Modeling the Variability of System Safety Analysis using State-Machine Diagrams

Lucas Bressan[1], André L. de Oliveira[1], Fernanda C. Campos[1], Leonardo Montecchi[2], Rafael Capilla[3], David Parker[4], Koorosh Aslansefat[4], Yiannis Papadopoulos[4]

[1]Universidade Federal de Juiz de Fora (UFJF), Juiz de Fora MG, Brazil
{lucasbressan, andre.oliveira, fernanda.campos}@ice.ufjf.br
[2]Norwegian University of Science and Technology, Trondheim, Norway
leonardo.montecchi@ntnu.no
[3]Universidad Rey Juan Carlos (URJC), Madrid, Spain
rafael.capilla@urjc.es
[4]University of Hull, U.K.
{D.J.Parker, K.aslansefat, y.i.papadopoulos}@hull.ac.uk

Abstract. Software Product Lines (SPLs) enable and maximize reuse of software artefacts, using software variability as central technique. In Model-Based Safety Analysis, system and software models are annotated with failure models that are used to produce safety analysis artefacts like fault trees and FMEAs. However, little work has been done to show MBSA in product lines, exploiting failure models to create safety analyses for variants in the product line. State machines have been widely used to support both fault propagation and probabilistic system safety analysis. In this paper, we introduce an approach to support variability modeling and reuse of state-machine diagrams used for system safety analysis. The approach enhances traditional software product line cycle with new activities aimed to support the reuse of safety information using state-machine diagrams and facilitates the management of the diversity of functional safety across system configurations using variability models. We evaluate our approach using an automotive braking system where we show reduction of the burden of safety analysis and improvements in traceability between safety artifacts and variability abstractions.

Keywords: Safety analysis, state-machine diagrams, software product lines, variability, reuse.

1 Introduction

Safety-critical software is becoming more complex due to the greater possibilities offered for inter-connectivity as well as due the increased computing power [1]. The mass customization in the automotive industry leads to a higher variability within a single product with thousands of variations points [2]. Automotive electronic control units (ECUs) [3], used in airbags, electronic window lifter and driver assistant systems, and powertrain controllers [4] are highly variant-intensive. A failure in safety-

critical software may lead to catastrophic consequences to the environment, finances, or putting human lives at risk.

In safety-critical systems, the reuse of software components demand the reuse of safety analysis artefacts. The safety information is the key point of diversity in safety analysis of variant-intensive systems and software product lines. Reusing state-machine diagrams for system safety analysis demands a way to manage diversity of emergent hazardous states and safety requirements. Safety requirements are placed measures to eliminate or minimize hazard and/or component failure effects on the overall safety. Current industry practice for safety artifact reuse in certification processes relies only on clone & own approaches [1] [3]. Also, balancing safety certification and reuse of safety information still remains a challenge to product line safety analysis [2] [5] [6] [7].

In order to address these challenges, we propose an approach, which extends a previous work [8] with the provision of semi-automated support for variability modeling and management in system safety analysis using state-machine diagrams. We evaluated the effectiveness of our solution in a variant-intensive automotive wheel braking system to enabling the systematic reuse of safety information of state-based safety models and reducing the burden to performing safety analysis activities. The remainder of this paper is as follows. Section 2 presents the related work. Section 3 introduces the background and concepts required to understand the proposed solution. In Section 4 we describe our approach consisting in new activities in addition to the classical Software Product Line (SPL) [9] lifecycle to support the reuse of safety information. Section 5 provides an evaluation of our approach in the automotive domain and Section 6 presents the conclusions and future work.

2 Related Work

The research on variability management in system safety analysis covers extensions of traditional safety analysis techniques to suit software product line processes [10] [11] [12] [13], and model-based techniques [5] [7] [14] [15]. Dehlinger and Lutz [10] and Feng and Lutz [11] proposed Software Fault Tree Analysis (SFTA) for product lines. In a SFTA, each fault tree leaf node is enriched with variability information in the domain engineering phase. In the application engineering phase, a pruning technique is applied for reusing software fault trees in a specific product variant. The Product Line SFTA was further extended to integrate state-based modeling [13]. This allows mapping fault tree leaf nodes to components and specifying the behaviour of a component in a state chart. Performing variability management on safety properties prior to FTA and Failure Modes and Effects Analysis (FMEA) enables the traceability of variability in the design and context through the safety lifecycle and the systematic reuse of safety assets. Schulze et al. [5] proposed an approach that integrates Medini[1] safety analysis and pure::variants[2] tools to support variability management in automotive functional safety. Kaßmeyer et al. [14] [15] propose a systematic model-based

[1] https://www.ansys.com/products/systems/ansys-medini-analyze
[2] https://www.pure-systems.com/products/pure-variants-9.html

approach integrated with change impact analysis techniques. This approach combines requirements engineering, architectural design, safety analysis, and variability management tools, allowing seamless safety engineering across product variants. Domis et al. [7] extended the Component Integrated Component Fault Trees (C^2FT) with variation points and integrated it within UML via a profile into Enterprise Architect[3] commercial tool. These approaches [7] [14] provide efficient solutions for variability management and change impact analysis in automotive functional safety.

Montecchi et al. [16] provide formalism for variability modeling into Stochastic Activity Networks (SAN) models. Bressan et al. [17] presented an approach to generate variants of SPEM 2.0 process models for automotive software components based on the allocated Automotive Safety Integrity Levels (ASILs). State of the art Model-Based Design [18] [19] [20] [21] and Safety Assessment frameworks [22] [23] [24] provide certain degree of support for variant management and reuse of safety models via inheritance [18] [19] [20] [21] [22] [23], and specification of different implementations for the failure behaviour of a component stored into error model libraries [24]. However, these mechanisms are limited to manage variability into safety models at a coarse grained level, not supporting variability on hazard causes, error states and state transitions. Although conventional SPL approaches [9] [25] have been extended [5][6] to address safety certification, supporting the diversity of safety information is still challenging [1] [8] due to: **CH1** the lack of considering the impact of variability in design choices for specifying component failure models, e.g., qualitative failure logic and stochastic state machine diagrams, used to create safety analysis for system variants; **CH2**: lack of traceability between SPL variability abstractions, design choices, and variation into elements and parameters from state machine diagrams; and **CH3**: lack of mechanisms to resolve structural and parametric variability into failure models (state machines).

3 Background

We provide an overview of Product Lines, Base Variability Resolution, ISO 26262 safety lifecycle, and the CHESS-State-Based Analysis used in the proposed solution.

3.1 Software Product Lines and Base Variability Resolution

A Software Product Line (SPL) is a set of software-intensive system that share a common and manageable set of features that satisfy the specific needs of a particular market segment [9]. *Feature* stands for a distinct system characteristic visible to the end-user [25], e.g., *wheel braking*. The commonalities and variability's of a family of systems from a particular domain and their relationships are expressed in a feature model. Feature modeling represents the product line variability at the highest abstraction level. Features express high-level functional and quality requirements from an application domain. Feature-Oriented Domain Analysis (FODA) [25] is the first and

[3] https://sparxsystems.com/

widely used feature modeling notation, which supports the specification of mandatory, optional and alternative features and structural relationships between features, i.e., decomposition, generalization, specialization, and parameterization. The wheel braking feature of a car family can be configured as four or front wheel braking (Fig. 1).

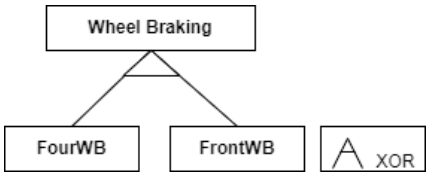

Fig. 1. Features for an automotive wheel braking system family.

The development of a SPL encompasses domain engineering and application engineering processes. The product line core assets are produced during the domain engineering and further reused in the application engineering [25]. The domain engineering process involves domain analysis, for identifying commonalities and variability in the domain requirements using a feature model, the realization of common and variant domain features by developing the SPL architecture and implementing their components (core assets), and specification of traceability between *features* and the core assets. In the application engineering, we create system variants from the core assets, produced in the domain engineering, by exploiting variability and ensuring the correct binding of variation points according to product requirements. A variation point stands for places in the domain artefacts, e.g., design, where variability can arise.

Base Variability Resolution (BVR) [26] is a language and tool, built upon Common Variability Language (CVL) [27] standard, to supports standard variability modeling in Meta-Object Facility (MOF)[4] compliant base models, e.g., UML and SysML models. BVR supports the generation of product variants from a base model via three different and inter-related models for specification of variability abstractions and variability realization. Variability abstractions are specified in a *VSpec* (feature) model supplemented with constraints, and a corresponding resolution model (*VResolution*) that defines the feature selection for a given product variant. In a *VSpec* model, the mandatory features are connected to the parent feature via *solid lines*, and *dashed lines* represent optionality. The *VSpec* model allows the specification of constraints between *features* in Object Constraint Language (OCL).

3.2 The ISO 26262 Safety Lifecycle

ISO 26262 [28] prescribes requirements for functional safety on electrical/electronic systems embedded into small and medium sized (up to 3.5 tons) general purpose road vehicles. This standard provides a safety lifecycle for automotive systems, a risk-based approach for determining risk classes (Automotive Safety Integrity Levels - ASILs), and requirements for validation and confirmation measures to ensure that an acceptable level of safety is achieved.

[4] https://www.omg.org/mof/

After allocating functions to systems/subsystems/components (items) and specifying their dependencies and interactions with the environment at the vehicle level, the *safety lifecycle is initiated*, i.e., the development category of all parts of the item is analyzed to make the distinction between a new item development and a modification to an existing item. *Items* can be software, electrical or electronic system components. *Hazard Analysis and Risk Assessment* (HARA) is performed to identify and categorize the hazards that malfunctions in the item can trigger and formulate the safety goals related to the mitigation of hazardous events, to avoid unreasonable risk. The *Functional Safety Concept* shall derive the functional safety requirements from the safety goals and allocating them to preliminary architectural elements of the item. Fault detection and mitigation, and fault tolerance mechanisms are examples of functional safety requirements. Safety goals are results from HARA, expressed in the form of ASILs, which define measures to mitigate hazard effects. Different safety goals are defined to achieve each ASIL. Achieving compliance with standards increases around 75% to 150% of the total development costs [29], due to the additional effort with system safety analysis, verification, and validation activities. Although standards provide guidance to the development of single products, the industrial production is inherently variable [1][3][4], which different products are built upon a common base system. Variation in the design propagates throughout potential fault, error, and failure behaviors of subsystems and components, stored into failure logic and/or stochastic state-machine diagrams, which may rise in different variants and environments.

3.3 CHESS Framework and CHESS State-Based Analysis

The Composition with guarantees for High-integrity Embedded Software components aassembly (CHESS) framework is a model-driven, component-based, methodology and toolset support for the design, development, and safety analysis of high-integrity systems from different domains [21]. CHESS defines a UML-based modeling language, called Composition with guarantees for High-integrity Embedded Software components aSsembly Modeling Language (CHESSML) [20], and includes a set of plug-ins to perform code generation, constraint checking, performance, and safety/dependability analyses. The CHESS framework supports qualitative fault propagation, using CHESS-Failure Logic Analysis (CHESS-FLA) [22], and quantitative/probabilistic state-based (via CHESS-SBA) [30] safety/dependability analysis techniques. CHESS-FLA allows engineers to decorate CHESSML component-based models with dependability information, execute Failure Logic Analysis, and get the results back-propagated onto the original model. In this paper, we focus solely on CHESS-SBA. CHESS State-Based Analysis (CHESS-SBA) [30] supports safety analysis (Hazard Analysis and Risk Assessment - HARA) using state machine diagrams. The term “state-based” denotes this technique uses a representation of the system based on possible states with respect to dependability, and possible transitions between them. The CHESS-SBA plugin supports users to perform quantitative (probabilistic) safety/dependability analysis on CHESSML models, by enriching them with stochastic dependability information, including failure and repair distribution of components, propagation delays and probabilities, and fault-tolerance and maintainability concepts. Such information can be added to a CHESSML element in two ways: i) via

«*simpleStochasticBehavior*» stereotype that allows attributes (e.g., *reliability*) to be attached to hardware components. This stereotype is used for specifying probabilistic information of a hardware failure, e.g., time to the occurrence of a failure, possible failure modes, or the time required to repair the component after the occurrence of a failure, or *ii)* with an *error model*, i.e., a state machine diagram in which a more detailed failure behavior, in terms of hardware *faults*, software *bugs* (*faults*, *errors*, *flaws*) introduced by developers in the design, development, or operation leading it to produce an incorrect or unexpected result, *failure modes*, and internal propagations, of a component can be specified. This enriched model is transformed into a stochastic state-based model, using a variant of the Stochastic Petri Nets (SPNs) formalism. The model is then analyzed to evaluate the satisfaction of system dependability properties (e.g., *availability*) in the form of probabilistic metrics.

Fig. 2 shows an example of a CHESSML error state machine diagram for the Electronic Pedal component of an automotive wheel braking system further used in the evaluation of our approach. This state-machine expresses the effects of *external faults* and their *propagations* as *internal error states*, and *output failure modes*. It comprises *healthy* and *undetected* states, with two *state transitions*. The first state transition moves from the *healthy* to an *undetected error state* due to the occurrence of an *omission* failure in the component input port (*InternalPropagation*) caused by an *external fault* in the *output port* of an *external component* or by a *cyber-attack*, e.g., *denial of service*. The second state transition is activated after the propagation of the *omission* external failure throughout the Electronic Pedal output port. A state machine may also contain *choice*, *fork* and/or *join* nodes for expressing logical relationships between states.

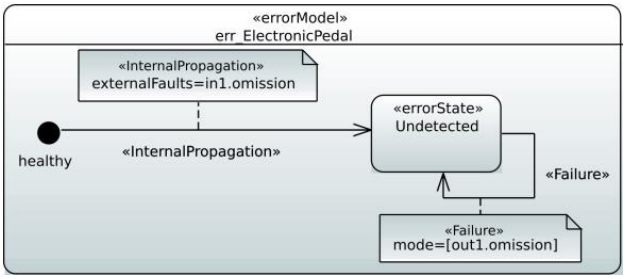

Fig. 2. Example of CHESS-SBA error model state machine diagram.

Performing dependability analysis using CHESS requires the specification of analysis contexts to collect information about the given analysis to be executed, e.g., fault tree analysis («*GaAnalysisContext*»), state-based analysis *(«StateBasedAnalysis»*). A UML component tagged with *GaAnalysisContext* stereotype is used to refer to the set of system entities, by setting the stereotype "context" property with a logical expression describing the combination of component failure leading to the occurrence of a system hazard, to be considered for fault tree analysis using xSAP [31] tool. Fault tree analysis results describe the propagation of component failures specified within the context. A UML component tagged with *StateBasedAnalysis* is used to set the parameters:

measure (the targeting property, e.g., reliability), the *targeting failure modes* (e.g., omission), dependable *components* and their *ports*, required to calculate the probability of components failing at a certain time or within a time range. In this paper, we used the integration between CHESS-SBA and BVR within the AMASS[5] platform for mapping error states and transitions from a state machine diagram to domain variability abstractions (*VSpecs*) in the *Variability Realization* model.

4 A State-Based Dependable Software Product Line

We introduce the State-Based DEPendable Software Product Lines (DEPendable-SPL), which extends the traditional variability analysis [9][25] techniques to cover system safety analysis, i.e., hazard analysis and risk assessment and component fault modeling, using state-machine diagrams. The approach supports the identification of hazards, error states and transitions, associated stochastic annotations, and safety requirements that hold for all product variants. We adopted the concept of 150% state-machine(s) for variability modeling in safety analysis for reconfigurable systems. We also adopted a negative variability strategy for mapping variability abstractions to system design and error state machine elements in the domain engineering phase, and for variability resolution in application engineering phase. The concept of a 150% model relates to a superset approach where 100% configuration model(s) for product safety analysis is/are obtained, via selection and resolution of variation points, from 150% model(s). In our case, the 150% models are CHESSML analysis context model for hazard and state-based analyses, and state-machines that describe the failure behaviors of components.

The 150% state-machine diagrams for product line safety analysis are produced in the domain engineering phase, and 100% models are then derived, via negative variability with the support of BVR tool in the application engineering phase. In a negative variability strategy, safety information elements not needed for a specific system variant are removed from 150% domain model (s). Our approach focuses on variability management on hazardous events, error states, and state transitions from CHESS state-machines in the domain engineering phase. In the application engineering phase, we focus on the reuse of domain safety artefacts and automatic synthesis of fault trees and FMEA from the reused state-machine diagrams. In our approach, we established a clear distinction between reusable safety artefacts, i.e., state machine diagrams, from those that can be generated. The State-Based DEPendable-SPL encompasses four activities in the domain engineering phase and three activities in the application engineering phase (Fig. 3). We describe each approach activity though this section.

4.1 Domain Engineering Phase

The domain engineering phase encompasses: domain analysis, product line design with safety analysis (Hazard Analysis and Risk Assessment and component fault

[5]https://www.amass-ecsel.eu/content/about

modeling), and mapping variability abstractions to CHESSML models, error states and transitions from state machine diagrams.

Domain Analysis: Here, we identify system features and their relationships with safety-related features (i.e., features that impact on safety analysis). Firstly, we identify common and variable system features, followed by the identification of features that impact on safety analysis. Safety-related features refer to characteristics of the operating environment (where) and how a SPL product is used. Finally, we specify the interactions among system and safety-related features via constraints, e.g., implication, exclusion, in the domain feature model. The product line feature model with system and safety-related features and their interactions, in our case, the BVR *VSpec* model (Fig. 3a), is the output of this activity. Interactions among system and safety-related features act as key driver to product line design [32] and safety analysis, taking a direct impact on design decisions, emergent hazardous events and safety requirements to avoid unreasonable risks and achieving compliance with standards.

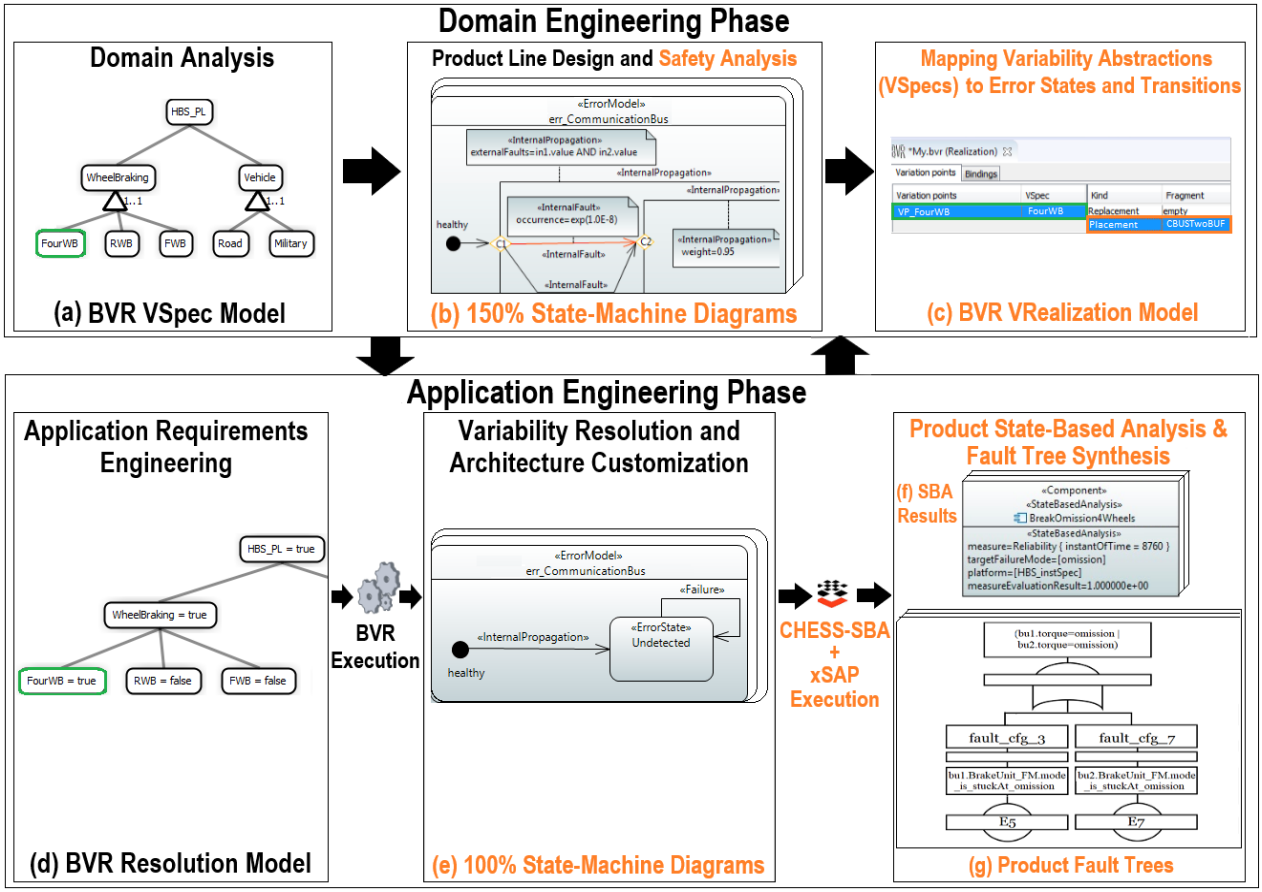

Fig. 3. An overview of state-based DEPendable-SPL.

Product Line Design with Safety Analysis. In this activity, we specify the realization of the variation points and their variants, defined in the SPL feature model, in the architecture using CHESSML Block Definition and Internal Block Diagrams, and we perform safety analysis using CHESS state-machine diagrams. Firstly, we specify the subsystems, components, their ports and connections that represent the realization/materialization of system and safety-related features in the architecture. After performing the preliminary product line design, we start safety analysis (HARA) considering the feature interactions specified in the feature model representing the domain of the targeted reconfigurable system, and the architecture model. During safety analysis, we identify the potential hazards and failures that can emerge in architectural subsystems and components associated with different feature interactions. Product Line Safety Analysis encompasses the following sub-activities: *Identification of Feature Interactions that may Impact on Product Line Safety Analysis*, *Hazard Analysis and Risk Assessment*, and *State-Based Component Fault Modeling*.

Identification of Feature Interactions that Impact on Product Line Safety Analysis: it encompasses: ***i)*** identify the combinations among system features, which conform to the product line feature model and their relationships, with elements from architectural and behavioural models. ***ii)*** For each identified interaction between system features, we analyse combinations among safety-related features. ***iii)*** Finally, we combine the identified system feature interactions with safety-related feature interactions to derive a set of combinations among system and safety-related feature interactions relevant for the stakeholders. We introduced this activity in our approach since it would be prohibitive performing domain safety analysis considering all possible interactions among system and safety-related features expressed in the SPL feature model. Although we consider feature interactions in our approach, it is important to highlight that we focus on performing safety analysis from the perspective of the whole SPL domain instead specific configurations.

Hazard Analysis and Risk Assessment: This activity encompasses the following steps: ***i)*** choosing a specific feature interaction to be considered in the analysis; ***ii)*** identifying combinations among component error states that may lead to the occurrence of hazards; ***iii)*** specify the context of each hazard by combining component failure modes leading to the occurrence of a hazard in expressions using logical operators, e.g., *Omission-C1.out1 and Omission-C2.out1*, in a UML component tagged with *«GaAnalysisContext»* stereotype; and *iv)* specify the targeted properties (e.g., *severity*), *failure modes* and *components* associated with each identified hazard in «StateBasedAnalysis» tagged components. The output of this activity is a 150% CHESS Analysis Context model (CAC).

State-based Component Fault Modeling: We specify the failure behavior of each product line architectural subsystem/component in a separated 150% CHESS state-machine diagram by describing: ***i)*** the potential output deviations (failures) that may contribute to the occurrence of hazards in each feature interaction under analysis. Each *output deviation* should be modeled as an *error state* that describes the component failure behavior; and *ii)* the potential *causes*, in terms of *input failures*, *internal faults* or combinations among them, which lead to the occurrence of each identified *output deviation*. We can also assign *probability* information to error states and state

transitions. We apply these steps to analyze the components based on the assumptions and the data associated with each feature interaction under analysis.

The outputs of this activity are 150% state-machine diagrams that describe how the components may fail and contributing to the occurrence of hazards in each analyzed feature interaction. Since the causes of a component failure may change according to design choices and feature interactions, we recommend specify the commonalities and variability inherent to the failure behavior of each component in a separated 150% state machine diagram (see Fig. 3b).

Mapping Variability Abstractions to Error States and Transitions: In this activity, we specify mappings linking variability abstractions (Fig. 3a) to their materialization into CHESSML analysis context and state machine model elements (Fig. 3b). This is needed for integrating safety analysis information from state machines within the SPL core assets, and enabling the systematic reuse of safety information in the application engineering phase. We adopted a negative variability strategy for mapping features to their realization into fragments of superset CHESSML models and state-machine diagrams into the BVR variability realization model (Fig. 3c). Since we followed a negative variability strategy, we specify the realization of variability abstractions into BVR fragment substitutions containing placement fragments with references to model elements that should be removed from superset CHESSML models and state machine diagrams when a given feature is selected. We specify the mappings of features to their materialization into: ***i)*** the 150% CHESS block diagrams, ***ii)*** analysis context model for hazard analysis and ***iii)*** component state machine diagrams via fragment substitutions with a placement and an empty replacement fragment. An enriched BVR *VRealization* model with the specification of mappings linking variability abstractions to their materialization into: CHESSML models and error state-machine diagrams is the output of this activity. The *VRealization* model enables the automatic derivation of 100% CHESSML models and state-machine diagrams in the application engineering phase.

4.2 Application Engineering Phase

Here we describe the application requirements engineering, variability resolution and architecture customization, product state-based and fault tree synthesis activities.

Application Requirements Engineering: We specify application-specific requirements in a feature model, via selection of features specified in the product line feature model (Fig. 3a), in the resolution model (Fig. 3d) using BVR resolution editor. The resolution model with the selected domain features that address the application requirements is the output of this activity. The resolution model is the input for variability resolution and architecture customization.

Variability Resolution and Architecture Customization: In this activity, we resolve the variability expressed in 150% CHESS-ML models and state machine diagrams to derive 100% models (Fig. 3e) according to the feature selection specified in the resolution model. We perform this activity with the support of BVR execution engine (see Section 3.1). Here, we execute the BVR engine for product derivation by providing the following input artefacts: the *VSpec*, *VResolution* and *VRealization*

models, the 150% CHESS-ML models and state machine diagrams. The 100% CHESS-ML architecture and analysis context models, and state machine diagrams that describe the failure behaviors of components in the feature interactions specified in the resolution model are the outputs of this activity.

It is important to highlight that in cases where application-specific features not provided by the SPL are specified during application requirements engineering; engineers can update the derived 100% CHESSML system model with the addition of subsystems and components that address application-specific features. Different assumptions for product safety analysis can emerge from the added product-specific components to the reused 100% CHESSML system models and interactions among product-specific features. The interactions between existing and product-specific features (system and safety features) should be considered during product safety analysis. In this case, engineers perform product safety analysis following the same steps defined in the domain engineering phase.

Product-specific components, analysis context and state machines may provide feedback to the SPL development process. To achieve this goal, it is needed enhancing the product line *VSpec* model, the 150% CHESSML models for design and safety analysis, and the *VRealization* model with additional fragment substitutions. This is required to enable the reuse of product-specific components and their associated safety analysis information in the development and certification of other safety-critical products. The feedback to the product line development process in the application development is supported in our approach via CHESS and BVR integration. Such integration allows users updating the *VSpec* model with application-specific features, and the *VRealization* model with fragment substitutions for mapping these features to CHESS-ML and state-machine elements added to the SPL repository.

Product State-Based Analysis and Fault Trees Synthesis: In this activity, we perform state-based analysis, via execution of CHESS-SBA, to estimate probabilistic properties (e.g., *severity*) associated with hazardous events (Fig. 3f), and synthesis of fault trees (Fig. 3g), with the support of xSAP [31] tool. The 100% CHESSML analysis context models, components and state machines are inputs for executing CHESS state-based analysis and xSAP for synthesizing fault trees for each product-specific hazard. The outputs of this activity are: state-based analysis results that provide metrics associated with hazardous events, fault trees for each identified hazardous event, and a FMEA table. The fault trees are further synthesized into FMEA that describe how component can fail and contributing to the occurrence of hazardous events. State-based analysis results support risk assessment, the assignment of Safety Integrity levels (SILs) to mitigate hazard effects, and derivation of safety goals. Fault trees and FMEA are required by standards for certifying a safety-critical system.

5 Evaluation

We evaluated our approach in an automotive Hybrid Braking System (HBS) [33]. The complete HBS models described through this section are available in [34].

5.1 Hybrid Braking System

The HBS is an automotive wheel braking system (Fig. 4), originally designed in MATLAB/Simulink. The term hybrid means that the braking occurs through the combined action of electrical In-Wheel Motors (IWMs), and frictional Electromechanical Brakes (EMBs) within Brake Units (BUs). While braking, IWM components transform the vehicle kinetic energy into electricity, charging the power train battery, increasing the vehicle's range. The HBS architecture comprises 4 variant wheel-brake units (subsystems), 30 components with 69 connections. Each wheel brake module comprises a Wheel Node Controller (WNC), for calculating the amount of braking torque to be produced by each wheel braking actuator, and it sends commands to Electromechanical Braking (EMB) and IWM power converters that control EMB and IWM braking actuators. While braking, the electric power flows from the Auxiliary Battery to the EMB via EMB Power Converter; and IWM acts as a power generator providing energy for the Powertrain Battery via IWM Power Converter.

The HBS was evolved into a product line. We re-designed the HBS using CHESS-ML to support the evaluation of State-based DEPendable-SPL approach. The wheel braking is the HBS architectural variation point (see Fig. 3a). We can combine the four wheel-brake units into different ways to derive different variants. In this paper, we considered two HBS product variants: four-wheel braking (FourWB) and front-wheel braking (FWB). The front-wheel brake units and their connections to other components (Fig. 4) represent the realization of the FWB product. The HBS product variants can be deployed in a road car or in a military vehicle. Different hazards with different risks can rise from interactions between components in each HBS variant and operating environment, thus, impacting on safety analysis.

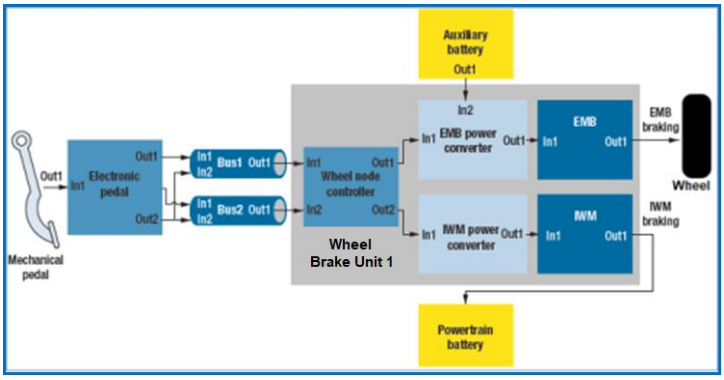

Fig. 4. An excerpt of hybrid braking system product line architecture [35].

5.2 HBS: Domain Engineering Phase

Identification of Feature Interactions for Braking System Safety Analysis: From the analysis of HBS system and safety-related features (Fig. 3a), we identified and considered the following feature interactions during domain safety analysis: **FI1** - **Front Wheel Braking** deployed in a **Road** car vehicle (FWB and Road), and **FI2** -

Four Wheel Braking deployed in a **Military** vehicle (FourWB and Military). **HBS: Hazard Analysis and Risk Assessment.** We performed this step from the analysis of the HBS architecture model (Fig. 4) considering **FI1** and **FI2** feature interactions. We identified four potential hazards (Table 1) that may rise in two particular feature interactions. During HBS hazard analysis we specified a 150% CHESS analysis context model with four «StateBasedAnalysis» components that provide the context for estimating the probability of occurrence, exposure and controllability of *omission* and *value* failure modes in both feature interactions. We also specified four «GaAnalysisContext» components that define the contexts for fault tree analysis.

Table 1. Wheel braking hazards.

Feature Interaction	Hazard	Sev.	Exp.	Contr.	ASIL
FourWB + Military	No braking four wheels.	S3	0.6% (E2)	C2	A
	Value braking	S3	14.9% (E4)	C2	C
FWB + Road	Value braking	S3	0.6% (E2)	C3	B
	No braking front	S3	0.7% (E2)	C3	B

HBS Component Fault Modeling: In our evaluation, we specified the failure behavior of 10 components, considering FI1 and FI2 feature interactions, into 10 state-machines. Fig. 3b shows an excerpt of the 150% CHESS error state machine diagram that describes the failure behavior of Communication Bus component in front-wheel (FI1) and four-wheel (FI2) braking feature interactions. The probability of occurrence of a random failure in this component when connected to front wheel brake units is 1.0E-6 per hour of operation. The Communication Bus components may also fail through the propagation of external faults through its input ports, raising either an omission and/or wrong value failures. Variation in state machines may impact on how hazards propagate throughout architectural components during fault tree analysis, and in the probability of occurrence of hazardous events, changing safety requirements.

Mapping HBS Variability Abstractions to CHESS-ML Models and State-Machines: We performed this step from the analysis of the HBS feature model, the identified feature interactions, CHESS-ML models and state-machine diagrams. The HBS *VRealization* model contains eight fragment substitutions: four to manage variation on CHESS-ML models, two for state machine diagrams, and two to manage variations on CHESS analysis context model. Fig. 3c show an excerpt of the *VRealization* model with the CBusTwoBUFS fragment substitution with a placement referencing a state transition from the 150% Bus state machine (Fig. 3b). The highlighted state transition should be removed from this state-machine when FourWB (Fig. 3b) is chosen.

5.3 HBS: Application Engineering Phase

Braking System Requirements: In this step, we specified the features that address FourWB and FWB requirements into two resolution models. Fig. 3d shows an excerpt of the FourWB resolution model, where the FourWB feature was chosen. **HBS Variability Resolution**: We input the following artefacts to BVR execution engine: FourWB and FWB resolution models, the superset CHESS-ML models and state machine diagrams. Firstly, we executed the BVR engine for deriving 100% CHESS-

ML models for FWB system variant, followed by FourWB (Fig. 3d). **Product State-Based Analysis and Fault Trees Synthesis**: After product derivation, we performed-state-based analysis, via execution of CHESS-SBA, to estimate the severity, probability of exposure and controllability of *omission* and *value* hazardous events in FWB and FourWB system variants. Table 1 illustrates the state-based analysis results for the FWB and FourWB hazards. The results demonstrated that all the identified wheel braking hazards have the potential to produce life threatening injuries to the occupants (S3 severity). With respect to probability of exposure, the probability of occurrence of *value braking* and *no braking* hazards in the FWB system variant is lower (E2). We identified the probability of exposure to a *value braking* hazard is very high (E4) in the FourWB variant. Finally, we estimated the controllability of the driver in hazardous driven situations based on qualitative attributes and fault tree analysis. From the analysis of the values assigned to severity, probability of exposure and controllability, we classified the risk posed by each hazard by checking the corresponding ASIL in the risk matrix. The occurrence of *no braking* hazard is most critical in FWB system variant demanding ASIL B safety goals. On the other hand, ASIL A is sufficient to mitigate the hazard effects in the FourWB. This example demonstrates the impact of variability in the design and operating environment on hazard analysis and risk assessment, derivation of safety goals/requirements. Such variation further propagates throughout fault tree analysis. Due to space limitations, the synthesized fault trees describing the propagation of *no braking* hazard throughout FourWB and FWB system variants are available elsewhere [34].

6 Conclusions

Model-based design and safety assessment become increasingly common in industry and safety standards from different domains, e.g., automotive, avionics, start to adopt them. Model-based languages like CHESSML [20], EAST-ADL [36], and AADL [18] have extensions, annexes, and metamodels that integrate safety concepts into model-based design. These developments are encouraging in terms of adaptation and adoption of the concepts of variant management and reuse in system safety assurance processes proposed in this paper. There are other powerful tools for Model-Based Safety Assessment (MBSA) including ALTARICA [37] [38], FSAP/xSAP [39] [31], and HiP-HOPS [24]. It is beyond the scope of this paper to discuss the relation of our work to these approaches. Our approach is generally complementary to other work in MBSA, and, some of the concepts we propose for MBSA of product lines could inspire other approaches. We introduced the State-Based DEPendable-SPL approach to support variability modeling and management in safety analysis using state-machine diagrams. The difference from our approach in comparison with related work [5][7][10][13] is the focus on establishing a clear distinction between reusable safety assets, i.e., state-based and failure logic models, from those that can be generated, e.g., FTAs. Our approach supports mapping variability abstractions to elements from state-machine diagrams in the domain engineering phase, and it enables the systematic reuse of these models, the execution of state-based analysis, and synthesis of fault

trees in the application engineering phase. The main benefits of our approach for safety assurance reuse are: *i)* the improvement of the verification product line safety properties across different scenarios; *ii)* the traceability between variability abstractions, and error states and transitions from state-machine diagrams; and *iii)* variability resolution in error states and transitions; and *iv)* the reduction of the burden of safety analysis for certifying individual system variants since this task is not performed from scratch. It may contribute to reducing the effort to perform hazard analysis and risk assessment, fault tree analysis, and FMEA required by standards like ISO 26262 for certifying a specific system variant. Our approach is applicable to other standards and domains (e.g., avionics) to enable safety assets reuse through certification processes.

In our experience, we achieved between 65-80% of reuse in state machine diagrams in the FourWB and FWB variants respectively. Our approach also contributed to reduce the effort on extending the SPL with newer components and state-machine diagrams via specification of newer features and fragment substitutions. Also, the State-Based DEPendable-SPL automates the traceability between variability abstractions (VSpec features) and safety analysis artefacts. As future work, we plan to extend our approach to support the management of the diversity of dependability information in safety and cyber-security assessment, and runtime variability in assurance cases. We also intend to assess the effectiveness of our approach in more complex industrial product lines from autonomous systems and other safety-critical domains.

References

1. Wolschke, C., Becker, M., Schneickert, S., Adler, R., and MacGregor, J. 2019. Industrial Perspective on Reuse of Safety Artifacts in Software Product Lines. In Proc. of the 23rd International Systems and Software Product Line Conference (SPLC '19), Paris, France. ACM, NY, USA, pp. 143-152.
2. Pohl, P., Höchsmann, M., Wohlgemuth, P., Tischer, C. 2018. Variant management solution for large scale software product lines. In Proc. of the 40th International Conference on Software Engineering: Software Engineering in Practice, Gothenburg, Sweden. ACM, New York, NY, USA, pp. 85-94.
3. Tischer, C., Muller, A., Mandl, T., Krause, R. 2011. Experiences from a Large Scale Software Product Line Merger in the Automotive Domain. In Proc. of the 15th Int. Software Product Line Conference, Munich, Germany, pp. 267-276.
4. SPLC.net, 2019. SPLC hall of the fame: General Motors Powertrain (GMPW). Online: https://splc.net/fame/general-motors-powertrain, last accessed 2022/07/10.
5. Schulze, M., Mauersberger, J., Beuche, D. 2013. Functional safety and variability: can it be brought together? In Proc. of the 17th International Software Product Line Conference, Tokyo, Japan. ACM, NY, USA, pp. 236-243.
6. Oliveira, A. L. de, Braga, R. T. V., Masiero, P. C., Papadopoulos, Y., Habli, I., Kelly, T. 2018. Variability Management in Safety-Critical Software Product Line Engineering. In R. Capilla, C. Cetina, and B. Gallina (Eds.), Int. Conf. on Soft. Reuse, LNCS 10826, 1–20.
7. Domis, D., Adler, R., Becker, M. 2015. Integrating variability and safetyanalysis models using commercial UML-based tools. In Proc. of the 19th International Software Product Conference, 20-24-July, Nashville, USA. ACM, NY, USA, pp. 225–234.

8. Oliveira, A. L. de, Braga, R. T. V., Masiero, P. C., Parker, D., Papadopoulos, Y., Habli, I., Kelly, T. 2019. Variability management in safety-critical systems design and dependability analysis. Journal of Software: Evolution and Process v. 31 n. (8), 2019, pp. 1-28.
9. Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns. Addison-Wesley.
10. Dehlinger, J., Lutz, R. Software fault tree analysis for product lines, 2004. In Proceedings of the 8th IEEE Int. Symp. on High Assurance Sys. Eng., Tampa, USA, pp. 12-21.
11. Feng, Q., Lutz, R. Bi-directional safety analysis of product lines. Journal of Systems and Software. 2005, 78 (2), pp. 111-117.
12. Gomez, C., Peter, L., Sutor, A. 2010. Variability Management of Safety and Reliability Models: An Intermediate Model towards Systematic Reuse of Component Fault Trees. In: Schoitsch E. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2010. LNCS, vol 6351. Springer, Berlin, Heidelberg, pp. 28-40.
13. Liu, J., Dehlinger, J., Lutz, R. 2007. Safety analysis of software product lines using stated modeling. J. of Systems and Software. 2007, 80 (11), pp. 1879-1892.
14. Käßmeyer, M., Schulze, M., Schurius, M. 2015. A process to support asystematic change impact analysis of variability and safety in automotive functions. In Proc. of the 19th Int. Software Product Line Conf., Nashville, USA. ACM, NY, USA, pp. 235–244.
15. Käßmeyer, M., Moncada, D. S. V., Schurius, M. 2015. Evaluation of asystematic approach in variant management for safety-critical systemsdevelopment. In Proc. of 13th Int. Conf. on Embedded and Ubiquitous Comp., IEEE, Porto, Portugal, pp. 35–43.
16. Montecchi, L., Lollini, P., Bondavalli, A. A Template-Based Methodology for the Specification and Automated Composition of Performability Models. In IEEE Transactions on Reliability, vol. 69, no. 1, pp. 293-309, 2020.
17. Bressan, L., Oliveira, A. L. de, Campos, F., Papadopoulos, Y., Parker, D. An Integrated Approach to Support the Process-Based Certification of Variant-Intensive Systems. In: Proceedings of the 7th Int. Symp on Model-Based Safety and Assessment, Lisbon, Portugal, 2020, Springer-Verlag, Berlin, Heidelberg, pp. 179–193.
18. SAE. 2017. Architecture Analysis & Design Language (AADL) AS5506C, SAE. Online: https://www.sae.org/standards/content/as5506c/
19. Shiraishi, S. 2010. An AADL-based approach to variability modeling of automotive control systems. MODELS, LCNC, v. 6393. Springer, pp, 346-360.
20. Intecs, CHESS Modelling Language: A UML/MARTE/SysML profile. April, 2020. Online: https://www.eclipse.org/chess/publis/CHESSMLprofile.pdf.
21. Mazzini, S., Favaro, J., Puri, S., Baracchi, L. CHESS: an open source methodology and toolset for the development of critical systems. In Join Proc. of EduSymp, 2016, pp. 59-66.
22. Gallina, B., Javed, A. M., Muram, F. U., Punnekkat, S. Model-driven Dependability Analysis Method for Component-based Architectures. In: Proceedings of the Euromicro-SEAA Conference, Cesme, Izmir, Turkey, 2012, pp. 233-240.
23. Delange, J., Feiler, P., Gluch, D., Hudak, J. AADL fault modeling and analysis within an ARP4761 safety assessment. Tech. rep., Carnegie Mellon Software Eng. Inst., 2013.
24. Papadopoulos Y., Walker M., Parker D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U., Lien, R. Engineering failure analysis and design optimization with HiP-HOPS. Journal of Engineering Failure Analysis, Elsevier. 18 (2), 590-608, 2011.
25. Capilla, R., Bosch, J., Kang, K. C. 2013. Systems and Software Variability Management: Concepts, Tools and Experiences. Springer Publishing Company.
26. Vasilevskiy, A. Haugen, Ø., Chauvel, F., Johansen, M. F., Shimbara, D. 2015 The BVR tool bundle to support product line engineering. In Proceedings of the 19th Int. Software Product Line Conf., Nashville, USA, ACM, NY, pp. 380–384.

27. Haugen, Ø., Moller-Pedersen, B., Oldevik, J., Olsen, G. K., Svendsen, A. 2008. Adding standardized variability to domain specific languages. In: Proceedings of the 12th Int. Software Product Line Conf., IEEE, pp. 139–148.
28. ISO, 2018. ISO 26262: Road Vehicles Functional Safety.
29. Thomas, E. 2009. Certification Cost Estimates for Future Communication Radio Platforms. Rockwell Collins Inc., Tech. Rep.
30. Montecchi, L., Gallina, B. SafeConcert: a Metamodel for a Concerted SafetyModeling of Socio-Technical Systems, in: 5th International Symposium onModel-Based Safety and Assessment, vol. 10437 of LNCS, Trento, Italy, 2017, pp. 129–144.
31. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A, Mattarei, C., Micheli, A., Zampedri, G. 2016. The xSAP Safety Analysis Platform. In Proc. of TACAS.
32. Lee, K. and Kang, C. 2010. Usage Context as Key Driver for Feature Selection. In Bosch J., Lee J. (eds) Software Product Lines: Going Beyond. 14th SPLC 2010, Lecture Notes in Computer Science, vol. 6287. Springer, Berlin, Heidelberg, 32-46.
33. De Castro, R., Araújo, R. E., Freitas, D. 2011. Hybrid ABS with Electric motor and friction Brakes. In Proc. of the 22nd Int. Symposium on Dynamics of Vehicles on Roads and Tracks, Manchester, UK.
34. HBS, Case Study, https://github.com/aloliveira/hbs
35. Azevedo, L., Parker, D., Walker, M., Papadopoulos, Y., Araújo, R. 2014. Assisted Assignment of Automotive Safety Requirements. IEEE Softw. 31, 1, pp. 62–68.
36. Blom, H., Chen, D., Kaijser, H., Lönn, H. Papadopoulos, Y., Reiser, M., Kolagari, R. T., Tucci, S. 2016. EAST-ADL: An Architecture Description Language for Automotive Software-intensive Systems in the Light of Recent use and Research. International Journal of System Dynamics Applications (IJSDA), 5(3), pp. 1-20.
37. AltaRica Project. 2020. Methods and Tools for AltaRica Language. Online: https://altarica.labri.fr/wp/?page_id=23.
38. Arnold, A., Gerald, P., Griffault, A., Rauzy, A. 2000. The Altarica Formalism for Describing Concurrent Systems. Fundamenta Informaticae, 34, 109-124.
39. Bozzano, M. & Villafiorita, A. 2006. The FSAP/NuSMV-SA Safety Analysis Platform. International Journal on Software Tools for Technology Transfers (STTT) – Special Section on Advances in Automated Verification of Critical Systems, 9(1), 5-24.