A Systematic Process for Applying the CHESS Methodology in the Creation of Certifiable Evidence

Lucas Bressan, André L. de Oliveira
Federal University of Juiz de Fora, Brazil
{lucasbressan, andre.oliveira}@ice.ufjf.br

Leonardo Montecchi
University of Campinas, Brazil
leonardo@ic.unicamp.br

Barbara Gallina
Mälardalen University, Sweden
barbara.gallina@mdh.se

***Abstract*—CHESS is an open source methodology and toolset for the development of safety-critical systems. More specifically, CHESS is a model-based methodology, which supports the design, dependability analysis, and code generation for critical systems. Despite its rather mature level in terms of technology readiness, systematic guidance needs to be developed to promote its usage for certification purposes. In this paper, we present a systematic process to guide designers and analysts in the usage of the CHESS toolset for model-based dependability analysis of safety-critical systems in compliance with ISO 26262 Parts 3 and 4, SAE ARP 4754A safety process, and DO-331 model-based development principles. We also have applied our process to a real world automotive hybrid braking system. The proposed process can be used to guide analysts in using CHESS methodology to support both system design and dependability analysis. Finally, we draw our conclusion and sketch future work.**

Keywords—Dependability analysis, model-based development, process, certification, CHESS, safety standards.

I. INTRODUCTION

Model-based development (MBD) has been contributing to raising the level of abstraction in software specification and to increasing automation in software development. Industry and safety certification standards from different domains, e.g., DO-178C and its MBD supplement DO-331 [23], and SAE ARP 4754A [11] for avionics, and ISO 26262 [15] for automotive, have recognized the maturity of model-based techniques, which are being increasingly adopted by the industry to provide semi-automated support for both system design and dependability analysis.

Qualitative and quantitative compositional model-based techniques for system design and dependability analysis exist in the literature [1][4][7][27]. However, safety-critical systems require the integrated application of different techniques, and an incremental modeling approach that can follow the evolution of the system. CHESS is an open source, integrated and multifaceted model-based methodology and toolset for the development of safety critical systems, which supports system design, dependability analysis, and code generation [16]. The CHESS methodology supports system architects to interpret human, organizational, and technological entities in terms of components, and modeling their behavior with respect to safety/dependability, i.e., erroneous and fault-tolerance behaviors [17]. CHESS supports the interplay among different dependability analysis techniques, namely failure propagation logic, and state-based stochastic analysis.

Despite its rather mature level in terms of technology readiness, its usage in real-life systems has been limited to industrial partners of the CHESS [2] and CONCERTO [3] projects. Systematic guidance to support the proper usage of the framework for certification purposes is missing. Actually, an aspect that was highlighted by CHESS project evaluation, by submitting questionnaire to experts [8], was a moderate belief that the provided analysis techniques could support engineers in the safety certification process. This is due to the lack of guidance for external users adopting the CHESS methodology for producing certification evidence in compliance with existing safety standards.

State of the practice in the assessment of critical systems adopting model-based techniques comprises proposals of MBD toolsets [6][10] to address system design, automatic code and documentation generation, verification and validation, and model/requirements traceability in compliance with the aforementioned standards. However, such MBD toolsets do not provide support for integrated system design and dependability analysis, not addressing ISO 26262 Part 3 – *Concept Phase* and Part 4 – *Product development at the system level*, and SAE ARP 4754A development and safety processes, which is required to produce certification evidence. We propose to fill this gap by augmenting the CHESS methodology with a systematic process that supports users at producing safety-related certifiable evidence in compliance with standards, thus, bridging the gap between standards, industrial practices, and academia, guiding analysts in the properly usage of the CHESS to generate certifiable evidence.

The main contributions of this paper are: *i)* a systematic process to guide analysts in using CHESS model-based methodology in dependability analysis of safety-critical systems to obtain certifiable evidence in compliance with ISO 26262, SAE ARP 4754A, and DO-331 MBD principles, *ii)* the application of the process in a real world automotive hybrid braking system case study, and *iii)* contextualization of the proposed process with respect to the ISO 26262 safety certification processes.

The rest of this paper is organized as follows. Section II presents an overview of the CHESS framework. Section III presents the proposed systematic process. Section IV presents a case study illustrating the application of the proposed process in an automotive Hybrid Braking System (HBS), while in Section V we discuss the mapping with the ISO 26262 standard. Section VI discusses the related work. Finally, conclusions are drawn in Section VII.

II. THE CHESS FRAMEWORK

CHESS is a model-driven, component-based, methodology and toolset for the development of high-integrity systems for different domains. The methodology has a strong focus on the specification and analysis of non-functional properties, especially predictability and dependability, and the generation of code preserving such properties. The CHESS methodology consists of a UML-based modeling language, named CHESS-ML [16], and a set of plugins to support code generation, constraints checking, and different kinds of analyses.

In the CHESS methodology, functional and extra-functional properties are addressed using dedicated views, which each view have different fixed privileges on model entities and properties that can be manipulated. The CHESS methodology uses an incremental and iterative process where components can be defined in an incremental way using repositories of components or via composability. Results of different analyses are *back-annotated* into the model, allowing engineers to perform an iterative development process.

Modeling is organized in a set of separated views. Each design view applies specific constraints on UML diagrams and entities that can be created, displayed or edited in that view [16]. The *requirement* view is used to model requirements by using the standard requirement diagram from SysML. The *system* and *component* views are respectively used to model system-level entities and software components with SysML [16]. The *component* view comprises two sub-views, the *functional* view and the *extra-functional* view. The *deployment* view is used to describe the hardware platform where the software runs (i.e. CPUs, buses), and software to hardware allocation. Finally, the *analysis* view is used to provide information to the different analysis techniques, also called analysis context. CHESS supports analysis techniques for real-time and dependability properties. In this paper, we solely focus on dependability analysis.

The CHESS methodology provides two plugins to perform dependability/safety analysis, namely CHESS-FLA and CHESS-SBA. CHESS-FLA [13] allows users, i.e., system architects and engineers, to decorate component-based architectural models, specified using CHESS-ML, with dependability information, execute Failure Logic Analysis (FLA), and get the results back-propagated onto the original system model. The CHESS *State-Based Analysis* (CHESS-SBA) plugin [18] allows users to perform quantitative dependability analysis on system models, specified using CHESS-ML, by enriching them with quantitative (i.e., probabilistic) dependability information, including failure and repair distribution of components, propagations delays and probabilities, and fault-tolerance and maintenance concepts.

The CHESS methodology is implemented by the CHESS framework, a collection of Eclipse plugins, released as open source under the PolarSys initiative [22]. The latest version of the CHESS framework allows both CHESS-FLA and CHESS-SBA plugins to operate together on a consistent set of UML stereotypes and share some pieces of information [17]. Still, to the best of our knowledge, the combined application of CHESS-FLA and CHESS-SBA techniques on a real use-case have not been experimented on real-life systems. One of the reasons, as highlighted by questionnaires submitted to experts [8], appears to be that the role of CHESS with respect to certification is not completely clear to the external community.

In the following, we present an integrated process for the application of dependability analysis using the CHESS to support the production of standard-compliant certification evidence and its application in a realist automotive braking system (Section IV), and contextualize the proposed process with respect to some recent safety standards. We believe this contribution can help in the diffusion of the CHESS, and possibly its extension with the definition of a systematic process, being it an open source toolset.

III. THE PROPOSED PROCESS

The proposed process was defined in compliance with the DO-331 MBD fundamentals/principles [23]: *i) "identifying the safe-subset use of MBD technology and suitable graphical engineering methods to be used in safety-related applications"* which is addressed by CHESS-ML constraints, by the fact that we can only use a specific subset of UML, and by CHESS having a separate dependability analysis view (failure logic and state-based analyses steps in Figs. 1b and 1c); *ii)* "*clear distinction between design and specification models*": it can be addressed since both the proposed process and CHESS comprise the specification of a high level system model (in a SysML *Block Definition Diagram*), and a detailed CHESS-ML design model (Fig. 1a), and by the integration between system design/dependability analysis via system and dependability views; *iii)* "*determining which artefacts will be in a model to drive the determination of applicable objectives and activities*": in CHESS, detailed architecture, data and control flow and implementation form the content of a *SysML Internal Block diagram*, which corresponds to the *Software Design Document*. Thus, the proposed process and CHESS can address this fundamental by supporting model traceability and verification; *iv)* "*MBD data items to be expected in a program-model planning, model standards, and model element libraries":* this fundamental can be addressed in CHESS via system design activities supported by CHESS-ML language for system specification, design, and dependability analysis; and finally *v)* "*MBD data items to be expected in a program-model coverage and model simulation"* fundamental can be addressed by the proposed process due CHESS methodology enabling support for back failure propagation analysis via failure logic and state-based analyses.

The proposed process, given in SPEM 2.0 and illustrated in Fig. 1, provides systematic guidance to produce standard compliant certification evidence using the CHESS methodology. This process prescribes a set of steps to guide engineers at performing system design using CHESS-ML *Block Definition Diagram* and *Internal Block Diagram*, component instance generation (Fig. 1a), and dependability analysis using CHESS-FLA (Fig. 1b) and CHESS-SBA (Fig. 1c). CHESS-FLA supports engineers at specifying qualitative behaviors of individual components in terms of component

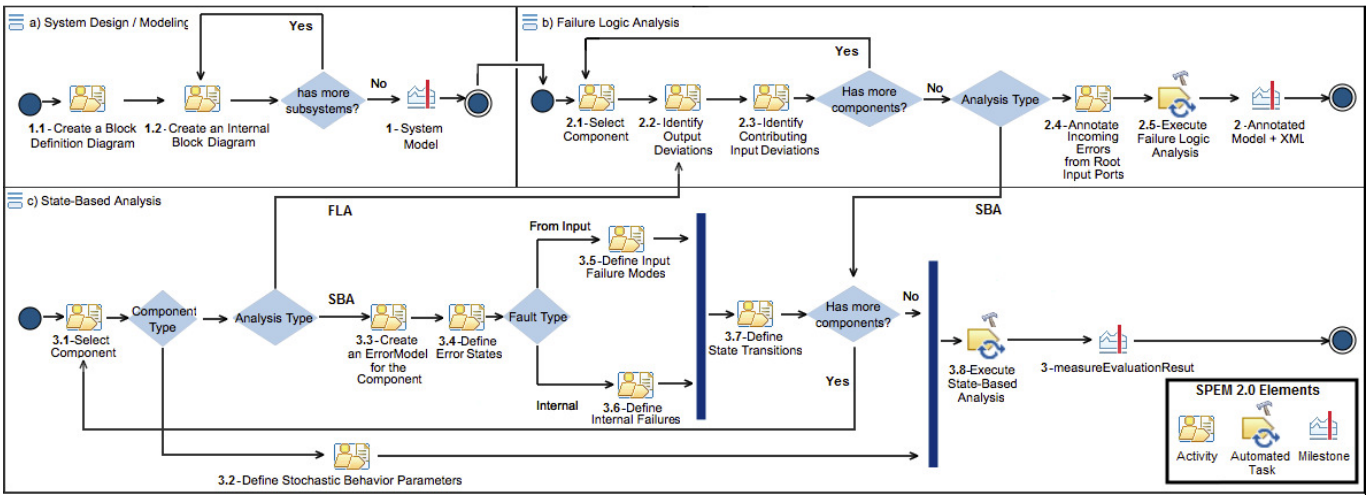

Fig. 1. The proposed CHESS model-based design and dependability analysis process.

failures and their causes, and partially automates FTA and FMEA synthesis back-propagated onto the original system model. CHESS-SBA allows engineers at specifying more expressive and detailed fault behavior of individual components and supports quantitative dependability analysis. *Execute failure logic analysis* and *execute state-based analysis* are fully automated tasks supported by CHESS toolset.

The proposed process, illustrated in Fig. 1, was built upon DO-331 principles [23], avionics SAE ARP 4754A [11] and automotive ISO 26262 [15] development and safety processes. A detailed mapping with ISO 26262 concepts and work products is provided in Section V. In the following, the individual steps of the process are illustrated in details, by applying them to a realistic automotive Hybrid Braking System (HBS) case study.

IV. The Hybrid Brake System (HBS) Case Study

HBS is a real world automotive braking system originally designed in MATLAB/Simulink. HBS is meant for integration in electrical vehicles, in particular for propulsion architectures that integrate one electrical motor per wheel [9]. The term hybrid comes from the fact that braking is achieved throughout the combined action of the electrical *In-Wheel Motors* (IWMs), and the frictional *Electromechanical Brakes* (EMBs). One of the most important features of this system is that the integration of IWM in the braking process allows an increase in the vehicle's range. Thus, while braking, IWMs work as generators and transform the vehicle kinetic energy into electrical energy that is fed into the *Powertrain Battery*. HBS should not raise omission of braking torque or incorrect value of braking torque failures in the wheel's while braking, since the occurrence of such hazardous events can lead to catastrophic consequences for the driver.

A. System Design/Modeling

The system modeling workflow, illustrated in Figure 1a, consists in the system definition which comprises the specification of subsystems, components and their connections, using *SysML Block Definition Diagram*, and the detailed description of each constituent subsystem, using *SysML Internal Block Diagram*.

1) System Definition. After creating a new CHESS project, and a *Papyrus UML* model, a new *Block Definition Diagram* should be created in the *«SystemView»* for specifying the system architecture. In this view, the system, its subsystems and components, their input, output ports, and connections can be specified, each one as a distinct *SysML «Block»* element. This constitutes the "inventory" of component types available for the modeling process.

The automotive HBS system, discussed through this paper comprises ten components/subsystems: four *Brake Unit* subsystems (i.e., one per wheel); one *Mechanical Pedal,* which is a hardware device aimed at capturing driver presses; one *Electronic Pedal* device that senses and processes the actions triggered by the *Mechanical Pedal*; two communication buses (*Bus1* and *Bus2*) that send wheel braking forces to the wheel *Brake Units*; one *Auxiliary Battery* device responsible for feeding the electromechanical brakes while braking; and a *Powertrain Battery*, a device that receives the electrical energy produced by the *In-Wheel Motors*. Fig. 2 illustrates an excerpt of the HBS *block definition diagram.* The HBS block represents the braking system which comprises four wheel brake units, mechanical and electronic pedals, two communication buses, one auxiliary battery, and one powertrain battery.

2) Subsystem Definition. Once the system and its sub components have been specified in a *block definition diagram*, for each composite system/subsystem, a new *Internal Block Diagram* should be created. Each of those diagrams defines the internal implementation of a subsystem, in terms of instances of components defined in the previous step.

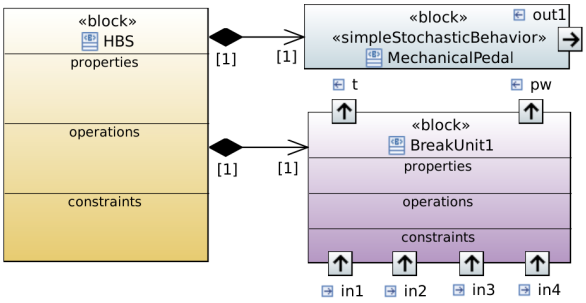

Fig. 2. Excerpt of the HBS block definition diagram in CHESS-ML.

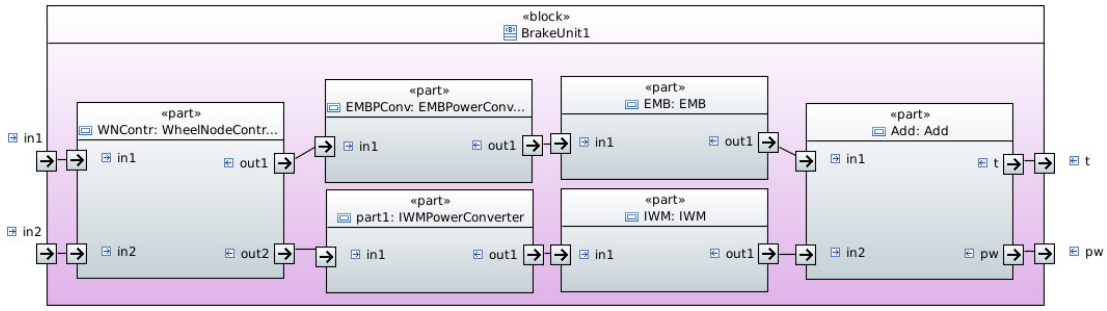

Fig. 3. Internal Block Diagram of wheel Brake Unit subsystem.

The internal implementation is defined by creating new instances of the previously declared components, which, following the UML terminology, become a *«part»* of the container, and connecting their ports in an *Internal Block Diagram*. Since HBS comprises four wheel braking subsystems, an *internal block diagram* should be created for each wheel brake unit.

As illustrated in Fig. 3, each wheel *Brake Unit* comprises: a *Wheel Node Controller* component, which calculates the amount of braking torque to be produced by each wheel braking actuator, and it sends commands to *Electromechanical Braking (EMB)* and *In-Wheel Motors (IWM)* power converters that control *EMB* and *IWM* braking actuators. The *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, the *Add* component outputs the braking torque, and the generated power while braking. Thus, HBS architecture comprises 4 subsystems, 24 subsystem components, 6 components, and 69 connections among subsystems and components.

B. Failure Logic Analysis

1) Annotation. In order to perform failure logic analysis (Fig. 1b), all system components and subcomponents should be annotated with *«fLABehavior»* stereotype. To obtain the necessary information to perform such annotation, each component should be analyzed to:

- **Identify output deviations**: failures on component output ports that can contribute to system failures. So, FPTC expressions can be written as illustrated in Fig. 4a. An output deviation is specified by pointing out which output port is affected followed by a failure mode, e.g., *out1.omission*. Fig 4a. illustrates *"omission"* and "*value subtle*" output deviations for the *IWMPowerConverter* braking system component *out1* port;
- **Identify contributing input deviations to output deviations:** contributing input deviations should be specified for each output deviation, as illustrated by FPTC expressions for the *IWMPowerConverter* (Fig. 4a). The first FPTC expression shows that the occurrence of an *omission* failure in *in1* input port implies an *omission* failure on the *out1* port. The second expression shows that a *subtle* incorrect value through the *in1* input port may lead to producing an incorrect value in *out1* output port; and
- **Annotate incoming errors from root input ports:** if the system and/or subsystems stated in the *block definition diagram* have any root input ports, those should be annotated with a failure mode. So when the analysis is executed, such failure mode is propagated throughout the system following the FPTC logic defined to each one of the system subcomponents. Fig. 4b illustrates *Brake Unit1* component root input ports annotated with *omission* failure modes.

After identifying how failures in input and output ports of different components may contribute to system failures, CHESS-FLA can be executed. Once the analysis is executed all output ports of each one of the stated HBS components, subsystems and sub-components will be annotated with failure modes resulting from the analysis.

2) Analysis and results. Two scenarios were considered in performing failure logic analysis for the *BrakeUnit1* subsystem from the HBS case study. In each scenario, different incoming input deviations were defined. In the first scenario, *omission* failure modes coming from brake unit input ports can cause *omission* of braking torque. In the second scenario, *omission* and *value subtle* failure modes propagated from brake unit input ports can cause a wrong braking torque. Details are shown in Fig. 5; note that, besides those on the input ports, *«FTPCSpecification»* annotations are automatically added by the CHESS-FLA analysis plugin. In the scenario illustrated in Fig. 5, *omission* failures in brake unit's input ports propagate throughout the *Wheel Node*

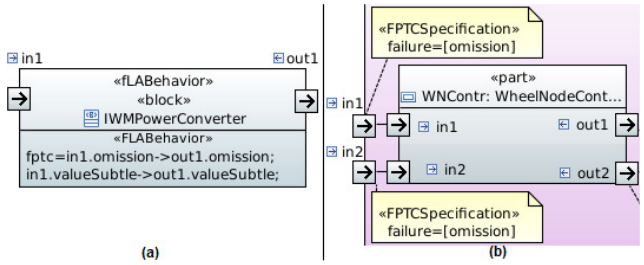

Fig. 4. (a) IWM power converter FLA, (b) Brake Unit input ports with failure annotations.

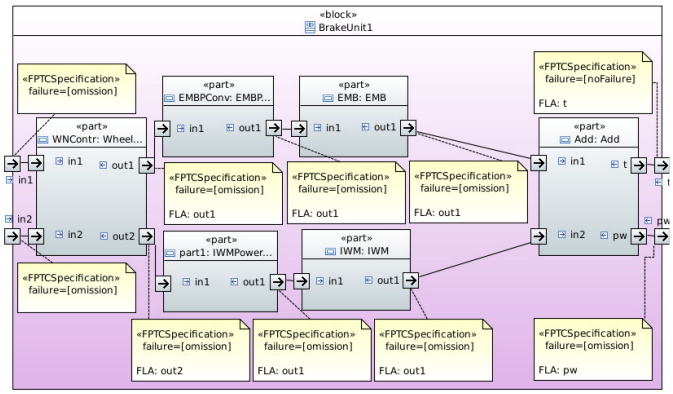

Fig. 5. Failure logic analysis results.

Controller input ports which further propagate throughout input ports from *EMB Power Converter*, *IWM Power Converter*, *EMB* and *IWM* components. In the end, the *Add* component receives an *omission* failure on both of its input ports, omitting the data that have not been propagated throughout its *pw* output port, meaning that the *BrakeUnit1* component won't send any power to the *Powertrain Battery* in the occurrence of *omission* failures on both of its input ports.

To summarize, the output of this analysis which is a failure mode such as an *omission* can actually be propagated as an output to the *pw* port of a *BrakeUnit*. In the following, using CHESS-SBA, we estimate their probability of occurrence, which will make it possible to determine the exposure to the related hazard.

C. State-Based Analysis

1) Annotation.

In order to perform state-based analysis, preliminary information or assumptions on the system architecture are used as input. System components that are mechanical or are going to be implemented in hardware should be annotated with *«simpleStochasticBehavior»* stereotype, since hardware failures are traditionally described using a probabilistic behavior. Repositories or handbooks like MIL-HDBK-217F [25] can be used to set failure rates. For software components, two options are possible. They can be annotated with *the «errorModelBehavior»* stereotype when a detailed failure model, e.g., containing errors, internal failures and repair rates, is needed. Otherwise, the same annotations used for failure logic analysis, *«fLABehavior»,* can be (re-) used. It is important to highlight that, when detailed information is available, the error model *(«errorModelBehavior»*) can be applied to hardware components as well. Thus, the following three kinds of annotations are possible for CHESS-SBA:

«simpleStochasticBehavior». In this case, dependability properties of the given hardware component should be specified: the time to the occurrence of a failure (as a probability distribution), possible failure modes and their relative probabilities of occurrence (optional), and the time required to repair the component after the occurrence of a failure (optional), which is also specified as a probability distribution (Fig. 1c).

In the hybrid braking system, *Auxiliary Battery*, *Powertrain Battery*, *Electronic Pedal*, and *Mechanical Pedal* hardware components were annotated with the *«simpleStochasticBehavior»* stereotype. As illustrated in Fig. 6, the time to failure of the *Mechanical Pedal* follow an exponential distribution with rate of 1.0e-6 per hour of operation. Once a failure occurs, only two possible failure modes can be propagated throughout its output ports: there is 90% of probability of propagation of an "*omission*" failure, and 10% of probability of propagation of a "*value subtle*" failure mode;

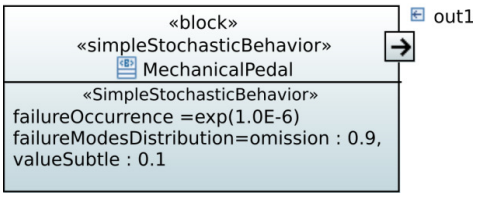

Fig. 6. Mechanical pedal component stochastic behavior.

«errorModelBehavior». In this case, a finite state machine or a set of finite state machines, containing information such as component internal fault occurrence and repair rates, relevant external faults (received on input ports), and possible failure modes (affecting output ports) must be defined, i.e., the activity "*Define Error States*" in Fig. 1c must be performed. This also implies that transitions among erroneous states should be created, and that the error model state machine must be linked to the component. In the HBS system, all software components, except *Add* components within wheel *Brake Units* were annotated with *«errorModelBehavior»* stereotype.

Multiple error models can be defined for the same component, addressing different perspectives. For example, the *Electronic Pedal* in the HBS, has two error models: one for modeling *internal fault occurrence* and propagations of internal faults (Fig. 7a), and another for modeling the *effect of* external faults (Fig. 7b);

«flABehavior».In this case, the steps *Identify Output Deviations* and *Identify Contributing Input Deviations* to the

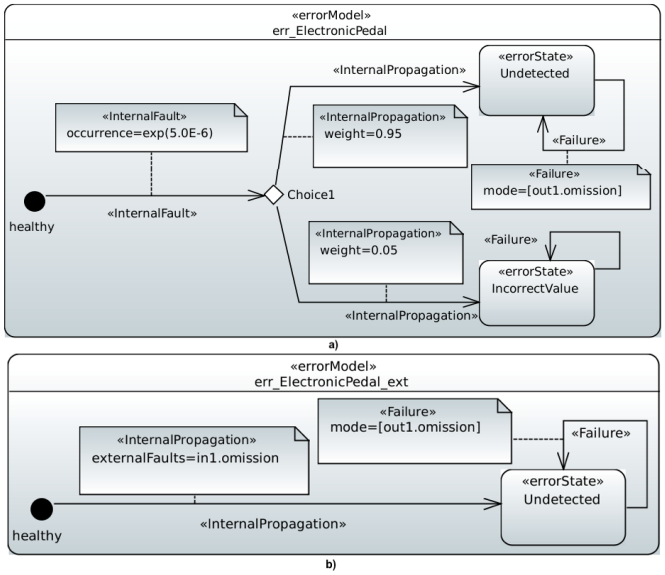

Fig. 7. Electronic Pedal component error models.

output deviations defined in the *Failure Logic Analysis* workflow (Fig. 1b) must be performed again in order to model the component failure behavior. In the HBS, the components annotated with *«fLABehavior»* stereotype were *Add* sub-components from *Brake Unit* subsystems. The reason for this choice is the *Add* sub-components being very simple comparators, thus, we assume no internal faults, but only propagation of errors received as input which can be specified via FPTC expressions as illustrated in Fig. 8.

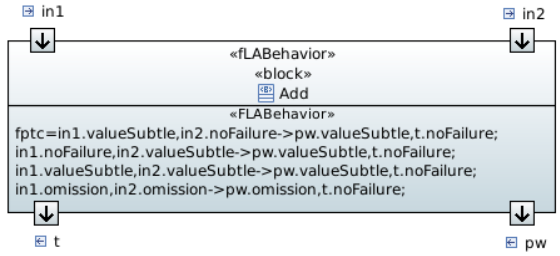

Fig. 8. Add component FPTC expressions for state-based analysis.

2) Analysis and results. In order to actually ***Execute State-Based Analysis*** (see Fig. 1c), the targeted system, in this case HBS, should be annotated with the *«CHGaResoursePlatform»* stereotype. This is part of the CHESS development methodology, which is capable of building a tree of instances (UML *InstanceSpecification* elements), out of multiple hierarchical UML *Composite Structure* or SysML *internal block* diagrams. After that, a "*ComponentName_instSpec"* package is automatically generated by CHESS for each composite component/block, containing all the component instances specified in the diagram.

Once instances are generated, a new *Class Diagram* is created in the CHESS *DependabilityAnalysisView*. In this diagram, a new *Component* annotated with *«stateBasedAnalysis»* stereotype is created for each metric to be analyzed. The details of the metric to be analyzed are set using attributes of such stereotype (see Fig. 9). The *platform* points to the package of instances under analysis; *measure* specifies the kind of measure, while *targetDepComponent*, *targetPort,* and *targetFailureMode* are used to identify the specific component(s), and possibly ports and failure modes of interest. Analysis results calculated by the CHESS-SBA tool are written in the *measureEvaluationResult* parameter of the component annotated with *«stateBasedAnalysis»* stereotype, in a process known as back-annotation.

In the HBS case study, the objective of the state-based analysis we want to evaluate, quantitatively, is the probability of occurrence of the *omission* failure mode highlighted by the execution of CHESS-FLA, and its impact on the system-level wheel brake functionality. Accordingly, we create a component, annotated with the *«StateBasedAnalysis»* stereotype (Fig. 9). In the created component, the *targetDepComponent* references the four brake units, and *targetPort* the four respective *pw* ports. The measure is defined as *Reliability {instantOfTime = 8760}*, meaning instant of time reliability at time 8760 (hours), and the *targetFailureMode* is *omission.* The interpretation of this specification is: "*what is the probability that, after 8760 hours (1 year), none of the four brake units have failed on their port pw with failure mode omission?*". This probability is given by the "*measureEvaluationResult*" value shown in the component.

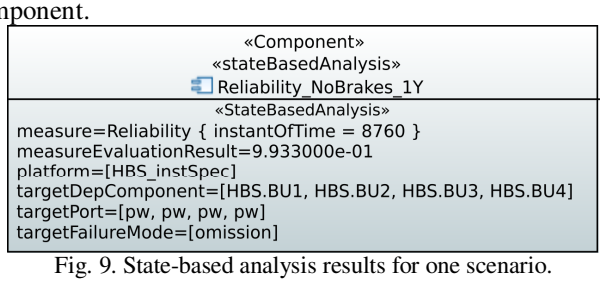

Fig. 9. State-based analysis results for one scenario.

This information should be used to mark the related hazards ("*no brakes*") as actually possible, and further actions should be taken when designing the hardware and software architectures. Thus, the proposed process has supported safety analysts in complying with ISO 26262 *3-7 Hazard Analysis and Risk Assessment* and *3-8 Functional safety concept* as illustrated in TABLE I.

TABLE I. HBS HAZARD ANALISYS, RISK ASSESSMENT, AND ALLOCATED SAFETY REQUIREMENTS.

Hazard	Hazard Causes	Prob. of Occurence	ASIL
No braking four wheels	Omission-BU1.out1 OR Omission-BU2.out1 OR Omission-BU3.out1 OR Omission-BU4.out1	9.933000e-01	D
Value braking	Value-BU1.out1 OR Value-BU2.out1 OR Value-BU3.out1 OR Value-BU4.out1	9.843000e-01	D

V. THE PROPOSED PROCESS AND ISO 26262

System and subsystem definition modeling process activities defined in Fig. 1a cover ISO 26262 2.5.2.2, and *3-5 Item definition* (TABLE II), producing a high-level description of the item (*system*) in a *block definition diagram*, and a low-level description of the items in *internal block diagrams*. CHESS-SBA process activities (Fig. 1c), support safety analysts in creating a detailed error for an existing item (*component*) in case of modification in the item or in its environment, supporting ISO 26262 *3-6.4.1 Determination of development category* (TABLE II). This is useful for the reuse of an existing item in other similar projects, since new failures modes can be raised when the item/environment is changed.

CHESS-FLA and SBA activities, and the execution of FLA model simulations support analysts in identifying the potential threats to the overall system safety, by producing information, e.g., component deviations and error models, that supports the analysis of failure propagation and identification of emergent hazardous behaviors in both hardware and software items, addressing ISO 26262 3-7. Additionally, the execution of SBA model simulation process task produce the calculus of the level of exposure to each identified hazard, thus, supporting analysts in hazard classification, addressing ISO 26262 3-7. The results of the execution of CHESS-FLA/SBA model simulations process activities support analysts in the derivation of Automotive Safety Integrity Level (ASIL) to be allocated to mitigate hazards and hazardous

items, and allocation of functional safety requirements through the system architecture. Thus, addressing ISO 26262-3-8: *Functional Safety Concept* and 4-6.4.2: *Safety Mechanisms*. The results of CHESS-FLA/SBA simulations also support the analysis of the impact of item failures in the overall system safety and determining measures to control random item failures, thus, respectively addressing ISO 26262 4-7.4.3.1 and 4-7.4.4. Finally, the execution of FLA/SBA simulations process activities support analysts in verifying whether or not the developed system architecture addresses the safety requirements, as stated in ISO 26262 4-7.4.8: *Verification of system design* (TABLE III). The relation of the proposed process and CHESS methodology with ISO 26262 activities and work products are summarized in Table II and Table III.

VI. RELATED WORK

In the literature, the necessity of modeling and analyzing non-functional properties at the architectural level is a research area that has flourished in recent years. Various architecture description languages (ADLs) with support for non-functional properties have been introduced. Given the wide adoption of UML, many of the proposed languages have been implemented as UML profiles. EAST-ADL2 [7], for instance, is a modeling language for electronic systems engineering within the automotive domain, which extends UML and SysML with various concerns, including safety analysis. Due to its nature, EAST-ADL2 is very tied to the automotive domain and to the AUTOSAR platform.

Across the years, OMG has released different profiles addressing non-functional properties. The most successful is the MARTE profile [20], which targets real-time properties and provides some stereotypes to specify the hardware architecture with richer details. The OMG "Dependability Assurance Framework for Safety-Sensitive Consumer Devices" [28] proposes a language for assurance for consumer devices. However, the specification explicitly excludes critical systems such as avionics or railways [28].

The work in [4] defines the Dependability Analysis Modeling (DAM) profile, which extends MARTE with the possibility to specify dependability properties. From the methodology viewpoint, DAM does not impose constraints to the modeler, allowing him/her to introduce inconsistencies, as the same information may be entered in multiple ways. From the practical viewpoint, we are not aware of tools actually implementing the DAM profile, or frameworks based on it.

Outside of the UML world, the most complete proposal is AADL, defined by the SAE "Architecture Analysis and Design Language" standard; the AADL Error Model Annex [24] is of particular relevance, since it allows users to add dependability-related information to AADL architecture models. Tool support is provided by the OSATE suite [5], which also includes some plugins to perform dependability and safety analysis. The CHESS Framework, being based on UML profiles and separated design views, provides better support of multiple concerns and greater extensibility.

HiP-HOPS [1] is a well-known methodology and tool to perform semi-automated safety analysis at the software/system architecture level, using failure logic analysis. The HiP-HOPS toolset is however commercial software. Furthermore, it does not address some of the concerns addressed by the CHESS Framework, e.g., schedulability analysis and code generation.

To the best of our knowledge, at the time of writing PolarSys CHESS [22] is the only open source toolset for the development of embedded systems which simultaneously provides: i) a UML-based language for the specification of non-functional properties, ii) a customized editor with enforcement of modeling constraints, iii) qualitative dependability analysis, iv) quantitative dependability analysis, vi) schedulability analysis, vii) back-annotation of analysis results, and viii) code generation.

Recently, since standards like ISO 262626 [15] and DO-331 [23] have started addressing model-based development; works on how to apply existing MDE techniques in a way that is compliant with standards have started to appear. The work in [6], discusses an artifact-centric compliance demonstration approach for software developed using with code generation. The work only addresses software certification, that is, Part 6 of ISO 26262. The work in [10] addresses qualification of code generation tools with respect to

TABLE II. PROCESS/CHESS METHODOLOGY, AND ISO 26262.

ISO 26262 Activities	The Proposed Process Activities	The Proposed Process Work Products
3-5: Item definition	System Definition: 1.1 – Create a Block Def. Diagram	1-System Model in a *Block Definition Diagram*
	Subsystem definition: 1.2 – Create an Internal Block Diagram	1-Subsystem *internal block diagrams*
3-6: Initiation of the safety lifecycle	3 - CHESS-SBA activities	3-Detailed component error models
3-7: Hazard analysis and risk assessment	2-CHESS-FLA and 3-SBA activities	2-Contrib. component deviations
		3-Component error models
	2.5 - Execute failure logic analysis	2-System, subsystem, components annotated with failure propagation information.
		2-Identified hazards
	3.8 - Execute state-based analysis	3-Risk assessment and hazard classification: evaluation of the risk posed by each hazard
3-8: Functional safety concept	3.8 - Execute state-based analysis	3-Allocated functional safety requirements to the architecture and safety integrity requirements to mitigate hazards
4-6.4.2: Safety mechanisms		3-Allocated safety integrity requirements to address fault detection and fault mitigation
4-7.4.3.1: Measures for avoidance of systematic failures	2.5 - Execute failure logic analysis, 3.8 - Execute state-based analysis	2-CHESS-FLA (Fig. 5.) and 3-SBA model simulations (Fig. 9)
4-7.4.8: Verification of system design		
4-7.4.4: Measures for control random hard-ware failures	3.8 - Execute state-based analysis	3-CHESS-SBA simulation interpretation enables analysts to determine measures for detection, control, or mitigation of random hardware failures

TABLE III. CHESS AND ISO 26262 WORK PRODUCTS.

ISO 26262 Part	ISO 26262 WP	CHESS Methodology WP
3-5: Item definition	3-5.5 Item definition	CHESS system and component models and their instances.
3-6: Initiation of the safety lifecycle	3-6.5.1 Impact analysis	CHESS-FLA and SBA
3-7: Hazard analysis and risk assessment	3-7.5.1 HARA	CHESS-FLA and SBA
	3-7.5.2 Safety goals	CHESS SBA
	3-7.5.3 Verification review report of HARA and safety goals	CHESS-FLA and SBA model simulations
3-8: Functional safety concept	3-8.5.1 Functional safety concept (FSC)	Risk exposure calculus derived from CHESS-SBA results.
	3-8.5.2 Verification report of the FSC	

ISO 26262-6. Similarly, an approach for the qualification of model-based tools according to ISO 26262 is proposed in [14]. The work in [12] analyzes the adequacy of a model-based testing tool, Fail-SafeMBT, with respect to the DO-178C/DO-331 standards. The authors define a process for using the tool in accordance with the work products expected by the standards. The authors of [21] performed a survey on model-based approaches to support avionics software development and certification according to DO-178C, concluding that there is a lack of integrated approaches [21]. In this paper, we make a step forward in demonstrating the applicability of the integrated PolarSys CHESS methodology as a support for the standard-compliant certification of critical systems.

VII. CONCLUSION

This paper has presented a systematic process to support system designers and safety analysts in using the CHESS PolarSys methodology capabilities in the system design and dependability analysis and modeling. The proposed process prescribes a set of steps to perform system design, dependability analysis using failure logic analysis and state-based stochastic analyses in compliance with DO-331 MBD principles and safety standards. The feasibility of the proposed process was verified by applying it for design and dependability analysis of a real world automotive hybrid braking system, originally designed in MATLAB/Simulink.

This paper contributed with real life examples and walkthroughs to support the end user in working with the toolset of the CHESS methodology. The proposed process and case study are expected to stimulate system designers and safety analysts in incorporating the CHESS methodology and toolset in their safety-critical systems projects, and guide them at producing certifiable evidence in conformance with safety standards. Being the toolset released as open source [22], we also aim to stimulate the interest of researchers and engineers in improving and extending the capabilities of the CHESS.

Further work intends to investigate the integration of variability management techniques and tools into CHESS methodology and toolset. In this way, we intend to present how CHESS can be combined with exiting Eclipse-based variability analysis and management techniques [26] to enable support for software product line engineering and systematic reuse of a CHESS-ML model. Thus, supporting the specification of different error models for the same component based on domain expertise, and support variability resolution based on the concrete application design. We also intend to investigate how dependability results can vary for a given system model when derived from a reusable CHESS model. Further work also intends to provide additional validation on in-depth scenarios with other types of failure modes such as timing and commission failures.

REFERENCES

[1] Adachi, M. Papadopoulos, Y., Sharvia, S., Parker, S., Tohdo, T. An approach to optimization of fault tolerant architectures using HiP-HOPS. Software: Practice and Experience, 41: 1303–132, 2011.

[2] ARTEMIS-JU-100022 CHESS – Composition with guarantees for High-integrity Embedded Software components aSsembly, available on-line: http://www.chess-project.org/

[3] ARTEMIS-JU-333053 CONCERTO - Guaranteed component assembly with round trip analysis for energy efficient high-integrity multicore systems, available on-line: http://www.concerto-project.org/.

[4] Bernardi, S., Merseguer, J., Petriu, D. A dependability profile within MARTE Software and Systems Modeling, Springer Berlin / Heidelberg, 2011, 10, 313-336.

[5] Carnegie Mellon University, OSATE 2.3.0, http://osate.org (Accessed 20 november 2017).

[6] Conrad, M., Artifact-centric compliance demonstration for ISO 26262 projects using model-based design. In Proceedings of the GI-Jahrestagung, v. 208, pp. 807-816, 2012.

[7] P. Cuenot et al. "The EAST-ADL Architecture Description Language for Automotive Embedded Software". In: Giese H., Karsai G., Lee E., Rumpe B., Schätz B. (eds) Model-Based Engineering of Embedded Real-Time Systems. Lecture Notes in Computer Science, vol 6100. Springer, Berlin, Heidelberg, 2010.

[8] CONCERTO, Deliverable D5.6: "Use case Evaluations – Final Version", May 2016. http://www.concerto-project.org/results

[9] De Castro, R., Araújo, R. E., Freitas, D. "Hybrid ABS with Electric motor and friction Brakes', 22nd International Symposium on Dynamics of Vehicles on Roads and Tracks, Manchester, UK, 2011.

[10] Dion, B. A Cost-Effective Model-Based Approach for Developing ISO 26262 Compliant Automotive Safety Related Applications. SAE Technical Paper, 2016.

[11] EUROCAE. ARP4754A - Guidelines for Development of Civil Aircraft and Systems, EUROCAE, 2010.

[12] Gallina, B., Andrews, A. Deriving verification-related means of compliance for a model-based testing process. In Proc. of IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1-6, 2016.

[13] Gallina, B., Atif Javed, M., Ul Muram, F., Punnekkat, S. A model-driven dependability analysis method for component-based architectures. In Proceedings of the Euromicro-SEAA Conference, IEEE Computer Society, Cesme, Izmir, Turkey, September, 2012.

[14] Hillebrand J., Reichenpfader P., Mandic I., Siegl H., Peer C. Establishing confidence in the usage of software tools in context of ISO 26262. In: Proc. of SAFECOMP 2011. LNCS, v. 6894. Springer, 2011.

[15] ISO. ISO 26262: road vehicles functional safety, 2011.

[16] Mazzini, S., Favaro, J., Puri, S., Baracchi, L. CHESS: an open source methodology and toolset for the development of critical systems. In Join Proceedings of EduSymp/OSS4MDE@MoDELS, 2016, pp. 59-66.

[17] Montecchi L., Gallina B. SafeConcert: A Metamodel for a concerted safety modeling of socio-technical systems. In: Model-Based Safety and Assessment (IMBSA). LNCS, v. 10437, Springer, 2017.

[18] Montecchi, L., Lollini, P, Bondavalli, A."A reusable modular toolchain for automated dependability evaluation. In VALUETOOLS, Torino, Italy, pp. 298-303, 2013.

[19] Object Management Group (OMG), "MDA Guide rev 2.0", OMG Document ormsc/2014-06-01, June 2014.

[20] Object Management Group. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version 1.1. OMG Document formal/2011-06-02. June 2011

[21] Paz, A., El Boussaidi, G. On the exploration of model-based support for DO-178C-compliant avionics software development and certification. In ISSRE Workshop, IEEE, pp. 229-236, 2016.

[22] PolarSys CHESS, available on-line: http://www.polarsys.org/chess.

[23] RTCA. DO-331: model-based development and verification supplement to DO-178C and DO-278A. Radio Technical Commission for Aeronautics, 2011.

[24] Society of Automotive Engineers. SAE Standards: AS5506/1, Architecture Analysis & Design Language (AADL) Annex Volume 1, June 2006.

[25] U.S. Department of Defense, "Military Handbook – Reliability Prediction of Electronic Equipment", MIL-HDBK-217F, December 1991.

[26] Vasilevskiy, A., Haugen, Ø., Chauvel, F., Johansen, M. F., Shimbara, D. 2015. The BVR tool bundle to support product line engineering. In *Proc. of the 19th Int. Software Product Line Conf.,* ACM, New York, USA, 380-384.

[27] Wallace, M. Modular architectural representation and analysis of fault propagation and transformation. Electronic Notes in Theoretical Computer Science, v. 141 n.3, pp.53-71, December, 2005.

[28] Object Management Group (OMG), "Dependability Assurance Framework for Safety-Sensitive Consumer Devices (DAF)", Version 1.0, February 2016.