Variability Management in AADL Design and Dependability Analysis/Modeling

André Luiz de Oliveira
Department of Computer Science,
Federal University of Juiz de Fora,
Minas Gerais, Brazil
andre.oliveira@ice.ufjf.br

Rosana T. V. Braga
Mathematics and Computer Science
Institute, University of São Paulo,
São Carlos, Brazil
rtvb@icmc.usp.br

Paulo Cesar Masiero
Mathematics and Computer Science
Institute, University of São Paulo,
São Carlos, Brazil
Masiero@icmc.usp.br

Yiannis Papadopoulos
Department of Computer Science,
University of Hull,
Hull, United Kingdom
y.i.papadopoulos@hull.ac.uk

Ibrahim Habli
Department of Computer Science,
University of York,
York, United Kingdom
ibrahim.habli@york.ac.uk

Tim Kelly
Department of Computer Science,
University of York,
York, United Kingdom
ibrahim.habli@york.ac.uk

ABSTRACT

Safety-critical systems developed upon a Software Product Line Engineering (SPLE) approach have to address safety standards. Standards establish guidance for analysing and demonstrating dependability properties of the system at different levels of abstraction in order to obtain safety-certification credits. However, the adoption of an SPLE approach for developing safety-critical systems demands the integration of safety engineering into SPLE processes. Architectural Analysis and Description Language (AADL) and tooling provide support for architectural, behavioural, and error modelling. In safety-critical SPLE, variability modelling and management should be considered in both design and dependability analysis. Variation in the design and context may impact on AADL error modelling. This paper presents an analysis of the impact of design and context variation on dependability analysis, and an approach to support variability modelling and management in AADL dependability/error modelling. The approach enabled efficient management of the impact of design and context variations on the AADL error model, enabling the systematic reuse of AADL architectural, behavioural, error models. Thus, the effort and costs in performing AADL error modelling for different product variants has been reduced.

CCS CONCEPTS

• **Software and its engineering** → Software safety; **Software system models** → Model-driven software engineering; **Safety-critical systems.**

KEYWORDS

Dependability analysis; reuse; variability management, error modeling.

ACM Reference format:

A. L. Oliveira, R. Braga, P. Masiero, Y. Papadopoulos, I. Habli, T. Kelly. 2018. Variability management in AADL design and dependability analysis. SIG Proceedings Paper in word Format. In *Proceedings of ACM 22th Int. Software Product Line Conf., Gotenborg, Sweden*, 10 pages.

1 INTRODUCTION

Safety-critical systems (SCS) are computer systems in which the occurrence of failures may lead to catastrophic consequences. Due to the benefits of large-scale reuse, Software Product Line Engineering (SPLE) and component-based approaches have been largely adopted by the industry in the development of safety-critical systems, especially in automotive [5][14][29] and aerospace domains [6][10]. The usage of model–based engineering provides unambiguous expression of requirements and architecture, and automated support for safety assessment [28]. Thus, both industry and automotive and aerospace safety standards have been recognized the maturity of model-based development techniques, which have been largely adopted to support both system design and dependability analysis.

The development of safety-critical systems built upon component-based approaches have to address guidance defined in safety standards, e.g., ISO 26262 [14] for automotive, DO-178C [27] and SAE ARP 4754A [8] for aerospace. Safety standards establish that dependability properties of a critical system should be analyzed and demonstrated at different levels of abstraction before its release for operation. The adoption of an SPLE or component-based approach for developing safety-critical systems demands the integration of safety engineering into SPLE processes [10][25]. Component-based design and dependability analysis techniques [4][23][26] provide the automated support for safety engineering, and seamless integration between system design and dependability analysis. AADL/Error Annex is a model and component-based design and error modeling language and toolset that supports system architecture, behavior, and dependability/error modeling in a single model. It contributes to reduce the complexity in performing both design and dependability analysis/modeling. Whereas safety-critical SPLE involves safety engineering, variability management in dependability analysis should be considered through software product line (SPL) life-cycle. Dependability analysis can be defined as the identification, early on the design, of potential threats to system reliability, availability, integrity, and safety, their potential causes, and measures to avoid or minimizing their effects.

Variation in the system design and its context may affect system dependability analysis/modeling. Thus, different hazards, with different causes, and levels of risk for the overall safety might be raised according to design choices and targeted contexts. In this way, dependability analysis should be performed aware of the impact of variation in the design and context on dependability properties to enable the systematic reuse of both design and dependability information in a safety-critical SPLE. Since dependability properties may change according to the chosen product variants and context, variability in dependability analysis/modeling should be managed in safety-critical SPLE. Thus, mappings linking context/design variations to their realization in the dependability/error model, i.e., in Hazard Analysis and Risk Assessment (HARA), and component failure data, should be specified in the product line variability model.

The state of the art of variability management techniques [1][11][13][19][29][31][32] provide support for managing variation at requirements, architecture, components, source code, and test cases. These techniques were not originally designed/or used to manage variability in dependability/error models built upon compositional techniques, e.g., OSATE AADL [4], CHESS [23], MATLAB/Simulink & HiP-HOPS [26]. Due the availability of a robust toolset built around OSATE AADL, AADL has been largely adopted by the industry in the development of safety-critical systems, especially in automotive [5][29][30] and aerospace [6][10] domains. Existing work on variability modeling and management on AADL component-based models are only focused on the provision of variability patterns to support the system design [30]. These variability patterns were built upon the concepts of AADL component *inheritance* and *refinement*. However, to enable the usage of AADL to support safety-critical SPLE it is necessary to integrate variability modeling and management in the AADL error modeling.

Variability modeling and management in AADL design and dependability/error modeling enables the traceability of the variation in the architecture, behavior, and usage context throughout dependability assets, and the systematic reuse of both AADL design and error model. It contributes to reduce the complexity, effort and costs in performing AADL dependability/error modeling for specific product variants, since such analysis is not performed from scratch. Thus, artefacts such as fault trees and Failure Modes and Effects Analysis (FMEA), required by safety standards for certification of safety-critical systems, can be automatically generated, with the support of OSATE toolset, for a given product variant from the reused AADL design and error models. AADL dependability/error modeling can only be treated as a product line asset if the variability model contains information about variation points and their realization in the error model. Therefore, the AADL error model should be included in the SPL core assets. Thus, enabling the systematic reuse of the AADL design and error models, reducing the costs of generating fault trees and FMEA artefacts for a larger number of product variants built around a reusable product line design.

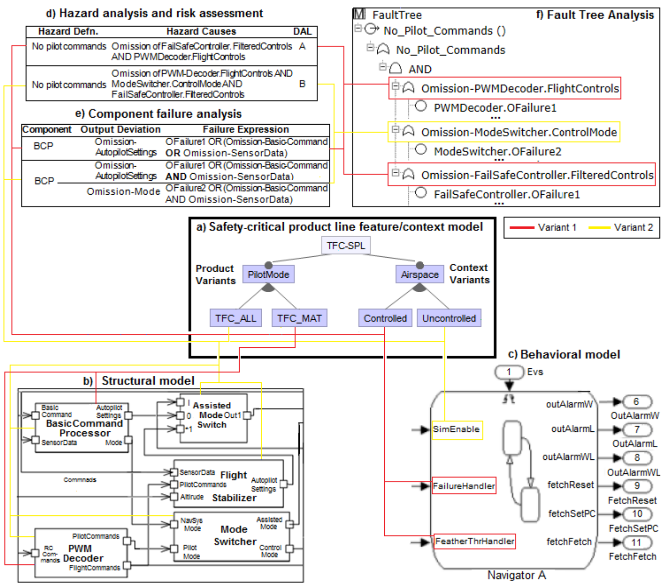
Although the systematic reuse of both AADL design and dependability/error models provided by an SPLE approach be an attractive idea, existing AADL variability modeling approaches only supports variability modeling in the AADL structural/architectural model. Thus, there is need for extending such approach [30] with support for variability modeling in the AADL behavioral and error models, to enable variability management in AADL behavior/error models. Thus, the contributions of this paper are: *i)* the proposal of a novel approach, which extends the current AADL architectural variability approach, with the support for variability modeling and management in the AADL behavioral and error models, and *ii)* an analysis of the impact of design and context variations on dependability analysis. The developed of tooling support for the proposed approach integrates BVR variant management, and OSATE AADL & Error Annex [4] compositional design/error modeling toolsets.

The proposed variability modeling/management approach and tooling for AADL behavioral and error modeling provides support for DEPendable-SPLE (DEPendable Software Product Line Engineering) approach [24]. DEPendable-SPLE extends traditional SPLE methods to address Dependability/Safety analysis in safety-critical SPLE, by enhancing both domain and application engineering phases with HARA, component fault modeling, variability modeling/management in dependability analysis. The proposed AADL variability modeling/management approach was applied in a realistic unmanned aircraft system SPL. The remaining of this paper is organized as follows. Section 2 presents the Tiriba flight control product line used through this paper. Sections 3 and 4 presents an analysis of the impact of design and usage context variations on architectural/behavioral and error modeling, and illustrates how variability can be specified in AADL structural, behavioral, and error modeling. Section 5 presents the related work. Section 6 present conclusion and future work.

2 TIRIBA FLIGHT CONTROL PRODUCT LINE

Tiriba Flight Control avionics software product line (TFC-SPL) is part of the Tiriba UAV-SPL [2], which comprises a control subsystem with the following goals: to start the flight mode (direct, stabilized, or autonomous), processing and setup flight commands, keeping flight conditions, and executing commands sent by the navigation subsystem. Two different TFC product variants in different usage contexts were considered through this paper. Although Tiriba has been originally designed in MATLAB/Simulink, to illustrate the application proposed AADL behavioral and error modeling variability approach and tooling, TFC architectural and error models were specified in AADL and AADL Error Annex [4]. BVR toolset [32] and the AADL/Error Annex BVR adapter developed by the authors and described in section 4, were used to support variability management in both AADL architecture and error models. TFC-SPL was designed based on an extractive strategy by analyzing the original Tiriba flight control subsystem Simulink model [31]. TFC-SPL comprises the *Pilot Mode* variation point, which comprises four pilot mode options/variants: *Manual Pilot* is mandatory, *Assisted Pilot*, *Autonomous Pilot*, and *Autopilot* are optional. *Manual Pilot* mode is a human operator sending commands to the unmanned aircraft vehicle (UAV) from a ground control station. *Autopilot* executes a pre-defined route. *Assisted Mode* allows the operator sending commands to the UAV configured with *Autopilot*. *Autonomous Pilot* allows the UAV performing actions according to its current environmental conditions captured by pressure sensors. *Assisted Pilot*, *Autonomous Pilot*, and *Autopilot* can be combined into several different ways, allowing seven different flight control variants. TFC product variants can operate in a range of different contexts defined by combining *Airspace*, *Application*, and *UAV Size* usage context variation points [2]. The composition of *pilot mode* and *context* variants leads to 84 different TFC variants. Since it would be prohibitive considering all these variants to perform dependability analysis, only TFC *manual and* *autonomous* (TFC-MAT) and *all pilot modes* (TFC-ALL) variants, *controlled and uncontrolled airspace* usage contexts, illustrated in Fig. 1a, were considered through this paper.

TFC-SPL architecture comprises 4 subsystems and 14 components, which are composed by 252 model elements and subcomponents. An excerpt of the TFC SysML main block diagram is shown in Fig. 1b. *Mode Switch* encapsulates the *pilot mode* variation point, whilst *Command Switch* encapsulates the variation point inherent to the source from the pilot commands, e.g., manual pilot subsystem, sent to the UAV. *Basic Command Processor (BCP)* and its ports represent the realization of the *Autonomous* pilot mode. *PWM Decoder* output port connected to *Command Switch* block represents the realization of the *Manual Pilot* in the design. The realization of *Autopilot* is given by: *Autopilot* subsystem, *PWMDecoder.FlightControls* output connected to *FlightStabilizer* and *CommandSwitch* model elements, and *FlightStabilizer.AutopilotSettings* port connected to *AssistedModeSwitch.* Finally, *ModeSwitcher.ControlMode* output port connected to *AssistedModeSwi-tch* and *CommandSwitch* blocks represent the realization of the *Assisted Pilot*.



Hazard Defn.	Hazard Causes	DAL
No pilot commands	Omission of FailSafeController.FilteredControls AND PWMDecoder.FlightControls	A
No pilot commands	Omission of PWM-Decoder.FlightControls AND ModeSwitcher.ControlMode AND FailSafeController.FilteredControls	B

Component	Output Deviation	Failure Expression
BCP	Omission-AutopilotSettings	OFailure1 OR (Omission-BasicCommand OR Omission-SensorData)
BCP	Omission-AutopilotSettings	OFailure1 OR (Omission-BasicCommand AND Omission-SensorData)
	Omission-Mode	OFailure2 OR (Omission-BasicCommand AND Omission-SensorData)

Figure 1: Product and usage context variation and their impact on design and error modeling.

Variability in safety-critical SPLs can be classified as system (design) or contextual variability [13]. System variability can occur in capability, operating environment, domain technology, implementation technique, or quality attribute features [15][21]. Capability features represent end-user visible characteristics. Operating environment comprises features associated with the target environment where a given software product is operated. Domain technology relates to features representing specific domain techniques or tools that can be used to implement the SPL assets. Implementation techniques are features regarding specific implementation strategies, e.g., redundant or non-redundant control system architectures in the avionics domain. Quality attribute refers to features that the SPL products must address such as usability, maintainability, and data integrity checking. Usage context features relate to where and how the SPL is used. Such classification has been considered through this paper. Combinations among these features act as key driver during safety-critical SPL design, dependability analysis, and product derivation/instantiation. Such multi-perspective features and combinations among them have a direct influence in SPL architectural decisions and dependability analysis aimed at safety certification.

Interactions among product and usage context features define constraints that may impact on the design, in which different design choices must be taken according to the targeted context. In both non-critical and safety-critical SPLs, variation points and their variants defined in a feature model have a directly impact on the derivation of variant-specific structural/architectural and behavioral models. The following sections presents an analysis of the impact of variation in design and usage context features on the AADL design and dependability modeling, and it illustrates how variability can be specified in both AADL structural, behavioral, dependability/error models. Such analysis was performed by considering the following AADL system architectures and their respective error models: aircraft braking system [9], door controller [22], and Tiriba UAV [31] product lines.

3. THE IMPACT OF VARIABILITY IN AADL DESIGN

In the development of safety-critical systems using compositional modeling languages such as AADL and CHESS, the system architecture is often expressed in data-flow oriented models, and the system behavior is expressed in Finite State Machines (FSMs). In the following, we present variability types in architectural and behavioral models and how such variability can be specified in AADL structural and behavioral models.

3.1 Architectural/Structural Variability and AADL

In AADL fashion data-flow oriented architectural models, structural variability can be found in systems, subsystems, components, their ports and connections via flows from input to output ports, which may change according to the targeted usage context. Considering the TFC-SPL [31], the selection of TFC-MAT variant from *Pilot Mode* variation point (Fig. 1a) implies in the selection of *PWM Decoder* and *Basic Command Processor* components, their ports and connections (Fig. 1b) during the product derivation process.

The realization of TFC-MAT and other product variants in the architecture model are highlighted in Fig. 1 with lines linking features (Fig. 1a) to their realization in components and their ports (Fig. 1b). It is important to highlight that variation in the context combined with variation in product features, and isolated variation in the usage context may impact on the derivation of variant-specific architectural and behavioral models. Considering the TFC-SPL, the selection of *Pilot Mode* product and *Airspace* context variants impacts on the derivation of redundant or non-redundant components in a variant-specific architecture model. Still in TFC-SPL, product features associated with *assisted*, *autopilot*, *autonomous*, and *manual* pilot modes, i.e., *all pilot modes* (TFC-ALL) product variant (Fig. 1a), are materialized in the architecture by all flight control components, their ports and connections (Fig. 1b).

In AADL, structural/architectural variability can be classified as: *definition* and *implementation* variability [30]. *Definition* variability relates to variation in the number and types of ports in an AADL system, software (*process*, *thread*, *thread group*, or *subprogram*), or execution platform (hardware) component (*device*, *processor*, *memory*, *or bus*) specification which might change from a variant to another. *Implementation* variability relates to variation in the number and types of subcomponents of a given *system* component, and in the way that subcomponents are connected in a *system component*, which might change from a variant to another. Both *definition* and *implementation* variability can be specified in an AADL model via: a) AADL component *inheritance* and *refinement*, or b) with the support of a variant management tool. In AADL component inheritance, variability can be specified using a hierarch of *system*, *software*, or *execution* platform component definitions and implementations [30]. On the other hand, variability can also be specified with the support of a variability management tool, which enables the definition of mappings linking product/context variants to their realization in the system architecture in the variability model. Variability management tools, e.g., BVR [32] and pure::variants[29], provide pruning techniques that enable the removal of undesirable model elements of a variable design specification during the product derivation. Details about AADL syntax and diagram notation for component definition and implementation can be found elsewhere [4].

Figure 2 illustrates the usage of AADL component *inheritance* to specify variability in the definition of *Basic Command Processor* (BCP) system component from the TFC-SPL. Thus, the base BCP *system* definition, shown in Figure 2a, comprises two input ports and *autopilotSettings* output port, applicable when the TFC-MAT (Figure 1a) product variant is chosen. The extended definition for BCP system component, shown in Figure 2b, comprises the three aforementioned input and output ports, inherited from *Base_BasicCommandProcessor* definition, and an additional *cmode* output port. Thus, the extended BCP system definition is included in the final product when TFC-ALL (Figure 1a) variant is chosen. Figure 3 shows the specification of the extended BCP system definition in AADL code, which extends the base BCP system definition (line 2) with *cmode* variant-specific output port (line 4).

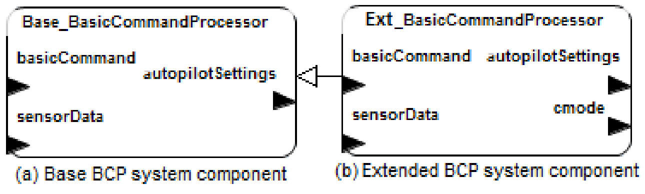

Base_BasicCommandProcessor
basicCommand
autopilotSettings
sensorData
Ext_BasicCommandProcessor
basicCommand
autopilotSettings
sensorData
cmode
(a) Base BCP system component
(b) Extended BCP system component

Figure 2: Variation in basic command processor system definition.

```
01 system Ext_BasicCommandProcessor
02    extends Base_BasicCommandProcessor
03    features
04       cmode: out data port; -- Variant
05 end Ext_BasicCommandProcessor;
```

Figure 3: Extended BaseCommandProcessor AADL code.

Figure 4 shows excerpts of the AADL code for base and extended Tiriba Flight Control System (FCS) implementations, which variation in subcomponents and their connections were specified using AADL component *inheritance* and *refinement* concepts. Base FCS system implementation (Figure 4a) contains *Base_AssistedModeSwitch* process, *Base_BasicCommandProcessor* and *Base_ModeSwitcher* system subcomponents, and their connections that represent the realization of the TFC-MAT product variant in the design (see fig 1b). The extended FCS system implementation, shown in Figure 4b, inherits *subcomponents* and *connections* defined in the base FSC implementation and it refines (lines 4-6) the specification of three subcomponents defined in the base FCS implementation. Subcomponent implementation refinements are expressed with AADL "***refined to***" statements. Since the extended BCP system comprises *cmode* additional output port, the extended FCS implementation contains an additional variant-specific connection between *BCP.cmode* and *ModeSwitcher.navSysMode* input port.

Table 1 summarizes the structural/architectural variability types and their relationships with AADL elements, identified from the analysis of the AADL and three different system models [9][22][31] with variability. By architectural variability types, we mean the elements of a product line architectural model in which variability can be present. AADL provides abstractions the support the specification of hierarchical system and subsystem/component decomposition in the same way as CHESS [23] and Simulink/HiP-HOPS [26]. *System* variability can be found in the *system name* and/or in the *number system* components which are part of a given system architecture. In AADL, such variation can be expressed in the *name* of a *Package*, which is the top element of an AADL architecture model, or in the number *Package* elements that comprises an AADL model. AADL enables the specification of more than one *Package* element in the same model, thus, supporting the design of modular system architectures. An example of *System* variability was

```
01 system implementation Base_FlightControlSystem.fcs
02   subcomponents
03       amw: process Base_AssistedModeSwitch.amw;
04       bcp: system bcp::Base_BasicCommandProcessor.bcp;
05       msw: system msw::Base_ModeSwitcher.msw;
06        ...
14   connections
15        bcp2amw: port bcp.autopilotSettings -> amw.in2;
16        ...
58 end Base_FlightControlSystem.fsc;
```

(a) Base flight control system implementation.

```
01 system implementation Ext_FlightControlSystem.fcs
02    extends Base_FlightControlSystem.fcs
03    subcomponents
04       amw: refined to process Ext_AssistedModeSwitch.amw; -- Variant
05       bcp: refined to system bcp::Ext_BCP.bcp; -- Variant
06       msw: refined to system msw::Ext_ModeSwitcher.msw; -- Variant
07        ...
15    connections
16        bcp2msw: port bcp.cmode -> msw.navSysMode; -- Variant
17         ...
61 end Ext_FlightControlSystem.fsc;
```

(b) Extended flight control system implementation.

Figure 4: Variation in AADL flight control system implementations.

Table 1: Architectural variability and AADL elements.

Variability Type	*AADL 2.0 Element(s)*
System	*Package*
Subsystem	*System*
Subcomponent	**HW**: *Processor*, *Memory*, *Device*, or *Bus*;

	SW: *Process, Thread, Thread Group, Subprogram,* or *Data*.
Port	*Data or Event Port, Subprogram, Parameter, Subcomponent access (requires, provides).*
Port Value	*Data* subcomponent connected (via Connection) to a system, hardware or software component.
Connection (data)	***Port**: Immediate, Delayed, Group, Refinement.*
	***Parameter**: Parameter Connection, Parameter Connection Refinement.*
	***Access**: Access Connection, Access Connection Refinement.*
Connection (event)	***Flow and Flow Implementation**: Flow path, Flow source, Flow sink.*

found in base and extended *basebcp* and *bcp* AADL *Packages* defined in the TFC-SPL AADL model. Each package stores different AADL model. Each package stores different definitions and implementations for the *Basic Command Processor* (BCP) system component (Figure 3).

Subsystem variability can be found in the *Subsystem name, type* and/or in the *number* of *Subsystem* components of an AADL architecture model. In the TFC-SPL, two different variant-specific Tiriba Flight Control (*sub) System* (TFCS) definitions and implementations were specified, one to address *manual and autonomous* variant, and another to address *all pilot modes* variant (Figure 4). Each flight control system have a different name, comprises subsystem component's definitions and implementations. *Subcomponent* variability represents variation in the *number* and/or *types* of *subcomponents* associated with a given AADL *System* element. In the TFC-SPL, such variation can be found in the *number* and *types* of subcomponents of the TFCS subsystem (Figure 4). The base TFCS implementation, shown in Figure 4a, comprises *BaseAssistedModeSwitcher.amw*, *BaseBCP.bcp*, and *BaseModeSwitcher.msw* subcomponent implementations. On the other hand, the extended TFCS implementation extends the base TFCS implementation with variant-specific implementations for each one of the aforementioned subcomponents (Figure 4b).

Subsystem/Subcomponent Port variability can be found in the *name, number* and/or *type* of ports of a given AADL *system*, *software* or *hardware* component. In AADL, *subsystem/subcomponent* ports define the way a component interfaces with other components. These connections can be specified using: a *Port*, a logical connection between components, which can be of the following types: data, event, or event data. A *port* can be input, output, or bidirectional; *Subprogram*, which can be an execution entry point in a data component, or an entry point for a remote procedure call; *Parameter*, a subprogram input or output variable; or an *Access*, to indicate that a subcomponent requires access to other subcomponents, to make a subcomponent accessible outside its containment hierarchy. *Port*, *Subprogram*, *Parameter* and *Access* port can be grouped in a *Feature Group*, providing a single connection point to these features. An example of system/subcomponent port variation can be found in variant-specific definitions for the *BCP* component. *BaseBCP* (Figure 2a) comprises two input ports and *autopilotSettings* out port, whilst *ExtBCP* (Figure 2b) comprises an additional *cmode* out port.

Port value variability is related to variation in the values assigned to a given input/output data, event, event/data port, subprogram input/output parameter, which might changes from a variant to another. In AADL, a *Data* subcomponent connected to a given *System/Subcomponent* is the equivalent representation of the assignment of values to *ports* in Simulink. Thus, variation in AADL port values can be found in the data stored in a *Data* subcomponent connected to a given *System/Subcomponent* that might from a variant to another. Variation in subcomponent port *type* and *value* can be specified using AADL *refinement to* statement system/software/hardware component implementations.

Connection variability relates variation in the way that components are connected in the direction of the data control flow, which can be unidirectional or bidirectional. *Connectio*n variability can be classified into: *data* or *event connection* variability. *Data connection* variability can be found in AADL: *Immediate*, *Delayed*, *Group* or *Refinement* connections among component input, output, and/or in/out data *ports*; *Parameter* and *Refinement* connections among input and output data *parameters* of a sequence of subprogram calls; and *Access* and *Access Refinement* connections among *requires* and *provides* component access ports which might change according to the targeted product variant. In an AADL *Access Connection Refinement*, a connection defined in a base component implementation can be refined in a variant-specific component implementation that extends the base implementation. An example of connection variability is variation in component connections specified in base and extended implementations of the TFCS shown in Figure 4. The extended TFCS implementation contains the specification of an additional connection linking *BCP.cmode* output port to *ModeSwitch.navSysMode* input port not present in the base TFCS implementation. *Event* or *Event/Data* connection variability can be found in the specification of flows between input, output, and/or in/out ports of AADL system, software or hardware component *definitions* and *implementations*. In AADL, *flow paths* are used to specify flows between component input and output ports. An AADL *flow sink* is used to specify the input port from which a flow started, and *flow source* is used to specify the output port from which is the end of a *flow specification*. *Flow specifications* might vary according to the component definition and implementation. For example, the AADL flow specification for base and extended BCP component definitions and implementations, shown in Figure 2, are different since the extended BCP contains an additional output port not present in the base BCP.

3.2 Behavioral Variability and AADL

Architectural variation inherent to TFC-MAT and TFC-ALL variants and in their context can be further propagated throughout Tiriba mission controller behavioral model expressed in a FSM (Fig. 1c). In safety-critical SPLE, variation in product and usage context features may also impact on the system behavior, which can be expressed in a FSM. Thus, FSM *states*, *state transitions*, and *events* that trigger state transitions might vary according to the targeted product/usage context. Variation in FSM can directly impact on elements of the system architecture, changing data port values, configuration of components, and their connections. FSM variation can be firstly found in the number and in the structure of the *state flows* associated with different product variants and their usage context. Variation in a *state flow* can be found in its *input* and *output* data, *states*, and *state transitions*. A *State* can have different local *variables* with different values, and it can be involved in different *transitions* according to the targeted product and usage context variants. *State Transition* variation can be found in *source* and *target* states, in the *event* that triggers the transition, its *execution priority* order and *outgoing events*. A transition *Event* may be triggered by different *mode/states* with different *guard conditions*. A *guard condition* is a condition that should be satisfied for the transition from a *source state* to a *target state*. *Transition events* may also have timing constraints. For example, an event is dispatched on an interval of one second after a system failure. *Effects* of a *transition event* represent changes in *state variables*, which might vary according to the targeted product variant and context. Finally, *state transitions* may dispatch different *outgoing events* affecting both behavior and structure of a given product variant. Examples of *outgoing events* are changes on *states* and *variables* from other FSMs, and changes in structural elements such as *systems*, *subsystems*, *subcomponents*, their c*onnections*, and *port* values.

Variation in FSMs can be found in Tiriba UAV optional mission features [31]. Variability in the selection of these features was specified in a FSM that defines the Tiriba mission controller using a mechanism of transitions conditioned to variables that define variability, as illustrated in Fig. 1c. Thus, when *Entry Segment Simulation* is selected, a simulation is performed whenever the UAV starts a new mission segment (Fig. 1c), in order to find out the suitable approach to switch between two mission segments. When *Feather Threshold* feature is selected, the UAV route is adjusted whenever the aircraft deviates more than a certain limit from the planned mission route, i.e., correction start state transition in the FSM. Finally, when *Failure Handler* is selected, the UAV is able to return to specific positions where a picture was not captured during the mission. The activation/deactivation of mission controller features are defined by alternative flows with *Boolean* conditional variables (Fig. 1c), which allow enabling/disabling flows and FSM states according to the selected product/context variants. Thus, for each mission-related feature, there is a variable whose value defines which FSM states and transitions will take place in the final product. *SimEnable* is a *Boolean* variable that controls the activation/deactivation of the behavior related to the *Entry Segment Simulation*. When this feature selected, this variable is set with *true* value. The same is valid for selection/deselection of *Failure Handler* and *Feather Threshold* features.

Context features may influence on the selection of product features that impact on the FSMs, changing the system behavior. In this way, when *uncontrolled* airspace context and TFC-ALL variants are chosen, *SimEnable* variable is set *true* activating the transition to the *Simulating* state in a variant-specific mission controller FSM. When *controlled* airspace context and TFC-MAT variants are selected, *Feather Threshold* and *Failure Handler* variables are set *true*, activating the *correction start* and *good simulation* state transitions. Thus, variation in the behavior of an AADL thread or subprogram software component can be specified using the aforementioned mechanism of state transitions conditioned to variables that define variability, which the variable values determines the state transitions, as depicted in Fig. 1c.

Otherwise to component *definition* and *implementatio*n, AADL *extends* statement cannot be used to specify variability in the behavior of an AADL *thread* or *subprogram* since the current version of AADL Behavior Annex does not allows inheritance of an *annex behavior specification* stated inside of a *thread* or *subprogram* implementation. Thus, variability in AADL behavior can be specified in the following ways: *i)* via multiple *annex behavior specifications* in a given thread/subprogram implementation, which each *behavior specification* statement contains the specification of a variant-specific finite state machine. or *ii)* via specification of multiple thread/subprogram definitions and implementations used to store variant-specific subprogram/thread variables and finite state machine. Independently from the chosen approach for variability modeling in AADL behavior, a variability management tool can be further used to support the specification of mappings linking product/context features to their realization in a *thread/subprogram definition*, and in states, state variables, and state transitions specified in a *thread/subprogram implementation*. Finally, a variability management tool can be used to resolve AADL behavioral variation during the product derivation.

Figures 5 and 6 show an example of the usage of two different AADL subprogram *definitions* and *implementations* to specify variability in the Tiriba mission controller finite state machine illustrated in Figure 1c. Details about *AADL Behavior Annex* can be found elsewhere [20]. Variability in each AADL variant-specific Tiriba mission controller FSM subprogram is specified using the mechanism of state transitions conditioned to variables [31] that determines the transitions. The *base navigator* subprogram definition shown in Figure 5a comprises the *simEnable Boolean* input parameter that controls the activation/deactivation of the behavior related to *Entry Segment Simulation* mission-related function. Thus, this parameter is set *true* when TFC-ALL variant and uncontrolled airspace context are chosen, and then, a simulation is performed whenever the UAV starts a new mission segment. It is important to highlight that the information gathered from the mission execution, by calling *startMissionSegment* subprogram (line 9 from *BaseNavigatorA.navA* subprogram implementation shown in Figure 5b, is stored into *simulationStart* out event data port (Figure 5a). In addition to *simEnable* ***in parameter***, and *simulationStart* out event data port, the *BaseNavigatorA* subprogram definition contains four input parameters, and seven out event data ports. The *BaseNavigatorA* implementation (Figure 5b) comprises *s0* initial final state, *simulating* and *simulationDone* states, and three state transitions (lines 08-11). In the transition *t1*, the subprogram moves from the initial state to the simulating state only if the *simEnable* input parameter is set *true*. In the

```
01 subprogram BaseNavigatorA
02   features
03     simEnable: in parameter Base_Types::Boolean;
08     simulationStart: out event data port Base_Types::Unsigned_16;
13 end BaseNavigatorA;
```

(a) BaseNavigatorA subprogram definition.

```
01 subprogram implementation BaseNavigatorA.navA
02    annex behavior_specification {**
03    states
04       s0: initial final state;
05       simulating: state;
06       simulationDone: state;
07       transitions
08          t1: s0 -[simEnable != false]-> simulating
09                   {startMissionSegment! (entrySimArgs->p);};
10          t2: simulating -[ ]-> simulationDone {…};
11          t3: simulationDone -[ ]-> s0 {…};
12    **};
13 end BaseNavigatorA.navA;
```

(b) Base NavigatorA subprogram implementation.

Figure 5: BaseNavigatorA subprogram definition and implementation.

simulating state, the *startMissionSegment* subprogram is called, and a simulation is performed whenever the UAV starts a new mission segment. After performing the simulation, the subprogram moves towards the *simulationDone* state (line 10). In *the simulationDone* state the subprogram moves towards the *s0* initial final state (line 11).

The *ExtNavigatorA* subprogram definition, shown in Figure 6a, extends *BaseNavigatorA* definition with variant-specific *failureHandler* and *featherThreshold Boolean* input parameters (lines 3-4), and four out event data ports, omitted for simplicity. *FailureHandler* and *featherThreshold* input parameters control the activation/deactivation of *feather threshold* and *failiure handler* Tiriba UAV mission controller functions. These functions are activated when *variable* values are set *true*, in this case, when TFC-ALL product and controlled airspace context variants are chosen. In addition to the states and transitions defined in Figure 5b, the *ExtNavigatorA* subprogram implementation comprises additional *failureHdl*, *featherThrHandler*, and *reset* states (lines 7-9), and *t4*, *t5*, *t6*, *t7*, *t8*, and *t9* state transitions (lines 15-20). The subprogram shown in Figure 6b moves from *simulationDone* state to the *featherThrHandler* state whenever *featherThreshold* subprogram input parameter is set *true* (line 15). In the *featherThrHandler* state, the *correctionStart* subprogram is called, and the UAV route is adjusted to the planned mission route. After that, the subprogram moves towards the *s0* state (line 20). The subprogram moves from *simulationDone* or *s0* state to the *failureHdl* state whenever *failureHandler* input parameter is set *true* (line 16). In the *failureHdl* state, the *goodSimulation(trackerBitParam)* subprogram is called, and UAV returns to specific positions where a picture has not been captured during the mission (line 17). After handling mission errors, the subprogram moves from *failureHdl* to the reset *state* (line 18). In the reset state, the subprogram moves towards *s0* (line 19), and then finish.

```
01 subprogram ExtNavigatorA extends BaseNavigatorA
02   features
03      failureHandler: in parameter Base_Types::Boolean; --Variant
04      featherThreshold: in parameter Base_Types::Boolean; --Variant
05      …
09 end ExtNavigatorA;
```

(a) ExtNavigatorA subprogram definition.

```
01 subprogram implementation ExtNavigatorA.navA
02    annex behavior_specification {**
03    states
04       s0: initial final state;
05       simulating: state;
06       simulationDone: state; --Variant
07       failureHdl: state; --Variant
08       featherThrHandler: state; --Variant
09       reset: state; --Variant
10    transitions
11     t1: s0 -[simEnable != false]-> simulating
12              {startMissionSegment! (entrySimArgs->p);};
13     t2: simulating -[ ]->simulationDone {…};
14     t3: simulationDone -[ ]-> s0 {…};
15     t4: simulationDone -[featherThreshold != false]->featherThrHandler{…};--Var
16     t5: simulationDone -[failureHandler != false]-> failureHdl {…}; --Variant
17     t6: s0 -[failureHandler !=false ]->failureHdl {…}--Variant
18     t7: failureHdl -[ ]-> reset {…}; --Variant
19     t8: reset –[ ]-> s0 {...}; --Variant
20     t9: featherThrHandler –[ ]-> s0 {…}; --Variant
21  **};
22 end ExttNavigatorA.navA;
```

(b) ExtNavigatorA subprogram implementation.

Figure 6: ExtNavigatorA subprogram definition and implementation.

Behavioral variability types and their relationships with AADL Behavior Annex [20] elements, identified from the analysis of AADL Behavior Annex, and three different system models [9][22][31] with variability, are summarized in Table 2. *State flow* variability relates to the variation in the number and structure of state flows associated with a subprogram or thread component implementation via *annex behavior specification*, which might change from a variant to another. Such variation is present in the mission controller state machines defined to address TFC-MAT and TFC-ALL variants shown in Figures 5b and 6b. *Data* variability relates to variation in the *name*, *type*, and *number* of input/output feature elements, which can be event/data ports, subprogram calls, parameters or subcomponent accesses, associated with a subprogram or thread definition. Feature elements specified in the definition of a subprogram/thread are inputs/outputs data for a state flow specified in a subprogram/thread implementation. *Data* variability can be found in the *number* of input parameters and output event data ports specified in base and extended *NavigatorA* subprogram definition, shown in Figures 5a and 6a, which respectively address TFC-MAT and TFC-ALL variants. *State* variability relates to variation in the *name*, *type*, and/or *number* of states in a state flow specified in a subprogram/thread implementation. Such variation can be found in the number of states of state flows defined for base and extended *NavigatorA.navA* subprogram implementations illustrated in Figures 5b and 6b.

The extended *NavigatorA.navA* state flow contains *failureHdl*, *featherThrHandler*, and *reset* states not present in the base state flow. *State variable* variability relates to variation in the name, type, and/or number of state variables specified in a state flow. *State transition* variability relates to variation in the name and number of state transitions specified in a state flow. For example, the extended *NavigatorA.navA* state flow contains variant-specific *t4, t5, t6, t7, t8,* and *t9* state transitions not present in the base *NavigatorA.navA* state flow. An AADL *state transition* might have different *source* and *target states*, *guard condition event, event triggered after* the state transition according to the target product variant. In AADL, guard condition variability relates to variation in the definition of the Boolean expression that triggers the state transition. In AADL, e*vent trigger* variability relates to variation in the

Table 2: Behavioral variability and AADL elements.

Variability Type	***AADL 2.0 Element(s)***
State flow	*Annex behavior specification* (a state/mode container)
Data (input and output)	*Port, Subprogram call, Parameter, Subcomponent Access*
State	*State*
State Variable	*State Variable*
State Transition	*Transition*
S Transition: Source State	*State*
S Transition: Event Trigger	*Subprogram or Port Identifier*
S Transition Evt: Guard Cond.	Boolean condition
S Transition Evt: Timing Constraint	*Transition*
State Transition Effect	Changes in *Property*/*State Variable* values, and in the configuration of components
S Transition: Target State	*State*
S Transition Ordering	Initial state transition and state transition sequencing

actions to be performing after a successful state transition, which can be modification of state variable, output parameter, or out event/data port values, or subprogram calls, which might change from a variant to another. The timing constraint of a transition guard condition might also vary from a variant to another. State transition effects/outgoing event variability relates to variation in the values assigned AADL *Property* elements and *state variables*, or changes in the *configuration of components* and their *connections* after a *state transition*. Finally, state transition ordering variability relates to variation in the initial state transition, and the order that the following state transitions are executed, which might change from a variant to another.

4 THE IMPACT OF VARIABILITY IN AADL ERROR MODELING

In addition to the impact of product and usage context variants on architectural and behavioral models in conventional SPL development, such variation may be propagated throughout dependability/error modeling in safety-critical SPLs. Then SPL architectural and behavioral variation, defined in product and usage context variants, can be further propagated throughout the SPLE safety lifecycle, impacting on HARA, and allocation of safety requirements (i.e., Functional-Safety Concept in ISO 26262). Fig. 1 illustrates the impact of variation in Tiriba SPL design (i.e., Fig. 1b) and behavioral (i.e., Fig. 1c) models and usage context (Fig. 1a), on product line HARA and allocation of safety requirements (Fig. 1d), and their propagation throughout component failure modeling (Fig. 1e), Fault Tree Analysis (FTA) and FMEA (Fig. 1f) dependability-related activities required to achieve safety certification.

Understanding how variation in product and usage context features impact on dependability analysis contributes to achieve the systematic reuse of both design and error models, reducing the certification costs of individual SPL products. Combinations among product and usage context variants may be useful to guide error modeling in safety-critical SPL architectures. In this way, different failure conditions can lead to system-level failures (*hazards*) with different probability, severity, criticality levels, and different safety requirements can be allocated to avoid/minimizing hazard effects on the overall safety according to design choices and targeted contexts. Safety requirement is the required risk reduction measure associated with a given system or component failure.

In AADL error modeling, can be specified in the following ways: *a)* via component *inheritance*, or *b)* with the support of a variant management tool. By using component inheritance, variant-specific HARA or component failure information can be specified in an Error Annex declaration in the corresponding variant-specific system/software/hardware component implementation. On the other hand, variation in HARA and component fault modeling can also be specified in multiple Error Annex declarations in a single component implementation. Later, a variant management tool can be used to support the specification of mappings linking product/context features to their realization on HARA and component failure information stored into AADL Error Annex declarations inside of component implementations. Details about *AADL Error Annex* modeling language are available elsewhere [4]. Dependable-related variability types and their specification in an AADL error model are detailed in the following.

4.1 Variability in HARA

During hazard analysis, different *hazards* and *hazard causes* can emerge according to the targeted product and usage context variants (Fig. 1d). Figure 7 illustrates an example of the usage of AADL component inheritance for specifying variability in HARA information associated with TFC-MAT product variant. TFC-MAT variant-specific HARA information is stored in an *Error Annex* declaration inside of *Base_FlighControlSystem* implementation. The causes for *no pilot commands* hazard, specified in an AADL composite error behavior (lines 6-7), are omission failures in *FailSafeController* and *PWMDecoder* component outputs when TFC-MAT product and *controlled* airspace context variants are chosen (Fig. 1a). On the other hand, omission failures in *FailSafeController*, *ModeSwitcher*, and *PWMDecoder* component outputs are the causes for the occurrence of the same hazard when TFC-ALL variant and *uncontrolled* airspace context are selected. Thus, by adopting the component inheritance strategy for variability modeling in the AADL error model, TFC-ALL variant-specific HARA information are stored into *Ext_FlightControlSystem* error model.

```
01 system implementation Base_FlightControlSystem.fcs
02 annex emv2 {**
04   composite error behavior
05             states
06               NoPilotCommands: [pwd.NoValueState and
07               fsc.NoValueState]-> NoValueState;
08   ...
16   end composite;
17   properties
18      EMV2::Hazards =>([failure => "Loss of UAV pilot cmds ";
19                   likelihood => 10e-9;
20                  severity => "Hazardous";
21                  developmentAssuranceLevel => A;])
22      applies to NoValueState;
23   ...
58 end Base_FightControlSubSystem.fcs;
```

Figure 7: Base_FlightControlSystem AADL error model.

In risk assessment, variation can be found in *probabilistic criteria*, e.g., *likelihood* and *severity* in SAE ARP 4754A avionics standard, used to classify the risk posed by each system hazard for the overall safety. In Tiriba SPL risk assessment, different risk probability and severity values were assigned to *no pilot commands* hazard, according to the targeted product and usage context variants. Thus, the probability of occurrence of an omission of pilot commands, i.e., *no pilot commands* hazard, is 10^{-9} per hour of operation, with a *catastrophic* severity (lines 19-20 from Figure 7) when TFC-MAT and *controlled airspace* context variants are chosen. Hazard severity and probability values are stored in the EMV AADL Property element. On the other hand, the probability of occurrence of this hazard is 10^{-7} per hour of operation with a hazardous *severity* when TFC-ALL product and *uncontrolled airspace* context variants are chosen. Such information is stored into *Ext_FlightControlSystem* error model.

Still in HARA, after classifying the risk posed by each identified hazard and contributing component failure mode, variation can be found in the allocation of *functional safety requirements* and *Safety Integrity Levels* (SILs) to mitigate the effects of system or component failures on the overall safety. Variation in *functional safety requirements* can be found in architectural decisions that must be taken to eliminate or minimizing the effects of a system or component failure, which might change according to the targeted product and usage context variants. For example, in the Tiriba UAV product line, the control system architecture can be *Redundant* when *controlled airspace* usage context is chosen. A *non-redundant* architecture can be adopted when the UAV is intended to operate in an *uncontrolled airspace* [2].

Variation in the allocation of SILs relates to the variation on the mitigation mechanisms to handle the risk posed by a given system hazard, component failure mode, or component, which might change according to the targeted product and usage context variants. This may impact on the SPL development process, in which different system engineering activities, e.g., verification, validation, and testing should be carried out to address the targeted level [3]. Process-oriented safety standards, e.g., DO-178C and ISO 26262, define a set of safety objectives that should be addressed per SIL. Safety objectives define a set of activities to be performed and artefacts to be produced. DO-178C defines five levels of integrity named *Development Assurance Levels* (DALs). Level A is the highest stringent integrity, and level E is the less stringent. ISO 26262 automotive standard also defines five levels of safety integrity where QM is the less stringent and level D is the stringent. Addressing higher stringent SILs demand the most stringent safety objectives, system engineering activities, and software artefacts, increasing the development costs. Allocating less stringent SILs to less-critical SPL components and more stringent SILs only to highly critical components can contribute to reduce the SPL development costs.

An example of variation in the safety integrity levels is observed in the allocation of DALs to mitigate the effects of *no pilot commands* hazard, during the Tiriba SPL dependability analysis (see Fig. 1d).

DAL allocation variability is specified in the values for developmentAssuranceLevel *EMV* property in variant-specific FCS component implementations as illustrated in Figure 7 line 21. Thus, DAL A should be assigned to mitigate the effects of this hazard in TFC-MAT variant operating in a *controlled airspace* as illustrated in the line 21 from the *Base_FlightControlSystem* AADL error model (Figure 7). On the other hand, DAL B should be assigned to mitigate the effects of this hazard in TFC-ALL variant operating in an *uncontrolled* airspace. Such variation may be further propagated throughout the decomposition of SILs allocated to hazards throughout contributing failures modes and components. It is important to highlight that when developing reusable components, all variability aspects of a component should be considered from the initial stages of the SPL lifecycle, and the most stringent SIL assigned to that component in different contexts should be assigned to that component to ensure its safety usage across the SPL. Thus, product and contextual variability will not change the mitigation mechanisms for that component in specific product variants.

4.2. Variability in Component Fault Modeling.

In the safety lifecycle, component fault modeling is intended to identify how components contribute to the occurrence of potential system-level failures identified during hazard analysis. Variation in component fault modeling can be found in *component output deviations* that contribute in some way for the occurrence of system failures, which might change from a targeted product/context variant to another. Different *input deviations*, *internal failures*, or combinations among them specified in a *failure expression*, which contribute to the occurrence of a given *output deviation*, may also be raised in different product/context variants. Variation in the design, i.e., in architecture and behavior, may also impact on how component failures *propagate* throughout other components. Thus, *output deviations* of a given component may be propagated throughout different components according to the chosen product/context variants. Different product/context variants lead to different connections among components, via their ports, so it may change the way in which failures are propagated throughout the system architecture.

Variation in HARA (Fig. 1d) can be propagated throughout how components contribute to the occurrence of system hazards (Fig 1e). The Tiriba *Basic Command Processor* (BCP) component fault modeling, specified in Error Annex declarations stored into base and extended BCP system component implementations, are examples of variation in *output deviations* and combinations among *component failures* leading to *output deviations* in TFC-MAT and TFC-ALL product variants. Base BCP AADL error model, shown in Figure 8, represents the realization of TFC-MAT variant the AADL error model. *Omission-autopilotSettings* output deviation may be raised when TFC-MAT product and *controlled* airspace context variants are chosen, whilst one additional output deviation, i.e., *omission-mode*, can also be raised when TFC-ALL product and *uncontrolled* airspace context variants are chosen. Such variation can also be propagated throughout the causes that lead to the occurrence of a given output deviation. The causes of an output deviation can be stated in terms of a failure expression using logical operators (AND, OR, NOT) that describe how combinations among internal and input failures of a component may lead to the occurrence of an output deviation. Such variation can be found in the causes of the *Omission-autopilotSettings* component output in both TFC-MAT and TFC-ALL product variants. An internal omission failure in BCP component or an omission in one of BCP inputs can raise an *Omission-autopilotSettings* output (lines 20-22 from Figure 8) in the TFC-MAT variant. Conversely, an internal omission failure in BCP or omissions in both BCP inputs may

```
01 system implementation Base_BasicCommandProcessor.bcp
02   annex emv2 {**
03     error propagations
05       autopilotSettings: out propagation {NoValue, BadValue};
06       sensorData: in propagation{NoValue, BadValue};
07       flows
08        OAP: error source autopilotSettings{NoValue}when Failed;
09        OSD2AP: error path sensorData{NoValue} ->
10        autopilotSettings{NoValue};
13    end propagations;
14    component error behavior
15        events
16          OmissionError: error event;
19          transitions
20            OFailure1: Operational-[OmissionError OR
21             (basicCommand{NoValue} OR
22             sensorData{NoValue})]-> NoValueState;
26          propagations
27             OAP: Operational -[basicCommand{NoValue} OR
28              sensorData{NoValue} ]-> autopilotSettings{NoValue};
31    end component;
32  **};
33 end Base_BasicCommandProcessor.bcp;
```

Figure 8: Base BCP AADL error model.

raise an *Omission-autopilotSettings* in the TFC-ALL variant. Such variant information is stored in an *Error Annex* declaration inside of the extended BCP implementation, not illustrated in this paper. Additionally, variation in AADL error models for base and extended BCP system component can be found in the specification of *error propagations*, which the extended BCP error model comprises an additional *cmode out propagation* of *no value* and *bad value* error types. Such variation is further propagated throughout error *flows*. Thus, the extended BCP error model comprises two additional *no value* and *bad value error source* and two additional *error paths* associated with the *cmode* output not present in the base BCP AADL error model shown in Figure 7.

Probabilistic risk criteria values, e.g., *likelihood*, *severity*, *failure and repair rates*, which can be assigned to a given AADL execution platform component, may also vary according to the targeted product and usage context variants. An example of such variation was found in the assignment of failure rates to *Barometric Processor* hardware component from the Tiriba UAV-SPL. The component failure rate is 10^{-9} per hour of operation when TFC-MAT variant is chosen, and 10^{-7} per hour when TFC-ALL variant is selected. In the TFC-SPL AADL error modeling, such variation is encapsulated in error models specified for base and extended *Barometric Processor* device component implementations. Each *Barometric Processor* error model stores different values for risk probabilistic criteria associated with such device in EMV2 *Properties*, in the same way for hazard analysis in lines 18-21 from Figure 7. It is important to highlight that AADL component fault modeling is performed by specifying *error propagations*, i.e., component input/output *error propagations*, *error source*, *sink*, and *paths*, and by specifying the *component error behavior*, *i.e*., error *events*, *transition* events, and *propagation* of errors from input to output ports.

Variation in component fault modeling can be further propagated throughout *fault trees and FMEA* artefacts, which can be automatically generated, with the support compositional techniques [[4][23][26] from HARA and component fault modeling information. Such variation can be seen in fault tree gates and nodes (Fig. 1f), and in the way how components contribute to hazards in a FMEA table in different product variants[24]. Thus, variation in hazard causes can be seen in the structure of a fault tree as shown in Fig. 1f, in which *PWMDecoder* and *FailSafeController* component output deviations are top-level failures of *no pilot commands* fault tree when TFC-MAT variant is chosen. On the other hand, an output deviation in *ModeSwitcher* component together with other two aforementioned output deviations, are the top-level failures of *no pilot commands* fault tree when TFC-ALL product variant is chosen. Finally, variation in HARA, allocation of safety requirements, component fault modeling, fault trees and FMEA are propagated throughout the structure of a variant-specific assurance case. An assurance case is a defensible, comprehensive, and justifiable argument supported by evidence which demonstrates that the system acceptably safe to operate in a particular context [18]. Assurance case is recommended by standards for certifying safety-critical systems in both automotive [14] and aerospace [8] domains. The analysis of the impact of variation in assurance case is beyond the scope of this paper.

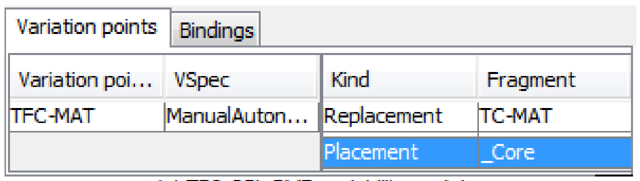
After performing AADL design and error modeling, the BVR toolset and the BVR AADL/Error Annex developed by the authors have been used to specify the TFC-SPL variability model. In this paper, we have adapted the BVR toolset [32] to enable support for variability management in OSATE AADL Behavior and Error Annex error models. The BVR adapter extends OSATE AADL model editor to enable BVR communicating with OSATE model editors to manage variability in AADL structural/behavioral/error models. Since BVR is built upon the Eclipse Modeling Framework [7], the adapter was implemented as an Eclipse-based plugin. Due to page limitation, BVR AADL adapter is not discussed through this paper.

The DEPendable-SPLE variability modeling steps [24] were performed by the authors with the support of BVR and BVR AADL adapter. Thus, mappings linking TFC-SPL features and their realization in AADL design, and error models were specified. Details about how to configure a BVR variability model for a given SPL can be found elsewhere [32]. BVR fragment substitutions were defined to show how variability in the TFC-SPL AADL architecture and Error Annex models are solved when TFC-MAT/*Controlled* and TFC-ALL/*Uncontrolled* structural and dependable-related variants are chosen. Table 3 shows placement and replacement fragments, and fragment substitutions associated with TFC-MAT/*Controlled* airspace dependable-related variant. The acronyms in table columns respectively represent *variation point* (*VP*), *fragment substitution* (*FS*), *fragment type* (*F*), and *component failure data* (*CFD*).

Table 3: Pilot mode/usage context variant and its realization in the AADL error model.

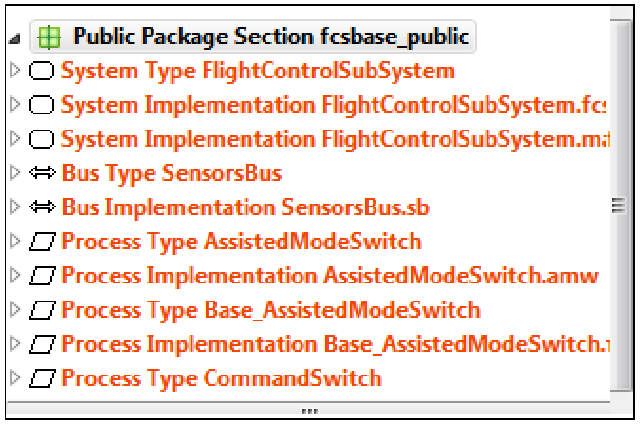
VP	FS	F	HARA Results	CFD
Pilot Mode/ Airspace	TFC-MAT/ CONTROLLED	P	MAS-NoPilotCommands, MAS-ValuePilotCommands, MAP-NoPilotCommands, MAP-ValuePilotCommands, ALL-NoPilotCommands, ALL-ValuePilotCommands	ALL-BCP, MAS-BCP, MAS-PWD
		R	MAT-NoPilotCommands, MAT-ValuePilotCommands	MAT-BCP, MAT-MS, MAT-PWD

An excerpt of mappings linking TFC variants to their realization in the AADL error model is shown in Figure 10a. So, when TFC-MAT variant and its context are selected, as defined in a placement fragment, HARA results and component failure data associated with other TFC variants are removed from the AADL error model (Figure 10b). On the other hand, as specified in the replacement fragment, TFC-MAT related HARA results and component failure data are included in the final dependability model. Finally, TFC-MAT fragment substitution is created by combining both placement/replacement fragments. The final TFC-SPL variability model comprises 2 fragment substitutions, 1 placement, and 2 replacement fragments. The two fragments substitutions are associated with AADL structural and behavioral, error model. The assets were input BVR during product derivation: TFC feature and instance models, variability model, and AADL design/error models. BVR has generated TFC-MAT and TFC-ALL AADL architectural, behavioral and error models. Variability management in TFC-SPL enabled the systematic reuse of almost 100% of the AADL product line design and error models produced in domain engineering, in application engineering.



Variation poi…	VSpec	Kind	Fragment
TFC-MAT	ManualAuton…	Replacement	TC-MAT
		Placement	_Core

(a) TFC-SPL BVR variability model



Public Package Section fcsbase_public
System Type FlightControlSubSystem
System Implementation FlightControlSubSystem.fc
System Implementation FlightControlSubSystem.m
Bus Type SensorsBus
Bus Implementation SensorsBus.sb
Process Type AssistedModeSwitch
Process Implementation AssistedModeSwitch.amw
Process Type Base_AssistedModeSwitch
Process Implementation Base_AssistedModeSwitch.
Process Type CommandSwitch

(b) TFC-SPL AADL model

Figure 10: An excerpt of TFC-SPL design and variability models in OSATE/AADL & BVR toolset.

4 RELATED WORK

Research on variability management in model-based development is split into two areas: variability management in design assets and variability management in dependability assets. In the first category, Shiraishi [30] has presented an approach to support variability modeling in the AADL structural/architectural model. The approach is built upon AADL inheritance and refinement reuse mechanisms applicable to component definitions and implementations. Shiraishi approach was applied in the variability modeling of an automotive cruise control system. Shiraishi approach only provide support for variability in the AADL architecture model, not covering variability modeling in AADL behavior and error modeling as presented in this paper. Thus, in this paper we have extended Shiraishi approach to enable support for variability modeling and management in AADL behavior/error models. In the second category, Schulze *et al.* [29] have proposed an approach, by integrating commercial *Medini* ISO 26262 safety analysis and *pure::variants* tools, to support variability management in functional safety-related assets, which was evaluated in an automotive case study. Their approach is based on a referencing model, which maps problem-domain features with artefacts in the solution space, in this case, requirements, FTA, and safety goals. Schulze *et al.* approach was further extended with a process for model-based change impact analysis of variability into automotive functional safety [16]. This process combines variability management techniques with safety engineering and software configuration management to achieve a complete safety assessment. In the same way as Schulze *et al* approach and its extension, the proposed variability modeling/management approach is built upon the variability model, linking problem-domain context and product features with artefacts from the solution space, targeting open source AADL design and error models. Although Schulze *et al.* [16] approach provides support for variability management in functional safety, they didn't emphasize the management of the impact of design and contextual variation in error modeling and not cover how variability can be specified in the error model, as presented in this paper. In addition, our approach is applicable to domains other than automotive. Nevertheless, Schulze *et al.* approach [29] and its extension [16] also presented a good and efficient solution for variability/change management in functional safety. Kaßmeyer *et al.* [17] presented a systematic model-based approach integrated to the change impact analysis approach [16]. Kaßmeyer *et al.* combines requirements engineering and architectural design, dependability analysis, and variant management tools, allowing seamless safety engineering across product variants by representing safety artefacts in a UML-compliant model notation. In such approach, HARA and component fault modeling is performed by annotating the UML model in the same way as the proposed AADL design and error modeling approach. In this paper, we presented how variability can be specified in AADL architecture, behavior and error modeling.

5 CONCLUSIONS

This paper has presented a novel approach that extends existing AADL architectural variability approaches [30] with the support for variability modeling in the AADL behavioral and error modeling. Such approach enabled the systematic reuse of both AADL design and error models in application engineering. The approach was applied with the support of OSATE AADL and BVR toolsets, and OSATE AADL BVR adapter developed by the authors. In the proposed approach, component inheritance and refinement, strategies were adopted to specify variability in AADL behavior and error models. Later, BVR tool and its adapter were used to manage variability in reusable AADL design and error models. From the analysis of the TFC-SPL AADL variability modeling, the authors have identified that the usage of AADL component inheritance is a more suitable approach to specify variability in AADL behavioral and error models than the specification of variation in multiple behavior/error annex specifications in a single component implementation. Thus, this strategy contributes to increase the size and complexity of AADL architecture, behavioral and error models of a safety-critical SPL, which may contribute to increase the complexity of variability management. The adoption of component *inheritance* for specifying variability in the AADL design and error models reduced in 50% the complexity of the Tiriba SPL variability model created with the support of the BVR toolset.

The proposed variability modeling/management approach enabled the systematic reuse of almost 100% of the reusable Tiriba AADL design and error models during the product derivation process. Thus, reducing the effort and costs for specifying variant-specific AADL design and error models, and enabled the automated generation of variant-specific dependability artefacts, e.g. fault trees, FMEA, and reliability block diagram, with the support tools integrated into OSATE such as EMFFTA for fault tree analysis. Further work intends to investigate the impact of design/context variations on SIL allocation and development processes of safety-critical SPLs. Further work also intend to investigate the usage of model-driven techniques [12] to generate variant-specific assurance cases from OSATE AADL design and error models, and the potential of SIL decomposition techniques in supporting SPL architectural decisions.

REFERENCES

[1] Big Lever. 2016. Gears. Available on-line: http://www.biglever.com.

[2] Braga, R.T.V., Trindade Jr., O., Branco, K. R. L. J. C., Lee, J. 2012. Incorporating certification in feature modelling of an unmanned aerial vehicle product line. In: *Proc. of the 16th SPLC*, Salvador, Brazil, 1–10.

[3] Braga, R.T.V., Trindade Jr., O., Branco, K. R. L. J. C., Neris, L.O., Lee, J. 2012. Adapting a software product line engineering process for certifying safety critical embedded systems. In *Proc. of the 31st* SAFECOMP, Springer,352-363.

[4] Delange, J., Feiler, P. 2014. Architecture fault modeling with the AADL error-model annex. In *Proc. of the 40th EUROMICRO*, Verona, 361-368.

[5] Domis, D., Adler, R. Becker, M. 2015. Integrating variability and safety analysis models using commercial UML-based tools. In *Proc. of the 19th SPLC*, ACM, USA, 225-234.

[6] Dordowsky, F., Bridges, R., Tschope, H., 2011. Implementing a software product line for a complex avionics system. In *Proc. of the 15th Int. SPLC*, IEEE, 241-250.

[7] ECLIPSE. 2016. Eclipse Modeling Framework Project. Available on-line: http://www.eclipse.org/modeling/emf.

[8] EUROCAE. 2010. ARP4754A - guidelines for development of civil aircraft and systems, EUROCAE.

[9] EUROCAE. Aircraft wheel braking system. Available on-line: https://github.com/osate/examples/tree/master/ARP4761

[10] Habli, I., Kelly, T., Hopkins, I. 2007. Challenges of establishing a software product line for an aerospace engine monitoring system. In *Proc. of the 11th SPLC*, IEEE, Japan, 193-202.

[11] Haugen, O., Moller-Pedersen, B., Oldevik, J., Olsen, G. K., Svendsen, A. 2008. Adding standardized variability to domain specific languages. In *Proc. of the 12th Int. Software Product Line Conf.*, Limerick, Ireland, IEEE, 139-148.

[12] Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T., 2015. Weaving an assurance case from design: a model-based approach. In Proc. of the 16th HASE, Daytona Beach,, IEEE, p.110-117.

[13] Heuer, A., Pohl, K. 2014. Structuring variability in the context of embedded systems during software engineering. In: *Proc. of the 8th Workshop on Variability Modelling of Software-intensive Systems*, ACM.

[14] ISO. 2011. ISO 26262: road vehicles functional safety.

[15] Kang, K. C., Kim, S., Lee, J., Kim, K., Jounghyun Kim, G., Shin, E. 1998. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Eng.*, v. 5, 143–168.

[16] Käßmeyer, M., Schulze, M., Schurius, M. 2015. A process to support a systematic change impact analysis of variability and safety in automotive functions. In *Proc. of the* 19th *Int. Software Product Line Conf.*, ACM, NY, USA, 235-244.

[17] Käßmeyer, M., Moncada, D. S. V., Schurius, M. 2015. Evaluation of a systematic approach in variant management for safety-critical systems development. In *Proc. of the 13th Int. Conf. of Embedded and Ubiquitous Comp.*, IEEE, 35-43.

[18] Kelly, T., MCdermid, J., 1997. Safety case construction and reuse using patterns. In: Proc. of the 16th SAFECOMP, Springer-London, LNCS, p. 55–69.

[19] Krueger, C. 2002. Variation management for software production lines. In: *Proc. of the 2nd Int. SPLC*, San Diego, USA, Springer, LNCS, v. 2379, 37–48.

[20] Lasnier, G,, Pautet, L., Hugues, J., Wrage, L. 2011. **Animplementation of the behavior annex in the AADL-toolset Osate2**. In: Proc. 1*6th IEEE Int. Conference on Engineering of Complex Computer Systems*, Las Vegas, NV, pp. 332-337.

[21] Lee, K., Kang, K. C. 2010. Usage context as key driver for feature selection. In *Proc. of the 14th SPLC*. Springer, Heidelberg, LNCS, v. 6287, 32–46.

[22] Leveson, N. Door control system. Available on-line: https://github.com/osate/examples/tree/master/Train

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

[24] Oliveira, A. L., Braga, R. T. B., Masiero, P. C., Papadopoulos, Y., Habli, I., Kelly, T. 2014. Variability management in safety-critical software product line engineering. In *Proc. of the 17th ICSR, LNCS, Madrid, Spain, 2018(to appear)*.

[25] Oliveira, A. L., Braga, R., Masiero, P. C., Papadopoulos, Y., Habli, I., Kelly, T. 2016. Model-based safety analysis of software product lines. *International Journal of Embedded Systems*, Inderscience Publishers.

[26] Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann. 2011. Engineering failure analysis and design optimization with HIP-HOPS. *Journal of Eng. Failure Analysis*, v.18, i.2, 590-608.

[27] RTCA. 2012. DO-178C software considerations in airborne systems and equipment certification..

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

[29] Schulze, M., Mauersberger, J., Beuche, D. 2013. Functional safety and variability: can it be brought together? In *Proc. of the 17th Int. SPLC*, ACM, NY, USA, 236-243.

[30] Shiraishi, S. 2010. An AADL-based approach to variability Modeling of automotive control systems. In: Proceedings of the MODELS, D.C. Petriu, N. Rouquette, Ø. Haugen (Eds.), LCNC, Vol. 6393. Springer, Heidelberg, 346-360. DOI: http://dx.doi.org/10.1007/978-3-642-16145-2_24.

[31] Steiner, E. M., Masiero, P. C., Bonifácio, R. 2013. Managing SPL variabilities in UAV Simulink models with Pure::variants and Hephaestus. *CLEI Electronic Journal*, v. 16, n.1, 1-16.

[32] 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.