Model-based systems product line engineering with physical design variability for aircraft systems

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Citation: LI, M. ... et al., 2016. Model-based systems product line engineering with physical design variability for aircraft systems. IN: Proceedings of 11th 2016 IEEE International Conference on System of Systems Engineering (SoSE 2016), Kongsberg, Norway, 12-16 June 2016.

Additional Information:

- © 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Metadata Record: https://dspace.lboro.ac.uk/2134/21652

Version: Accepted for publication

Publisher: © IEEE

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the published version.
Abstract— Software Product Line Engineering (SPLE) has drawn large amounts of attention during the last two decades as it offers the benefits of reducing cost and time to market by reusing requirements and components. Recently, more and more large scale industries start to implement SPLE in their domains (combining SPLE with model-based modelling methods). However, the problem of how to combine SPLE with Model-Based System Engineering is still a challenge, as systems are much broader than the software domain. Unlike software engineering, system engineering has to consider the physical resources aspect. This paper classifies typical types of physical variability and provides general modelling solutions for each type of physical variation at the system design stage. Specifically, this approach combines a variability model with a SysML Block Definition Diagram and an Internal Block Diagram to model the contextual variability, architectural variability, connector variability, instance number variability, component variability, location variability and evolutional variability of physical designs. Variability is modelled separately to help reduce the complexity of design models. Last but not least, the proposed method is illustrated by an aircraft system case study.

Keywords— Product Line; Model-Based Systems Engineering; Cyber-Physical Systems; Variability Modelling.

I. INTRODUCTION

Product Line Engineering (PLE) is a reuse-based development method that has shown promising results for reducing both cost and time to market by considering the commonality and variability of systems [1]. PLE publications over the last 20 years have mainly concentrated on the software domain [2-6], therefore PLE, as it has so far been referred to, should actually be called Software Product Line Engineering (SPLE). A well-known definition for a Software Product Line is given by [7]: “A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” SPLE is responsible for developing a family or repository of products, which is called the “product line” [8]. After the creation of the product family, each piece of target software that is called a “product”, as defined by [8], is derived from the product line according to specific requirements. There are two development processes in PLE [9] (adopted as part of ISO/IEC standard #26550):

- Domain engineering deals with defining the commonality and variability of a product line in different development stages and includes: requirements engineering, designing, implementation and testing for reusability concerns. The outcome of domain engineering is a domain artefact, also called a core asset [8].
- Application engineering handles the derivation of concrete products from domain artefacts by binding variants according to specific customers’ requirements.

Due to the promising benefits of SPLE for reducing development costs and shortening development times, it has also gained significant attention in recent years in both academia and industries in the systems domain. The author Queiroz [10] proposed a systematic review on developing Critical Embedded Systems with PLE in 2014. Dumitrescu [11] and Góngora [12] both introduced methods to combine SPLE with Model-Based System Engineering (MBSE) in an industrial context. While Software Product Line Engineering is important in the systems domain, System Engineering is a much broader discipline; it includes the consideration of hardware aspects (mechanical and electrical engineering) as well as the overall integration of hardware and software components (for cyber-physical systems) [13]. However, as previously stated, current PLE methods mainly focus on the software domain. These methods lack physical variability representation, or do not differentiate physical variability from functional variability [10]. Indeed, the challenge of combining SPLE with MBSE is mainly about how to represent physical variability.

Therefore the aim of this paper is to classify major types of physical design variability and propose a practical physical variability modelling method that can be combined with model-based system design models in SysML. Previously, we proposed a variability modeling framework [17] for separating different types of variability, such as functional, physical and quality, at the requirement engineering stage. This paper further details how to represent physical variability at the design stage.

The rest of the paper is structured as follows: Section II classifies typical types of physical variability that exist in the aircraft systems domain; Section III introduces the proposed modelling approach framework in terms of its meta-model and
further illustrates it with a case study of the aircraft systems family. Finally, a conclusion to this paper and information about future work are provided.

II. CLASSIFICATION OF PHYSICAL VARIABILITY SOURCES

This section introduces a simplified case study from aircraft system design to illustrate where typical types of physical variability occur. According to Moir and Seabridge [14], the aircraft system can be described as a system of systems that integrates sub-systems to perform a particular role or set of roles. Each sub-system can be a viewpoint or a system for a different stakeholder or organisation. In general, the aircraft system consists of general systems, avionic systems and data buses. Several variabilities of the aircraft system are ignored here for the sake of simplicity. The general systems are classified by grouping the fuel system, electrical power system, secondary power system, propulsion system, flight control systems, landing gear, engine control system and environmental control system to provide a source of power to the air vehicle. Avionic systems consist of communication systems, navigation systems, automated landing systems, air data measurement, flight management systems, displays and controls. These provide basic navigation, communication, aircrew display and control functions. All sub-systems are connected by data buses for communication and data exchange.

However, in reality, aircraft systems are not always the same. They have numerous variabilities, even when they are manufactured by the same company. The general classifications of these variabilities are listed in Table 1. The most common variability type is components varying between different systems. For example, the level sensors of the fuel system are used to measure the fuel level in a particular tank in order to inform fuel management decisions [15]. There are various types of level sensors: float level sensors, zener diode level sensors and ultrasonic sensors (used in the Boeing 777). Beside that, the Central Processing Unit (Intel 4004, Intel 8086, Motorola M68020 and PowerPC, detailed by [16]), memory (D-RAM, S-RAM, ROM, EPROM, EEPROM and Flash [16]), etc. that configures aircraft systems can also be different. The second type of variability is the number of instances of the components. For instance, gas turbine engines may consist of two shafts, a high pressure shaft and a low pressure shaft (this is typical of Pratt & Whitney and General Electric designs [15]), or of three shafts (Rolls Royce turbofan designs). The third type of variability is connector variability. As introduced by [16], in aircraft systems, components such as sensors and computers are connected via networks. There are numerous digital data bus technologies in today’s aircraft, including ARINC 429, MIL-STD-1553B, ARIC 629, AIRNC 664 (AFDX) and CANbus. They are different in terms of data transmission rate and message format. As the development of digital computing technology improves, different data buses and components (as introduced previously) keep evolving, increasing the performance and complexity of aircraft systems.

The configuration or selection of these available options also impacts a system’s architecture and leads to the evolution of systems. These variabilities are grouped into two types: architecture variability and evolution variability. One simplified example is shown in Figure 1 to illustrate how avionics architecture vary and evolves over time. For distributed analogue architectures, the major units are interconnected by hardwiring instead of data buses [14]. Every function is a “point-solution”, implemented by analogue electronics and relays [16]. Typical aircraft with this architecture includes the Boeing 707, VC10 and KC 135, which are still flying. During the 1970s, the maturity of digital systems lead to analogue devices being replaced by digital computers [16]. Functionalities were performed by application software running on digital computers [16]. ARINC 429 (although slow by today’s standards at 110kbps for ARINC429 [14]) was used for connections between major units. The advanced technology in high-speed data buses such as ARINC 429, MIL-STD-1553B and ARIC 629 boosted the occurrence of federated architecture during the 1980s. This type of architecture used inter-bus bridges rather than connecting every component individually. Computers (Line Replaceable Units) were functionally connected but physically discrete [16]. The final advance occurred with the maturity of Commercial-Off-The-Shelf (COTS) technology around the 1990s. Instead of federated computers, integrated modular avionics principles promoted a general-purpose, centralised computing resource comprising a set of common hardware computing modules [16]. The fifth variability type is location variability, which means systems can be different due to components being mounted at different locations. As illustrated in [15], thermocouples can be located on the rear of the engine (gas stream) or mounted on the engine casing to measure engine temperature. The sixth variability type is context variability, which distinguishes systems components using the environment. For example, an aircraft can be either a commercial aircraft or a military aircraft. Commercial aircraft require passenger cabins, passenger seats and entertainment equipment. Military aircraft are equipped with defensive aids and armaments. The last variability is evolution variability that represents how system components, architectures and locations vary over time.

![Figure 1. Evolution of Avionic System Architectures [16]](image)
III. PHYSICAL VARIABILITY MODELLING FRAMEWORK

This section introduces the proposed method for modelling physical design variability for aircraft systems. It extends our previous work [17] with more precise classification of physical variability types and illustrates how to represent hardware variability modelling at the design stage. In addition, the proposed variability modelling method is combined with SysML diagrams to represent hardware architecture and designs artefacts.

The meta-model of the proposed method is shown in Figure 2. In software engineering, a meta-model is a mechanism for representing a well-formed formula or the abstract syntax of a modelling language [18]. The newly extended concepts are highlighted in dark grey on the diagram. The top of Figure 2 shows the variability modelling method, whilst the bottom section illustrates how the variability model links with different types of physical design variability artefacts (also called base models) for hardware design.

In general, variability models are used to distinguish and manage variability. They provide a separate view of variable system designs. Different types of variation point define where a certain type of variability occurs. The proposed variability modelling method has the advantage of reducing the effort of introducing variability into existing documents by only representing variable parts, rather than both commonality and variability. Design artefacts are used to represent hardware architecture and components. They describe both the commonality and the variability of system designs. The existing legacy of system design models is not changed; variable designs are introduced by adding variable artefacts (refinements of each variant in the base model) into existing models.

A. Variability Modelling

The definition and semantics of the extended concepts are illustrated below. Explanations of other meta-classes can be found in [17].

- The “Physical Variability” meta-class further specifies the physical “Type” meta-class of a “Variation Point” in more detail. Each of these classifications is explained precisely in Section II, “Context Variability”, which classifies physical variabilities by their environments, is implemented by the tag “Physical->Context Variability” of “Type” stereotype at the Variation Point. Other types of physical variability are also represented in the same way; for example, the tag “Physical->Architecture Variability” of “Type” stereotype represents architecture variability.

- The “Priority” meta-class further extends the “Binding Time” meta-class to illustrate when a certain variation point should be decided and which point should be instantiated first. Different types of variability may occur at any level of design. For example, Architecture Variability can take place in general aircraft systems architecture design or at the bottom level, component architecture design. Using priority information facilitates the further representation of the hierarchical relationship of blocks in “Variants” on variability diagrams. Normally, “Context Variability” should be decided first, as it is a higher level of abstraction of systems and impacts the selection of numerous lower level components. Therefore, the tag “Priority” of “Binding Time” should be assigned a higher value. This is followed by “Architecture Variability” of high-level abstraction blocks, then “Connector Variability”. The fourth is “Component Variability” and the last is “Location Variability”. However, as illustrated in Section II, sometimes the selection or occurrence of new, more advanced components or connectors may influence architecture, therefore the priority of Architecture Variability may be lower than that of Connector or Component Variability in some cases. It mainly depends on physical dependencies and constraints, which can be represented by “require” and “exclude” relationships in variability models. The analysis of physical dependencies and constraints is not within the scope of this paper.

- The “Time in Use” attribute of “Variant” is used to describe their valid periods. This represents “Evolutional Variability” as evolutional variability is within the scope of this paper.

B. Physical Design Modelling

Traditionally, Model-based System Engineering implies the use of SysML Block Definition Diagrams to represent the hierarchical structure of systems and Internal Block Diagrams...
for modelling interconnections between internal parts of blocks. The proposed method adopts this method as it is standardised and introduces variability by adding variable designs to diagrams without affecting existing legacy models. Variants are recognised by the “Dependency on Artefacts” between artefacts “Block” and “Variation” in variability models.

- The “Scalability” meta-class refers to specific physical system designs at different levels of abstraction. The “Hardware Component Library” is the bottom level of abstraction and allows the reusability of individual hardware components across different projects. “Sub-system Level” groups instances of bottom-level blocks in terms of providing certain functions or consisting of certain sub-systems. Each artefact within this level can be reused as a sub-system for other products that require this type of sub-system. “Context Level” is used to identify interactions and relationships between sub-systems and external environments.

- The “Port” and “Item Flow” of SysML Block Diagrams are combined with “Variant” in the variability model to represent connector variability.

- The “NFR Artefact” refers to the representation of the variability of Non-Functional Requirements (Quality of Service attributes), such as the accuracy, weight and cost of components in terms of stereotypes. These stereotypes are implied by the different abstraction levels of “Blocks” which provide the metrics to select the suitable physical design components/solutions (trade-off analysis) or to estimate whether the target configuration satisfies system requirements. However, quality attributes trade-off analysis is beyond the scope of this paper.

C. Representing Different Physical Variability Views

This part of the paper illustrates how different types of physical variability are modelled in at Domain engineering. After domain models are generated in the Domain engineering stage, selections can be made from them to produce each target product.

1) Context Variability: The initial step of design variability modelling is generating the variability model. As Figure 3 shows, the variation point “Aircraft Type” is represented by triangles labelled “VP”. Its variants are “Commercial” and “Military” are represented by the label “V”. The cardinality “1…1” between these variants indicates that only one of them can be selected for the variation point. The <<VPExtension>> stereotype describes the extended concepts of the proposed method’s meta-model. For example, the “Aircraft Type” variation point belongs to “Physical-> Context Variability”. It should be decided by “domain engineers” at the design stage as a top priority. Its “Open” state shows that this variability can be modified in the future. Context variability artefacts are represented by SysML blocks in the SysML Block Definition Diagram at “Context Level”, the boundary of which is represented by the SysML package. “Dependency to Artefacts” (dashed arrow at bottom of diagram that links blocks in red) binds the block with its variant. The related lower-level components within that context are grouped by part properties and composite
relationships on the diagram. Therefore, decisions regarding context variants can directly impact the appearance or disappearance of artefacts and related artefacts.

2) Architecture Variability: Similarly, the type name “Physical->Architecture Variability” distinguishes this variability type. Architecture variability artefacts are modelled by different arrangements of blocks instances in the SysML Internal Block Diagram. Figure 5 shows two simplified “Avionic System Architecture” variants in SysML Internal Block Diagrams. The top part is the “Distributed Analog Architecture” and the bottom is “Federated Architecture”. Each Internal Block Diagram (white boxes with “[Block]”) is a variant and linked with variants in the variability model, as Figure 4 illustrates. Variants also link instance blocks (red boxes) that have been used within architectural variants.

3) Connector variability: Connector variability artefacts are illustrated by SysML “Port” and “Item Flow” concepts in Block Definition Diagram and Internal Block Diagrams. According to Friedenthal, Moore and Steiner [19], a port describes an interaction point on the boundary of a block. Two kinds of ports exist in SysML: flow ports and standard ports. It was decided for the purposes of this paper to use the flow port, as it can specify not only physical flow, such as water or fuel, but also the flow of information or control in electronic systems. In contrast, the standard port is used to represent services into or out of blocks. It should be used when considering functional (software) design allocations of physical resources. However, it is not discussed in this paper, as functional and physical design interactions are not within its scope. Normally, generalised flow specifications are used as artefacts that initiate connector variation points in design models. Each variant of that variation point can be modelled as a child flow specification of it. If any variant flow port is an atomic flow port [19], it is modelled as a flow property of the generalised flow specification variation point (a new block or value type that categorises the flow port variant).

4) Instance Number Variability: For instance number variability, each variant is represented via an individual composition dependency in a Block Definition Diagram, with different multiplicities. Similarly, if the instance number does not impact the architecture or internal structure of a system, every variant can be modelled by an instance of a block with specific multiplicities. If this is not the case, every variant should link with block artefacts that each represent one instance.

5) Component Variability: Component variability artefacts are represented via SysML blocks. The block is the fundamental modular unit of SysML, and defines a logical,
conceptual or physical entity. Variation points of this type of variability are generalised through Superclasses, which have the common properties of variants. Variations are modelled via the same block but different properties if their similarity is high; for example, sensors are often different in accuracy range. Otherwise, they are represented by different subclasses. The variable component blocks are saved in the bottom-level “Physical Component Library” package to provide reusability within different projects.

6) Location Variability: These artefacts are illustrated by placing block instances in the target environment or position blocks in Internal Block Diagrams (for instance, positioning a “Sensors” block within an “Engine” block). Variations are modelled by placing them on all possible locations, such as sensors on both “Engine” and “Airframe” blocks. Sometimes the frame box is used to further represent a specific location.

7) Evolutional Variability: This is represented via “Time in Use” tags on variants. It is different from variability in space, as discussed above, evolutional variability may occur in any type or asset of types of space variability. Figure 4 may be an example of avionic systems having evolved in terms of architecture, also including components and connections variability. Modelling systems evolution is important in the aircraft systems domain, as an aircraft may service and be maintained for several decades. Forward and backward compatibility, as discussed in [14], illustrates this consideration. For instance, a legacy system should be compatible with similar new systems for long-term maintenance and upgrading purposes. Specific usage duration variants can link all related artefacts.

IV. CONCLUSIONS AND FUTURE WORK

This paper has proposed a framework to combine Software Product Line Engineering with model-based cyber-physical systems designs for the aircraft systems domain. Specifically, it initially identified the major types of physical variability that exist in aircraft systems, which are evolutionary, contextual, architectural, connectional, instance number, component and location variability. A framework was proposed to represent variability separately in variability models with detailed physical type classification, stakeholder and binding time specifications. Compared to integrated variability modelling methods such as feature models [5], it reduces the complexity of models, as only variable parts are modelled. Variants of the model are linked with blocks, ports and item flows in SysML Block Definition Diagrams and Internal Block Diagrams to illustrate different variability views.

The challenge of combining functional (software) design variability and physical design variability with analysis of interactions is the next stage of this work. It also includes management of evolitional variability in further detail.

ACKNOWLEDGMENT

The author would like to thank the Rolls-Royce Controls and Data Services team, who are supporting the current research. Special thanks also go to Atego for providing a licence for the modelling tool Artisan Studio.

REFERENCES