Beyond Software Product Lines: Variability Modeling in Cyber-Physical Systems

Jacob Krüger
Otto-von-Guericke-University
Magdeburg, Germany
Harz University of Applied Sciences
Wernigerode, Germany
jacob.krueger@ovgu.de

Sebastian Nielebock
Otto-von-Guericke-University
Magdeburg, Germany
sebastian.nielebock@ovgu.de

Sebastian Krieter
Otto-von-Guericke-University
Magdeburg, Germany
sebastian.krieter@ovgu.de

Christian Diedrich
Otto-von-Guericke-University
Magdeburg, Germany
christian.diedrich@ovgu.de

Thomas Leich
Harz University of Applied Sciences
Wernigerode, Germany
tleich@hs-harz.de

Gunter Saake
Otto-von-Guericke-University
Magdeburg, Germany
gunter.saake@ovgu.de

Sebastian Zug
Otto-von-Guericke-University
Magdeburg, Germany
sebastian.zug@ovgu.de

Frank Ortmeier
Otto-von-Guericke-University
Magdeburg, Germany
frank.ortmeier@ovgu.de

ABSTRACT

Smart IT has an increasing influence on the control of daily life. For instance, smart grids manage power supply, autonomous automobiles take part in traffic, and assistive robotics support humans in production cells. We denote such systems as *Cyber-physical Systems (CPSs)*, where *cyber* addresses the controlling software, while *physical* describes the controlled hardware. One key aspect of CPSs is their capability to adapt to new situations autonomously or with minimal human intervention. To achieve this, CPSs reuse, reorganize and reconfigure their components during runtime. Some components may even serve in different CPSs and different situations simultaneously. The hardware of a CPS usually consists of a heterogeneous set of variable components. While each component can be designed as a software product line (SPL), which is a well established approach to describe software and hardware variability, it is not possible to describe CPSs' variability solely on a set of separate, non-interacting product lines. To properly manage variability, a CPS must specify dependencies and interactions of its separate components and cope with variable environments, changing requirements, and differing safety properties. In this paper, we *i)* propose a classification of variability aspects, *ii)* point out current challenges in variability modeling, and *iii)* sketch open research questions. Overall, we aim to initiate new research directions for variable CPSs based on existing product-line techniques.

SPLC '17, September 25-29, 2017, Sevilla, Spain

ACM ISBN 978-1-4503-5221-5/17/09...$15.00
https://doi.org/10.1145/3106195.3106217

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Software and its engineering** → **Software product lines**;

KEYWORDS

Software product line, Cyber-physical system, Modeling

ACM Reference format:
Jacob Krüger, Sebastian Nielebock, Sebastian Krieter, Christian Diedrich, Thomas Leich, Gunter Saake, Sebastian Zug, and Frank Ortmeier. 2017. Beyond Software Product Lines: Variability Modeling in Cyber-Physical Systems. In *Proceedings of SPLC '17, Sevilla, Spain, September 25-29, 2017,* 5 pages.
https://doi.org/10.1145/3106195.3106217

1 INTRODUCTION

Cyber-physical Systems (CPSs) describe autonomous and adaptable technical systems. They integrate computation into physical processes using sensors and actuators to monitor, control, and influence physical objects [5, 10, 12]. We consider a CPS as a set of multiple, reusable components that denote a hardware and/or software element to fulfill a purpose or accomplish a particular task, e.g. a production cell with multiple workstations. This does not restrict a component to operate in just one CPS but rather to be reusable in multiple CPSs. Thus, components have to adapt their behavior and interactions according to a superior goal (e.g. higher throughput) and the surrounding environment (e.g. temperature or human interactions). This results in feedback

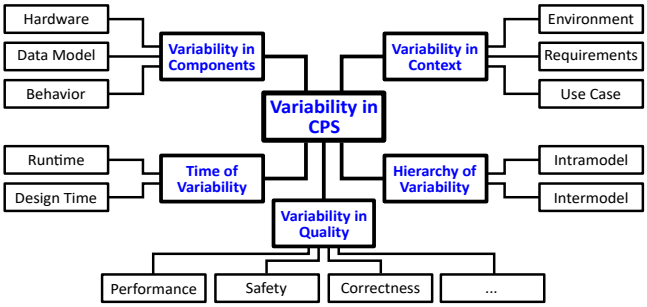

Figure 1: Classification of variability aspects in CPSs.

loops between the computational and physical processes. In addition, components of a CPS interact and communicate with each other. For these reasons, new design and development approaches arise, combining several disciplines, e.g., model-based development [10], self-adaptation [5], dynamic reconfiguration [12], and methods for CPS reliability [10, 12]. However, existing approaches share one common challenge in CPS development: variability. We think that one reason for many challenges in CPS engineering is the variability and heterogeneity in software and hardware as well as the variability in the environmental contexts, requirements, and application scenarios. To the best of our knowledge, this has not been addressed on a holistic scope.

In this paper, we aim to pave the way for further research by sketching a vision on a development approach for variable CPSs. Within this development approach, we have to connect different aspects of variability. For this purpose we propose a classification of variability aspects, namely *variability in components*, *variability in context*, *hierarchy of variability*, *quality in variability*, and *time of variability*. Furthermore, we derive current challenges and corresponding research questions based on this classification.

2 ASPECTS OF VARIABILITY

In contrast to classical systems, variability of CPSs cannot simply be defined on software or hardware level. Due to the interdisciplinary nature of CPSs, they span additional variability aspects that we sum up in the classification scheme in Figure 1. This classification is a result of discussions among domain experts, a review of existing literature, as well as a comparison with challenges and properties in related communities, such as industry 4.0, service-oriented architectures, multi-agent systems, and adaptive control. In Figure 1, we show the five aspects that describe the variability of a CPS.

Every component in a CPS should be variable. We classify this aspect as *variability in components*, which includes the variable hardware, data models, and behavior of a component. Hardware variability describes all changes that can be applied to a component's hardware (e.g., its sensors and actuators). Variability in data models is used to redefine how a component interprets and handles information, for example, changing an aggregation strategy. Behavioral variability represents changes in algorithms and tasks, such as, enabling object detection in a camera component.

As described before, a CPS operates in a specific context that is scoped by its use case, its specified requirements, and its surrounding physical environment. We classify the variability of these points within *variability in context*. Each CPS and its components execute a specific use case, which can be redefined dynamically, e.g., autonomous production versus collaborative construction with humans. In a particular use case, the functional requirements may change as well, e.g., a component adapts its path of movement to new circumstances. The surrounding environment and changes therein are crucial to a CPS. For example, a CPS must deal with varying brightness levels that influence a camera component.

Hierarchy of variability describes the structure of interacting, variable components of CPSs. Here, variability models (i.e. intramodels) describe the variability of a single component and its parts (e.g., a sensor or actuator). In contrast, interactions and dependencies between these components are part of an integrated variability model (i.e. intermodel).

A CPS has to fulfill several quality requirements, such as performance, safety, correctness, and others, which are adaptable to flexible circumstances. We classify these requirements within *variability in quality*. Performance and availability requirements of a CPS may vary depending on different customer needs, e.g., customizable throughput in a production cell. Safety requirements, which ensure the absence of harms on humans and the environment, are variable to the particular circumstances, for instance, in a fully autonomous production cell humans are not endangered, while they are in a co-working space. Depending on a CPS's use case and functional requirements, the set of applied tests and correctness analyses must be adapted to ensure correct behavior. Other quality requirements, as security, might be affected as well but are not considered within this paper.

A key feature of a CPS is its property to be adapted at design and runtime, which we classify within *time of variability*. On the one hand, a CPS's variability can be configured before startup (i.e. at design time). On the other hand, it must also be capable to be adapted or adapt itself during runtime, for instance, a productive CPSs often may not be interrupted in their processes to change its behavior.

3 CHALLENGES

In this section, we state four main challenges for future research in the development of variable CPSs: modeling heterogeneous variability, handling variability interactions, enabling self-configuration during runtime, and ensuring quality.

CH-1 Modeling: *A CPS's variability typically includes heterogeneous aspects, which have to be integrated into a common model.*

A CPS is heterogeneous by its nature, as it consists of various software and hardware components. In addition, we describe five aspects of a CPS's variability (e.g., for environment, use cases, and qualities), which introduces a new kind

of heterogeneity to the variability modeling of CPS's. This heterogeneity originates from three reasons: First, models must describe a wide range of variable parts such as source-code fragments, hardware components, or documentation. Second, models for different variability aspect of a CPS differ in the represented information. For example, a use-case diagram will rarely contain information about thermal behavior in the environment, while an environmental model may do. Third, models for different variability aspect also differ in their elements, structure, and development processes in order to suit their represented information. While the first reason is considered in classical software product line engineering (SPLE), the latter two arise from the variability aspects within a CPS.

Considering such heterogeneous models, we still lack in combining them. This hampers developers to achieve variable CPSs, as elements from one model cannot simply be represented in another. To overcome this, one may either develop a mapping between heterogeneous models or a uniform model for all aspects of variability. Either way, the complexity of the resulting variability models will increase significantly compared to current SPLs. Hence, challenges arise not only for modeling but also for scalability and analysis.

CH-2: Interaction *A CPS contains components that require variability-aware interaction channels.*

Variability mechanisms for a CPS have to consider all relevant aspects of variability. Therefore, components in CPS must be able to communicate and interact through two channels. Firstly, two components can identify themselves before communicating, as it is done with TCP/IP, Profibus, or Ethercat, to which we refer as *direct* interaction. In contrast, *indirect* interaction does not require identification, but rather communication via physical changes in the environment, for instance, based on sensors and actuators. As both interaction channels depend on the CPS's variability, they have to be designed variability-aware. This poses new challenges in communicating changes to adapt interactions and behavior, for instance, if a new component is added to the environment.

CH-3: Configuration *A CPS must be self-adapting and (re)-configurable at runtime.*

A CPS contains different components that are configurable. However, depending on the heterogeneous variability, these components must be able to configure themselves to adapt to changes, for instance, in their environment. This has to be achieved during runtime to not interrupt the behavior of the whole CPS. For software alone, dynamic SPLs [2, 3] seem promising. However, the heterogeneous variability poses new challenges considering the inclusion of hardware, environment, and interactions between them. For example, components must be notified about changes and have to propagate these into their different aspects of variability. In addition, some configurations might be more suitable in particular use cases and environments than others. Thus, sophisticated methods to find the best configuration of the whole CPS under their current circumstances are needed.

CH-4: Quality *A CPS acts within an environment and must ensure safety and correctness at all times.*

A CPS aims to achieve a defined goal based on qualitative properties. Ensuring quality in an SPL was rarely done in the past [13]. As heterogeneous aspects of variability and components interact, new methods to estimate and assure quality are required. This poses challenges on modeling variable qualities, measuring them, and guaranteeing that they are fulfilled during runtime. For example, safety properties may differ depending on the specific use case and environment of a component. In particular, handling unforeseen changes that conflict quality properties require further investigations.

4 TOWARDS CPS DEVELOPMENT

In this section, we discuss detailed research questions for the derived challenges. We remark, that these questions are strongly connected and depend on each other. For us, the application of approaches from SPLE, enhanced with methods from multi-agent systems and adaptive control, is promising.

4.1 Modeling Variability

SPLE is an established approach to design a family of customized software products, especially for embedded systems. We think that existing variability modeling approaches [4] (e.g., feature models and decision models) are promising to model variability in a CPS but, at the moment, lack in several aspects, as the following research questions point out.

How can we model environmental variability of a CPS? As stated before, variability in the environment plays an important role for a CPS. In contrast to software and hardware, a domain analysis, which is typically the first step in variability modeling, can hardly be done completely for the environment, due to its unmanageable complexity. Therefore, we find that modeling environmental variability is one of the hardest parts in describing a variable CPS. The scope of the environment has to be focused on those parts that are relevant for a particular CPS: Each component needs to only consider environmental changes that may influence its behavior. However, the environmental model must also be universal enough, so that components, which may be designed by different developers, can cooperate. Thus, it might be necessary to combine small internal environmental models of single components to a global model for the CPS. Ideally, it should also allow components to be reused in a different CPS. One promising approach might be the introduction of a generic, overarching environmental model, which describes universal concepts (e.g. mass, position, orientation, or time) and from which individual variability models can be derived.

How can we deal with heterogeneous variability models? A CPS consist of many variable parts, such as hardware, software, or the environment. For each, there exist numerous approaches to model variability that have their own assets and drawbacks. Thus, a modeling approach that is well-suited for modeling variability of one particular part might not be optimal for another. Moreover, two variability models can differ in their granularity, semantic meaning, and understanding of used concepts and terms. Therefore, the question arises how we can manage a set of heterogeneous variability

models, each describing one specific part of a CPS. Between two or more variability models, it must be possible to model dependencies, map variable parts that are equal, and adapt them to (re-)configurations.

How can we ensure scalability of the models and their corresponding analyses? Due to the complexity of a CPS's variability, the size of its variability models can be significantly larger than models originating from classical SPLE. Variability models are the basis for analyses of a system, such as detecting anomalies in the source code, testing the system's behavior, and finding configurations that fit certain properties. Since most of theses analyses have an exponential time complexity regarding the size of the variability model, the scalability of these analyses can become a problem for both, the developer and the system. To cope with these scalability issues, new analyses must be developed or existing analyses have to be adapted. Thus, we strive for efficient algorithms as well as adapted analysis strategies.

4.2 Managing Interactions

A CPS is usually a set of distributed components that cooperate and communicate to achieve common or antagonistic goals. For example, a production cell may produce individually configured cars. In a variable CPS these interactions between components can vary depending on the current configuration. Such variable interaction must be taken into account at design time and handled properly during runtime, wherefore the following questions arise.

How can we consider interactions in a CPS? So far, interactions in a CPS are modeled and implemented during design time. This process is insufficient, as system developers can hardly imagine every possible interaction that may appear during runtime. Hence, we propose to define families of interactions, which can be understood as an SPL of interaction models. This interaction model family describes a potential set of changes, e.g., which data can be communicated, that can be used to (self-)adapt a CPS's components to environmental changes. Thereby, system designers are released from the task of finding all possible interactions, but rather to determine parts in a CPS's interaction that could change. However, it is still uncertain, how to infer specific interaction models as well as implementing them on components.

How can we extend SPLE to handle direct and indirect interactions? Typically, components in a CPS interact in two different manners, *direct* (e.g., communication via identification in a network) and *indirect* (e.g., changing the environment, which is recognized by other components). Especially indirect interactions are challenging due to side effects, e.g., components undeliberately start an interaction although they did not intend to. Moreover, controlling changes in the physical environment is accompanied with huge additional effort or might even be impossible. Nevertheless, we think the idea of interaction model families is a suitable extension for SPLE to model and control both kinds of interactions.

How do we propagate changes to interacting systems? As components in a CPSs interact with each other, changing a component may require adaptations in others. For instance, components could exchange data differently, if a sensor is added or their physical distance changes. Hence, adaptations must be propagated during runtime in a suitable period. In particular, this poses the questions how to identify affected components and limit cascading changes.

4.3 Configuring Variants

One main goal of a variable CPS is to self-adapt according to environmental changes (e.g., increased input in a production cell or broken components) by automated reconfiguration at runtime. This includes all variability aspects of a CPS, which is why we also have to consider their interdependencies. Thus, the following research questions arise.

How can we enable self-adaptive reconfiguration at runtime? One of the most important aspects of a CPS is its ability to adapt to its context. This means that the components of a CPS react to changes and are capable to self-adapt. For example, an autonomous transport unit can reconfigure itself to find its path with cameras, if its laser scanner fails and can still react to unexpected obstacles in its environment. Hence, it replaces localization data from a local laser sensor with those from external image-based sensors. This has to be possible during runtime, for instance, to ensure safety and avoid interruptions. Consequently, it is necessary to develop mechanisms that allow rating and selecting variants autonomously, based on information of the current context and the provided variability models.

How can we deal with incomplete variability models during runtime? Most existing approaches for SPLs require that all (runtime) variability is known at design time [3]. For a CPS, this is rarely possible and, thus, a significant limitation. A CPSs consists of a number of heterogeneous components, potentially built by different vendors and with a different scope of use cases, e.g., a production cell for different products consisting of different workstations. Hence, universally modeling the domain will often not be possible. For this reason, we need to enable a CPS to (re-)combine and extend heterogeneous variability models. In addition, the component's context is crucial for its configuration. Building a static a priori variability model of all potential context instances is not possible with existing methods, and may never be. An approach to manage this issue, might be to utilize the concept of ontologies, restricting the model to aspects relevant to the CPS (e.g., a production solely constructing different cars) and introduce variability in its structures.

How can we integrate new functionality into a running CPS? For many applications of a CPS, dynamic extensions of functionality is a requirement. Technically, we can imagine two ways to integrate new functionality. First, implementing a new feature for a component and second, adding a new (type of) component to the CPS. However, both ways require updates to the variability models and interacting components. For the first way, we can adopt the aforementioned approach to reconfigure a CPS at runtime. To address the second way, we have to significantly update existing components that are

affected by the new one. For example, a new autonomous transport unit may affect the movements of other components. While we can build on existing approaches as for the first way, we have to consider that direct interactions may not be immediately available but must be established. In contrast, indirect interactions can immediately affect the CPS, wherefore adapted design approaches seem necessary.

How can we design interfaces for functions and data to enable a CPS to reconfigure at runtime? To enable a CPS to automatically detect, configure, and run new or changed components, interfaces require a specific and variable design. Conceptually, we propose a three step process, which describes how interfaces in a CPS could be handled:

(1) Request a component's possible interfaces.
(2) Create and configure interfaces with existing capabilities or by transferring new code to the component.
(3) Notify a CPS's components about the updated component and its interfaces.

The first step requires that components have a set of active and non-active interfaces available. Within the second step, it should be possible to extend the component's capabilities by transferring external code, e.g., extending a camera with an object detection algorithm. For that purpose, this mechanism has to evaluate this external code on static and dynamic criteria. Here, methods from symbolic code execution and code verification [7, 11] could support the SPLs. The last step requires runtime mechanisms for updating other components, which could be handled via the same update mechanism described in Section 4.1. Moreover, in case of a failing update, backup mechanisms are needed, e.g., self-x mechanism from the organic computing domain [9].

4.4 Assuring Quality

A CPS manipulates its physical environment and often operates autonomously for a long time. Thus, a CPS needs to reliably fulfill its required quality properties, which is why the following questions arise.

How can we model quality metrics within variability models? In addition to a CPS's behavior, environment, and interaction models, requirements and dependability metrics have to be integrated. For instance, components should be able to report their dependable values, such as variations in sensor values. However, quality is traditionally assessed on system level, which requires an inference concept to derive quality metrics from multiple variability models. Moreover, if applied at runtime, the active variants of all components and the current environment must be taken into account.

How can we provide dependability guarantees for quality metrics? Mechanisms to validate a CPS's quality are scarce. An ideal way to cope with this problem would be to apply the runtime verification paradigm [8] to enrich a system with a safety quality mechanism. During runtime, this mechanism continuously monitors the system and guides it back into a safe state when a problem is detected. For that purpose, expertise from other domains can be applied, such as, multi-criteria decisions [6] or model-checking approaches [1]. However, these approaches may cause other issues, such as scalability of the variability model or uncertainty of finding a safe state. Thus, other mechanisms might be more feasible.

5 CONCLUSION

CPSs take over more responsibilities in daily life. Innovations emerge through new systems controlling and coordinating their components to physically interact with the real world. These systems are often based on a CPS continuously monitoring and analyzing it's environment, and (re-)configuring itself accordingly. As a result, the system's behavior and variability reach far beyond software.

In this paper, we sketched and categorized relevant aspects of variability in a CPS. Additionally to software variability, a CPS also includes, for instance, variable environments, dynamically changing hardware, as well as variable requirements. Based on these observations, we identified four areas of future research to address this heterogeneity: modeling, interaction, configuration, and quality. We then list research question for each of these challenges and outline ideas on addressing these with SPL and related domains. In future work, we aim to investigate the research questions and join communities to address the inter-disciplinary scope of variable CPSs.

ACKNOWLEDGMENTS

This research is supported by DFG grants LE 3382/2-1, SA 465/49-1, and Volkswagen Financial Services AG.

REFERENCES

[1] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. 2008. *Principles of Model Checking*. MIT Press.
[2] Jan Bosch and Rafael Capilla. 2012. From Static to Dynamic and Extensible Variability in Software-Intensive Embedded System Families. *Comput.* 45, 10 (2012), 28–35.
[3] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Cortés, and Mike Hinchey. 2014. An Overview of Dynamic Software Product Line Architectures and Techniques: Observations from Research and Industry. *J. Syst. Software* 91 (2014), 3–23.
[4] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *VaMoS*. ACM, 173–182.
[5] Ilias Gerostathopoulos, Tomas Bures, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, Frantisek Plasil, and Noël Plouzeau. 2016. Self-Adaptation in Software-Intensive Cyber–Physical Systems: From System Goals to Architecture Configurations. *J. Syst. Software* 122 (2016), 378–397.
[6] Alessio Ishizaka and Philippe Nemery. 2013. *Multi-Criteria Decision Analysis: Methods and Software*. John Wiley & Sons.
[7] Sarfraz Khurshid, Corina S Păsăreanu, and Willem Visser. 2003. Generalized Symbolic Execution for Model Checking and Testing. In *TACAS*. Springer.
[8] Oded Maler. 2016. Some Thoughts on Runtime Verification. In *RV*. Springer, 3–14.
[9] Christian Müller-Schloer. 2004. Organic Computing: On the Feasibility of Controlled Emergence. In *CODES*. ACM, 2–5.
[10] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-Physical Systems: The Next Computing Revolution. In *DAC*. ACM, 731–736.
[11] David A. Ramos and Dawson Engler. 2015. Under-Constrained Symbolic Execution: Correctness Checking for Real Code. In *USENIX Security*. USENIX, 49–64.
[12] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. 2011. A Survey of Cyber-Physical Systems. In *WCSP*. IEEE, 1–6.
[13] Thomas Thüm. 2015. *Product-Line Specification and Verification with Feature-Oriented Contracts*. Ph.D. Dissertation. University of Magdeburg.