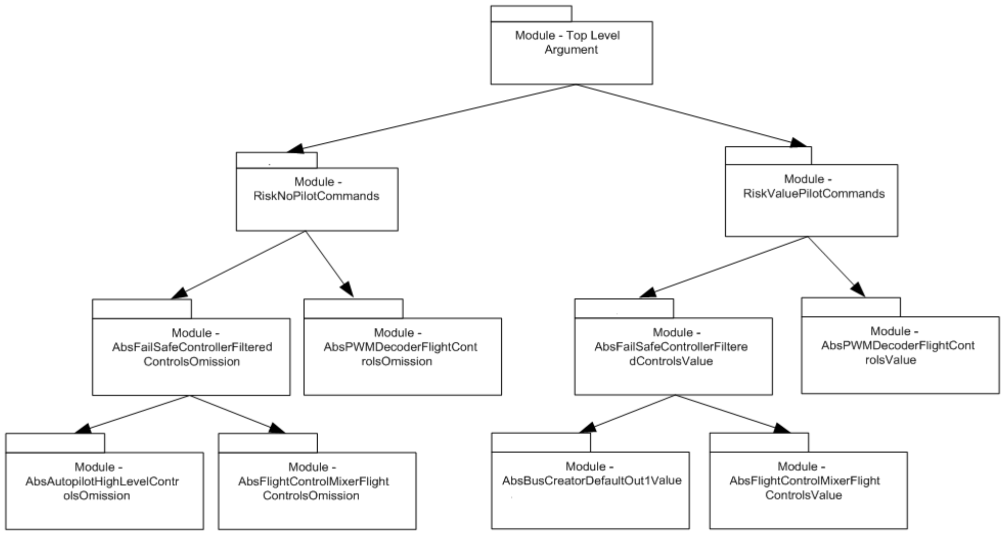
D.9 TribaFC-SPL Variant-Specific Model-Based Assurance Cases

The Model-Based Assurance Case (MBAC) approach and tooling (HAWKINS *et al.* 2015) was used to support the automatic generation of assurance cases for: "*Manual and Autonomous Pilot*", "*Manual and Assisted Pilot*", "*Manual and Autopilot*", and "*All Pilot*

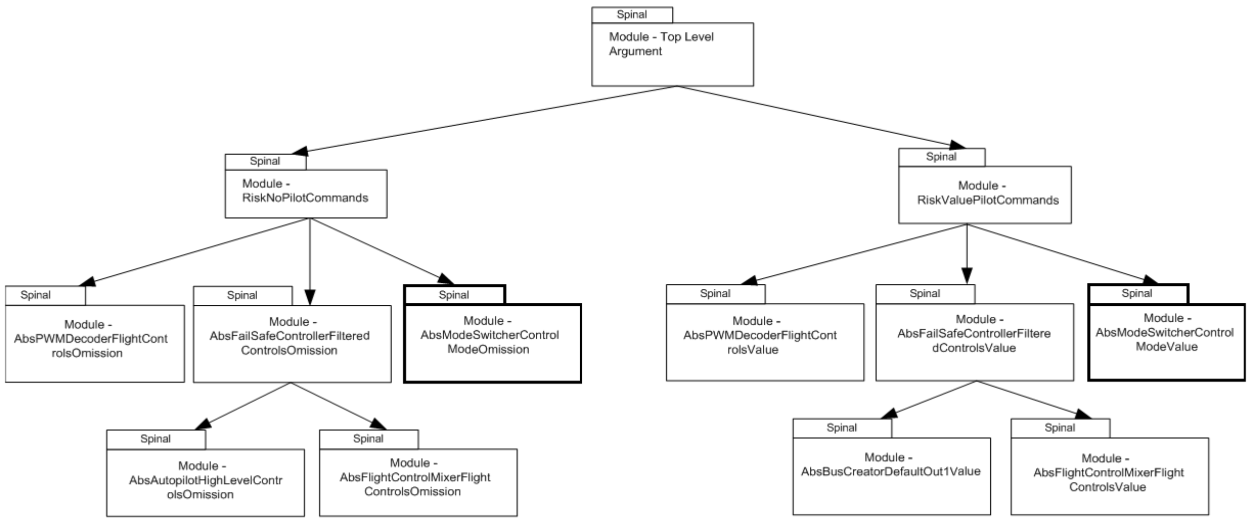
Modes" Tiriba flight control variants. For each system variant, the hazard-directed assurance case pattern models, the reference information models, and the weaving model artefacts presented in Chapter 6, together with variant-specific development and assessment models were input to configure the MBAC tool to generate a variant-specific assurance case. The MBAC tool generates variant-specific assurance cases in "*.gsnml"* and tabular formats. The "*gsnml"* file contains the description of argument elements and their relationships, and the tabular format shows the relationships between assurance pattern elements and their instantiation information in an instantiation table.

Figure D.7 shows excerpts of the modular view of the assurance cases generated for the TFC-MAT and TFC-ALL system variants. A variant-specific assurance case model comprises a top-level argument module, arguing that the system variant is acceptably safe to operate in its environment. This module is further supported by argument modules addressing the hazards posed by the given variant, called *Risk Argument* modules. Each risk argument module is further supported by modules addressing specific component faults that contribute to the occurrence of hazards, e.g., "*AbsOmissionPWMDecoderFlightControls"* module in Figure D.7a. These modules, named "*Absence of Hazardous Software Failure Mode*" modules argue that primary, secondary, and control failure modes, which can lead to the occurrence of a given hazardous software failure mode, are acceptable.

TiribaFC-SPL variant-specific assurance cases differ from each other in the number of argument modules, addressing component faults that contribute to the occurrence of hazards, called "*Absence of Hazardous Software Failure Mode"* modules and their supporting modules. Such variation is resultant of the impact of context and design variation in architecture and safety analysis models. Figure D.7 shows the variability in assurance case models generated for "*Manual and Autonomous Pilot*" (TFC-MAT), and "*All Pilot Modes*" (TFC-ALL) system variants. Although these variants have common "*Risk Argument"* modules, variability has been found in the "*Absence of Hazardous Software Failure Mode"* supporting modules. For example, the supporting modules for "*RiskNoPilotCommands"* argument module are "*AbsFailSafeControllerFilteredControlsOmission"* and "*AbsPWM-DecoderFlightControlsOmission"* in the TFC-MAT system variant (Figure D.7a). On the other hand, the "*RiskNoPilotCommandsOmission"* module is supported by these two modules and an additional "*AbsModeSwitcherControlModeOmission"* module in the TFC-ALL variant, as shown in the Figure D.7b. Such variation also occurs in the supporting modules for the "*RiskValuePilotCommands"* module in the TFC-MAT and TFC-ALL system variants.


Module - Top Level Argument
Module - RiskNoPilotCommands
Module - RiskValuePilotCommands
Module - AbsFailSafeControllerFilteredControlsOmission
Module - AbsPWMDecoderFlightControlsOmission
Module - AbsFailSafeControllerFilteredControlsValue
Module - AbsPWMDecoderFlightControlsValue
Module - AbsAutopilotHighLevelControlsOmission
Module - AbsFlightControlMixerFlightControlsOmission
Module - AbsBusCreatorDefaultOut1Value
Module - AbsFlightControlMixerFlightControlsValue

(a) TFC-MAT assurance case modula view


Spinal
Module - Top Level Argument
Module - RiskNoPilotCommands
Module - RiskValuePilotCommands
Module - AbsPWMDecoderFlightControlsOmission
Module - AbsFailSafeControllerFilteredControlsOmission
Module - AbsModeSwitcherControlModeOmission
Module - AbsPWMDecoderFlightControlsValue
Module - AbsFailSafeControllerFilteredControlsValue
Module - AbsModeSwitcherControlModeValue
Module - AbsAutopilotHighLevelControlsOmission
Module - AbsFlightControlMixerFlightControlsOmission
Module - AbsBusCreatorDefaultOut1Value
Module - AbsFlightControlMixerFlightControlsValue

(b) TFC-ALL assurance case modular view

Figure D.7. TiribaFC-SPL variant-specific assurance cases[24].

Variation in "*Absence of Hazardous Software Failure Mode"* modules are further propagated throughout their supporting claims. These claims argue that all causes of failures in other components, i.e., secondary failures, leading to the occurrence of a given hazardous software failure mode are acceptable. The following subsections show the detailed description of the "*Top-Level Argument"*, "*Risk Argument"*, and *"Absence of Hazardous Software Failure Mode"* argument modules generated for the TFC-MAT system variant shown in Figure D.7a.

[24] Only certain parts of the Tiriba flight control variant-specific assurance cases are presented this thesis due to commercial sensitivity of the Tiriba UAV system.

D.7.1 Top-Level Argument

The top-level argument, shown in Figure D.8, is a hazard and risk-directed module. This module argues that the TFC-MAT system variant is acceptably safe to operate in its environment by addressing and showing the risk posed by each variant-specific hazard is acceptable. Each risk argument is encapsulated into a separated module, which justifies that the component failures that can cause a given hazard are acceptable, i.e. do not lead the given system variant to an unsafe state. Such justification is defined on the basis of the DAL allocated to each hazard, according to the SAE ARP 4754A risk classification as stated in "*C3"* context element in Figure D.8.

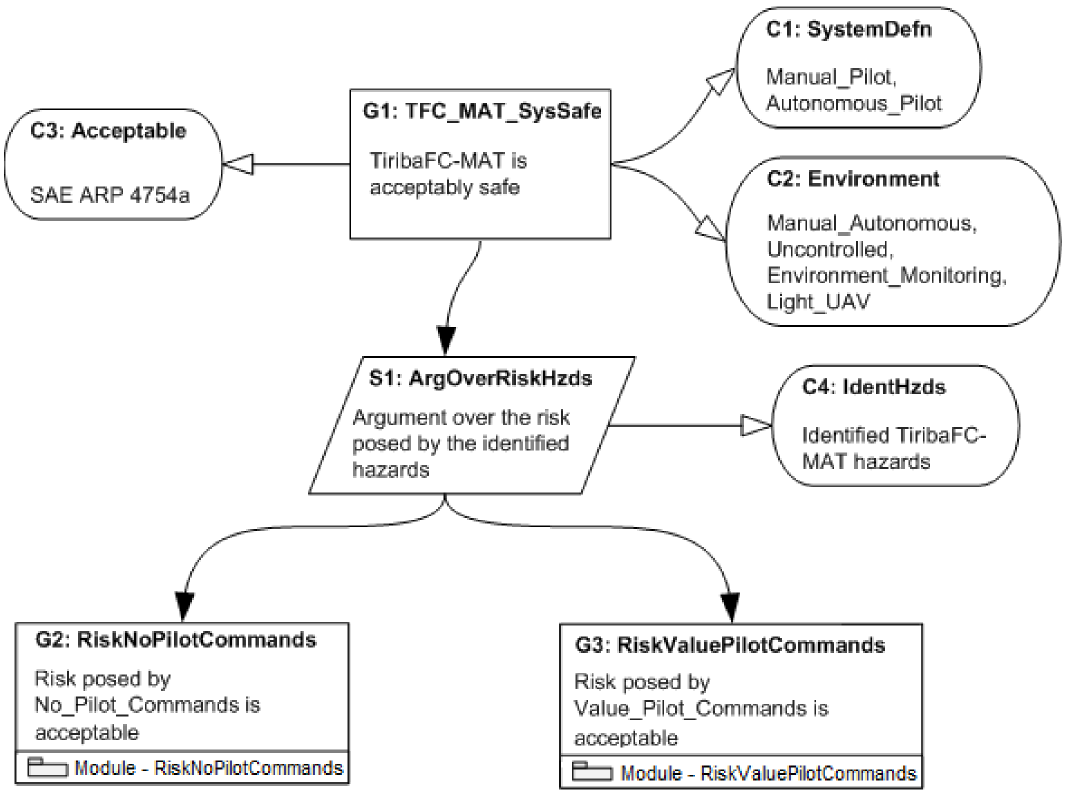

C1: SystemDefn
Manual_Pilot, Autonomous_Pilot
G1: TFC_MAT_SysSafe
TiribaFC-MAT is acceptably safe
C3: Acceptable
SAE ARP 4754a
C2: Environment
Manual_Autonomous, Uncontrolled, Environment_Monitoring, Light_UAV
S1: ArgOverRiskHzds
Argument over the risk posed by the identified hazards
C4: IdentHzds
Identified TiribaFC-MAT hazards
G2: RiskNoPilotCommands
Risk posed by No_Pilot_Commands is acceptable
Module - RiskNoPilotCommands
G3: RiskValuePilotCommands
Risk posed by Value_Pilot_Commands is acceptable
Module - RiskValuePilotCommands

Figure D.8. TFC-MAT top-level argument module.

D.7.2 Risk and Absence of Hazardous Software Failure Mode Argument Modules

The TFC-MAT top-level argument module is supported by "*RiskNoPilotCommands"* and "*RiskValuePilotCommands"* modules. In both modules, the claim arguing that "*the risk posed by the given system hazard is acceptable*" is supported by sub-claims arguing that each contributing "*hazardous software failure mode is acceptable*". Figure D.9 shows the structure of the "*NoPilotCommandsRisk"* argument module. The "*G2"* top-level level claim is stated in the context of "*DAL A*" allocated to the "*No pilot commands"* system hazard ("*C5"*), and the

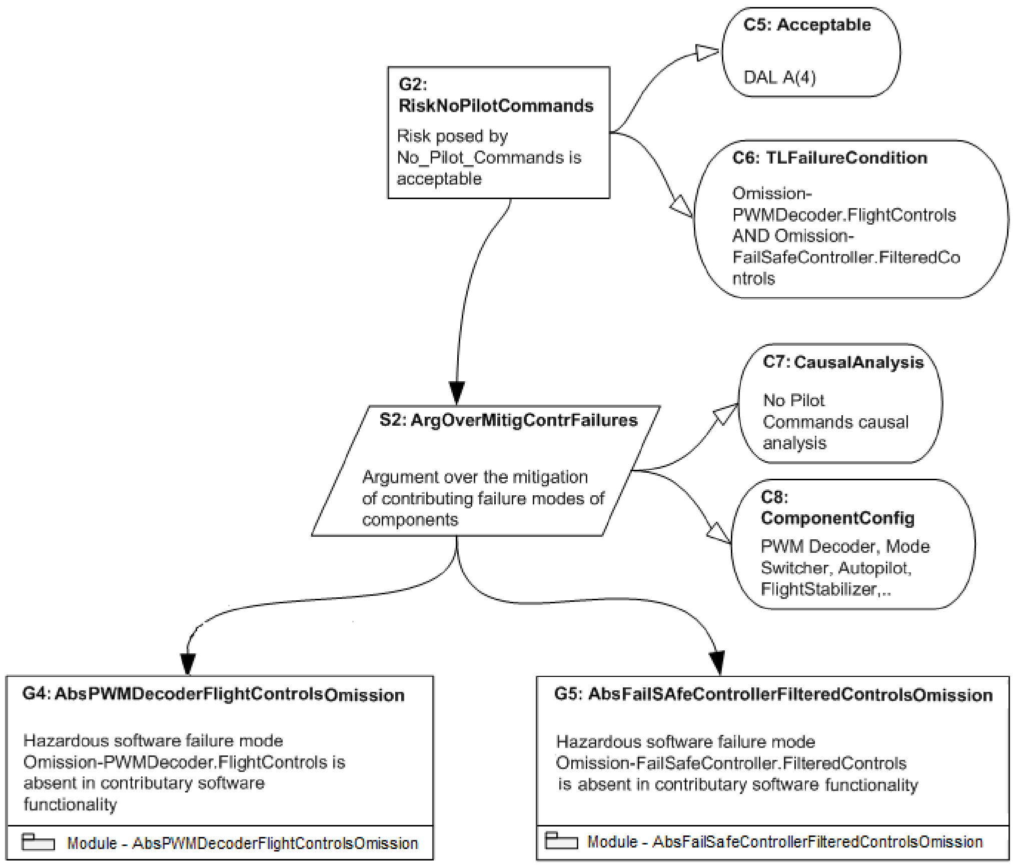

C5: Acceptable
DAL A(4)
G2: RiskNoPilotCommands
Risk posed by No_Pilot_Commands is acceptable
C6: TLFailureCondition
Omission-PWMDecoder.FlightControls AND Omission-FailSafeController.FilteredControls
C7: CausalAnalysis
No Pilot Commands causal analysis
S2: ArgOverMitigContrFailures
Argument over the mitigation of contributing failure modes of components
C8: ComponentConfig
PWM Decoder, Mode Switcher, Autopilot, FlightStabilizer,..
G4: AbsPWMDecoderFlightControlsOmission
Hazardous software failure mode Omission-PWMDecoder.FlightControls is absent in contributary software functionality
Module - AbsPWMDecoderFlightControlsOmission
G5: AbsFailSAfeControllerFilteredControlsOmission
Hazardous software failure mode Omission-FailSafeController.FilteredControls is absent in contributary software functionality
Module - AbsFailSafeControllerFilteredControlsOmission

Figure D.9. No pilot commands risk module.

top-level failure condition leading to this hazard (“*C6”*). This claim is decomposed into sub-claims arguing the absence of component failures contributing to the occurrence of “*No pilot commands”* hazard (“*S1”*), in this case, omission failures in “*PWMDecoder”* and “*FailSafeController”* component outputs, which contribute to the occurrence of the omission of UAV flight control commands. Such decomposition strategy is defined in the context of the hazard causal chain defined in the “*No pilot commands*” fault tree (see Figure D.6c), and the TFC- MAT variant-specific components defined in its architecture model (*“C8”*). “*G4”* and “*G5”* are references to “*AbsPWMDecoderFlightControlsOmission”* and “*AbsFailSafe-ControllerFilteredControlsOmission”* modules. In these modules, claims arguing the absence of omission failures in “*PWMDecoder”* and “*FailSafeController”* components are decomposed into sub-claims arguing that primary, secondary, and control failure modes that can cause these failures are acceptable. Figure D.10 shows the “*AbsPWMDecoderFlightControlsOmission”* module. This module decomposes the claim “*G4”* into sub-claims arguing that: an internal failure in *PWMDecoder* component is acceptable (“*G4.1”*), failure modes of other components that contribute to the omission of *“PWMDecoder.FlightControls”* output are acceptable (“*G4.2”*), and that the “*PWMDecoder”*

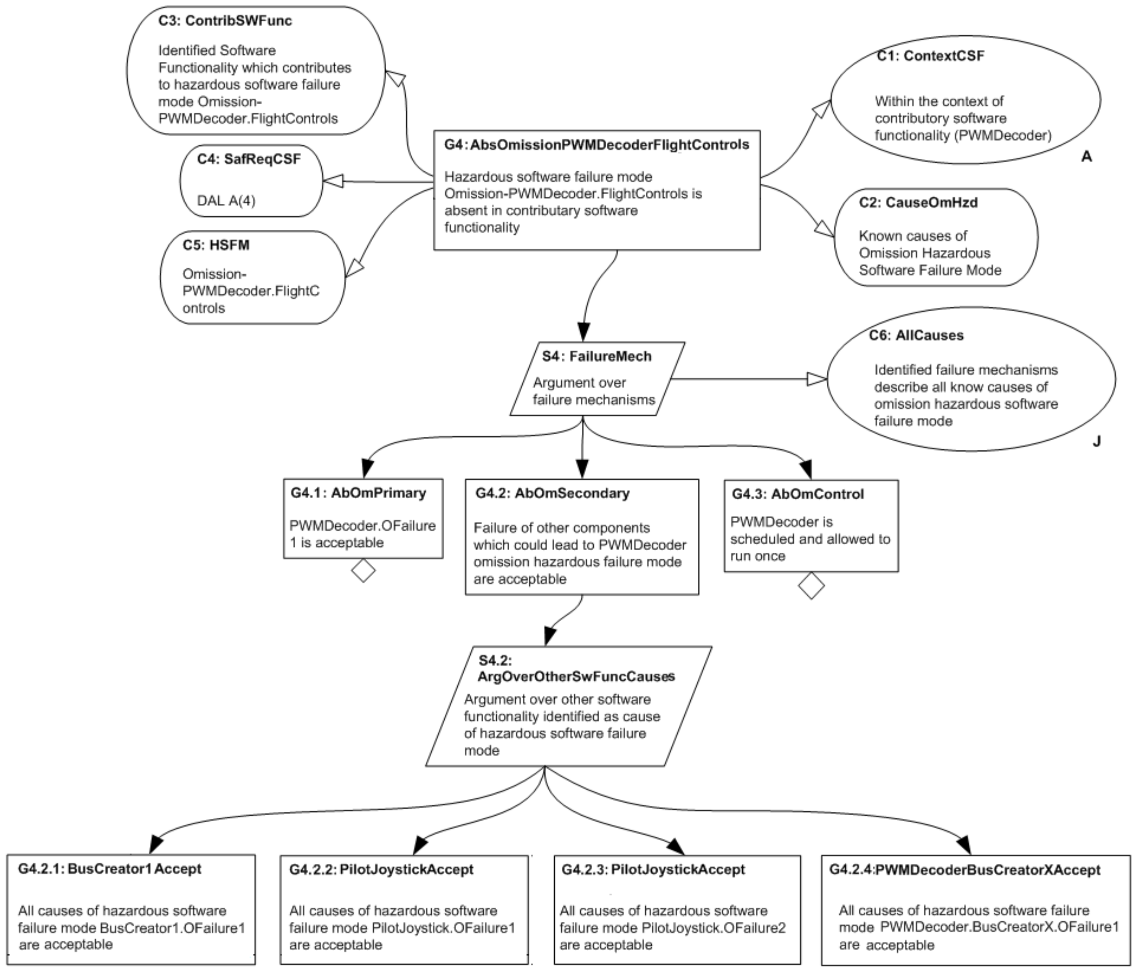

C3: ContribSWFunc
Identified Software Functionality which contributes to hazardous software failure mode Omission-PWMDecoder.FlightControls
C1: ContextCSF
Within the context of contributory software functionality (PWMDecoder)
A
C4: SafReqCSF
DAL A(4)
G4:AbsOmissionPWMDecoderFlightControls
Hazardous software failure mode Omission-PWMDecoder.FlightControls is absent in contributary software functionality
C2: CauseOmHzd
Known causes of Omission Hazardous Software Failure Mode
C5: HSFM
Omission-PWMDecoder.FlightControls
S4: FailureMech
Argument over failure mechanisms
C6: AllCauses
Identified failure mechanisms describe all know causes of omission hazardous software failure mode
J
G4.1: AbOmPrimary
PWMDecoder.OFailure1 is acceptable
G4.2: AbOmSecondary
Failure of other components which could lead to PWMDecoder omission hazardous failure mode are acceptable
G4.3: AbOmControl
PWMDecoder is scheduled and allowed to run once
S4.2: ArgOverOtherSwFuncCauses
Argument over other software functionality identified as cause of hazardous software failure mode
G4.2.1: BusCreator1 Accept
All causes of hazardous software failure mode BusCreator1.OFailure1 are acceptable
G4.2.2: PilotJoystickAccept
All causes of hazardous software failure mode PilotJoystick.OFailure1 are acceptable
G4.2.3: PilotJoystickAccept
All causes of hazardous software failure mode PilotJoystick.OFailure2 are acceptable
G4.2.4:PWMDecoderBusCreatorXAccept
All causes of hazardous software failure mode PWMDecoder.BusCreatorX.OFailure1 are acceptable

Figure 8.10. Absence of omission of *PWMDecoder.FlightControls* module.

component is scheduled and allowed to run once" ("*G4.3"*). *"G4.2"* claim is further decomposed into fault mitigation modules arguing the absence of omission failures in *BusCreator1*, *PilotJoystick*, and *PWMDecoder.BusCreatorX* component outputs.

The argument modules presented in Figures D.8, D.9, and 8.10 are instances of "*Hazard Avoidance*" (KELLY and McDERMID, 1997), "*Risk Argument*" (HABLI, 2009), and "*Absence of Hazardous Software Failure Mode*" (WEAVER, 2003) assurance case patterns. These argument modules contain references to information elements provided by different TFC-MAT variant-specific models. Table D.11 shows the relationship between "*Top-Level Argument"* elements shown in Figure D.8, and information elements from different system models. For example, the "*TiribaFC-MAT*" information stated in "*TFC_MAT_SysSafe"* and "*IdentHzds"* argument elements is provided by the TFC-MAT architecture model. "*No_Pilot_Commands"* and "*Value_Pilot_Commands"* hazard information elements stated into "*G2"* and "*G3"* module references are provided by the TFC-MAT failure model. Therefore, changes in the Tiriba flight control variant are automatically propagated throughout the assurance case. In addition, the automated traceability between the assurance case and evidence items, i.e., development and assessment artefacts, referenced in the assurance case, is achieved. Therefore, the generated assurance cases contribute to goal-based certification of Tiriba flight control system variants.

Table D.11 – Assurance case pattern and system model elements.

Assurance Case Model	*GSN Element (s)*	*Information Element*	*Model Element*	*Source System Model*
Top-Level Argument	G1, C4	TiribaFC-MAT	Model.name	Simulink model
	C1	Manual_Pilot, Autonomous_Pilot	Feature.name	Feature model
	C2	Manual_Autonomous, Uncontrolled, Environment_Monitoring, Light_UAV	ContextFeature.name	Context model
	C3	SAE ARP 4754a	Model.description	Failure model
	G2, G3 module ref.	No_Pilot_Commands, Value_Pilot_Commands	Hazard.name	Failure Model

References

BOTTERWECK, G., POLZER, A., KOWALEWSKI, S. Using higher-order transformations to derive variability mechanism for embedded systems, In: WORKSHOPS AND SYMPOSIA AT MODELS, 2009, Denver, USA, **Proceedings**…, Springer, Models in Software Engineering, 2010, v. 6002, p. 68-82.

BRANCO, K., PELIZZONI, J., NERIS, L., TRINDADE, O., OSORIO, F., WOLF, D. Tiriba: a new approach of UAV based on model-driven development and multiprocessors. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2011, Shanghai, China, **Proceedings**…, IEEE, p. 1-4.

EUROCAE. ARP4754A - Guidelines for Development of Civil Aircraft and Systems, EUROCAE, 2010.

HABLI, I. 2009. Model-based assurance of safety-critical product lines. PhD Thesis, Department of Computer Science, The University of York, York, United Kingdom, 2009.

HAWKINS, R., HABLI, I., KOLOVOS, D., PAIGE, R., KELLY, T. Weaving an assurance case from design: a model-based approach. In: INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEMS ENGINEERING (HASE), 16th, 2015, Daytona Beach, USA, **Proceedings**…, IEEE, 2015, p.110-117.

KELLY, T., MCDERMID, J. Safety case construction and reuse using patterns. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY, 16th, 1997, **Proceedings**…, Springer-London, 1997, p. 55–69.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). Fault Tree Analysis with Aerospace Applications. Technical report, an update to NUREG-0492, NASA Office of Safety and Mission Assurance, Washington, DC, USA, August, 2002.

OLIVEIRA, A. L., BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., AZEVEDO, L., PARKER, D., HABLI, I., KELLY, T. Automatic allocation of safety requirements to components of a software product line. In: IFAC SYMPOSIUM ON FAULT DETECTION, SUPERVISION AND SAFETY FOR TECHNICAL PROCESSES (SAFEPROCESS'15), 9th, 2015, Paris, France, **Proceedings**…, Elsevier, 2015, v. 48, i. 41, p. 1309-1314.

PAPADOPOULOS, Y., WALKER, M., PARKER, D., RÜDE, E., HAMANN. Engineering failure analysis and design optimization with HIP-HOPS. **Journal of Engineering Failure Analysis**, v. 18, i. 2, p. 590-608, 2011..

RTCA. DO-178C Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, 2012.

SOROKOS I., PAPADOPOULOS Y., AZEVEDO L., PARKER D., WALKER D. Automating allocation of development assurance levels: an extension to HiP-HOPS. In: IFAC INTERNATIONAL WORKSHOP ON DEPENDABLE CONTROL OF DISCRETE SYSTEMS (DCDS'15), 5th, 2015, **Proceedings**…, Elsevier, 2015, v. 48, i. 7, p. 9-14.

VASILEVSKIY, A., HAUGEN, Ø., CHAUVEL, F., JOHANSEN, M. F., SHIMBARA, D. The BVR tool bundle to support product line engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE PRODUCT LINE (SPLC '15), 19th, 2015, **Proceedings**…, ACM, New York, NY, USA, p. 380-384.

WEAVER, R. A. The safety of software – constructing and assuring arguments. PhD Thesis, Department of Computer Science, University of York, York, United Kingdom, 2003.

Appendix E

LIST OF PUBLICATIONS

Excerpts of this thesis have been either published or submitted for the appreciation of editorial boards from journals, conferences, and workshops, according to the list of publications presented below:

- **OLIVEIRA, A. L.,** BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., HABLI, I., KELLY, T. Model-based safety analysis of software product lines. **International Journal of Embedded Systems**, Inderscience Publishers, 2016. **[*to be published*]**
- **OLIVEIRA, A. L.,** BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., HABLI, I., KELLY, T. Supporting the automated generation of modular product line safety cases. In: INTERNATIONAL CONFERENCE ON DEPENDABILITY AND COMPLEX SYSTEMS (DepCoS-RELCOMEX), 10th, June, 2015, Brunów, Poland, **Proceedings**…, Springer Int. Publishing, Advances in Intelligent Systems and Computing, Theory and Engineering of Complex Systems and Dependability, 2015, v. 365, p. 319-330.
- **OLIVEIRA, A. L.,** BRAGA, R. T. V., MASIERO, P. C., PAPADOPOULOS, Y., AZEVEDO, L., PARKER, D., HABLI, I., KELLY, T. Automatic allocation of safety requirements to components of a software product line. In: IFAC SYMPOSIUM ON FAULT DETECTION, SUPERVISION AND SAFETY FOR TECHNICAL PROCESSES (SAFEPROCESS'15), 9th, 2015, Paris, France, **Proceedings**…, Elsevier, 2015, v. 48, i. 41, p. 1309-1314.
- **OLIVEIRA, A. L.,** BRAGA, R. T. B., MASIERO, P. C., PAPADOPOULOS, Y., HABLI, I., KELLY, T. A model-based approach to support the automatic safety

analysis of multiple product line products. In: BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEMS ENGINEERING, 4th, 2014, Manaus, Brazil, **Proceedings**…, IEEE, 2014, p. 7-12. **[*best paper awards*]**

- **OLIVEIRA, A. L.,** BRAGA, R. T. V., MASIERO, P. C., HABLI, I., KELLY, T. A pattern to argue the compliance of the system safety requirements decomposition. In: LATIN AMERICAN CONFERENCE ON PATTERN LANGUAGE OF PROGRAMS (SugarLoaf PLoP), 10th, Nov., 2014, Ilhabela, Brazil, **Proceedings**…, Hilside Group, 2014, p. N/A.
- **OLIVEIRA, A. L.,** BRAGA, R. T. V., MASIERO, P. C., HABLI, I. KELLY, T. Impact of feature interaction on the safety analysis for unmanned avionics product lines. In: INTERNATIONAL CONFERENCE ON COMPUTER SAFETY, RELIABILITY AND SECURITY (SAFECOMP) – FAST ABSTRACTS, 32nd, 2013, Toulouse, France, **Proceedings**…., Available on-line: <https://hal.archives-ouvertes.fr/hal-00926563/document>. Accessed in: December, 15th, 2015.