Managing Cyber-Physical Production Systems Variability using V4rdiac: Industrial Experiences

Hafiyyan Sayyid Fadhlillah
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Linz, Austria
hafiyyan.fadhlillah@jku.at

Antonio M. Gutiérrez Fernández
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Linz, Austria
antonio.gutierrez@jku.at

Rick Rabiser
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Linz, Austria
rick.rabiser@jku.at

Alois Zoitl
CDL VaSiCS, LIT CPS Lab
Johannes Kepler University Linz
Linz, Austria
alois.zoitl@jku.at

ABSTRACT

Cyber-Physical Production Systems (CPPSs) are highly robust and versatile production systems that utilize diverse hardware components through control software. Employing a systematic variability management approach for developing variants of control software can reduce cost and time-to-market to build such complex systems. However, employing this approach in the CPPS domain is challenging. Engineering CPPSs require multidisciplinary engineering knowledge (e.g., process, signal, mechanical). Knowledge about CPPS variability is thus typically scattered across diverse engineering artifacts. Also, variability knowledge is usually not documented explicitly but rather tacit knowledge of mostly senior engineers. Furthermore, control software is commonly implemented using a graphical Domain-Specific Modeling Language (DSML) which only provides minimal support to express variability. This paper describes our experiences dealing with these challenges in an industrial context using a multidisciplinary variability management approach called Variability for 4diac (V4rdiac). V4rdiac is an integrated approach that allows CPPS engineers to conduct stepwise product configuration based on heterogeneous variability models from multiple engineering disciplines. V4rdiac also provides a mechanism to automatically generate control software based on a set of selected configuration options. We evaluate how V4rdiac implements and manages CPPS control software variants in the metallurgical production plant domain. We describe the benefits and lessons learned from using V4rdiac in this domain based on feedback from industrial practitioners.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

Variability Modeling, Software Product Line, Cyber-Physical Production System, Software Configuration

SPLC '23, August 28-September 1, 2023, Tokyo, Japan

ACM ISBN 979-8-4007-0091-0/23/08…$15.00
https://doi.org/10.1145/3579027.3608994

ACM Reference Format:
Hafiyyan Sayyid Fadhlillah, Antonio M. Gutiérrez Fernández, Rick Rabiser, and Alois Zoitl. 2023. Managing Cyber-Physical Production Systems Variability using V4rdiac: Industrial Experiences. In *27th ACM International Systems and Software Product Line Conference - Volume A (SPLC '23), August 28-September 1, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3579027.3608994

1 INTRODUCTION

The market is currently experiencing unprecedented fluctuations and immense pressure for customization, resulting in high demand for the development of versatile and long-running Cyber-Physical Production Systems (CPPSs) [3, 39]. CPPSs are a type of Cyber-Physical System (CPS) which employ diverse hardware components through control software to deliver production functionality [16, 26, 35]. Developing a CPPS involves a team comprising multiple engineering disciplines (e.g., process, mechanical, and electrical) [5]. During the design phase, engineers from each discipline produce diverse domain-specific engineering artifacts (e.g., Computer-Aided Design (CAD) drawings, P&I diagrams, and Detailed Technical Specification (DTS)) with diverse semantics describing domain-specific knowledge to build such complex systems. Specifically, control software is typically developed using a graphical Domain-Specific Modeling Language (DSML) defined by industry standards such as IEC 61499 [18] or IEC 61131 [36].

The challenges posed by a globalized market have driven industries, including CPPS industries, to reduce the time and effort required to develop system variants [3, 31]. To achieve this, industries must address challenges in their engineering processes. Building a system with artifacts from multiple engineering disciplines raises the learning curve. Variability knowledge is typically scattered across domain-specific engineering artifacts from multiple disciplines and mostly not explicitly documented but tacit knowledge by senior engineers. Although there are mechanisms for configuring certain subsystems, high engineering efforts are still required. In particular, DSMLs used for building control software do not provide a formal mechanism to express variability.

To address these challenges, we proposed a multidisciplinary variability management approach for CPPS called Variability for 4diac (V4rdiac) [6, 7]. The approach is inspired by Software Product Line (SPL) engineering [1, 27] and developed in a long-term industry-academia collaboration project. The goal of this project is to reduce the implementation and maintenance effort required for developing control software variants. We built V4rdiac by utilizing

an open-source framework for developing industrial distributed control software called Eclipse 4diac™ [41][1] as well as the feature-oriented software development framework FeatureIDE [2]. V4rdiac encourages each engineering discipline of a CPPS to explicitly document variability using a variability model of their choice. The overall CPPS variability is represented by linking these models using cross-discipline constraints. Furthermore, V4rdiac provides a delta modeling approach for IEC 61499 DSML to express control software variants. V4rdiac also provides tooling that enacts configuration options from the variability models to support stepwise product configuration. CPPS engineers can automatically generate (parts of) the control software based on a set of selected configuration options and the delta models.

In this paper, we describe our experiences of using V4rdiac for managing industrial CPPS control software variants. Specifically, we describe a scenario from one CPPS that motivates using the V4rdiac approach. We also present the V4rdiac tool architecture and the process activities to use it in CPPS engineering process. Then, we describe how V4rdiac provides benefits that can address different challenges of managing CPPS variability. Additionally, we describe the lessons learned and further challenges captured from industrial practitioners using V4rdiac in this domain.

Due to non-disclosure agreements, we cannot reveal details about our industry partner. However, we show real examples from their CPPS throughout this paper and discuss engineers' feedback. Our aim is also to make our work better generalizable and applicable to other CPPSs. Thus, we selected examples that we expect to also be relevant in other domains. For example, control software in many CPPSs will have to deal with the variability of valves, pumps, and tanks (cf. Section 2).

The remainder of this paper is structured as follows. In Section 2, we motivate our work with a concrete CPPS scenario. We then describe the V4rdiac approach in Section 3. In Section 4, we describe benefits and lessons learned based on experiences of using V4rdiac for a concrete CPPS in the industrial context. We discuss related work in Section 5 and conclude our paper in Section 6.

2 CHALLENGES OF MANAGING CPPS VARIABILITY IN INDUSTRY

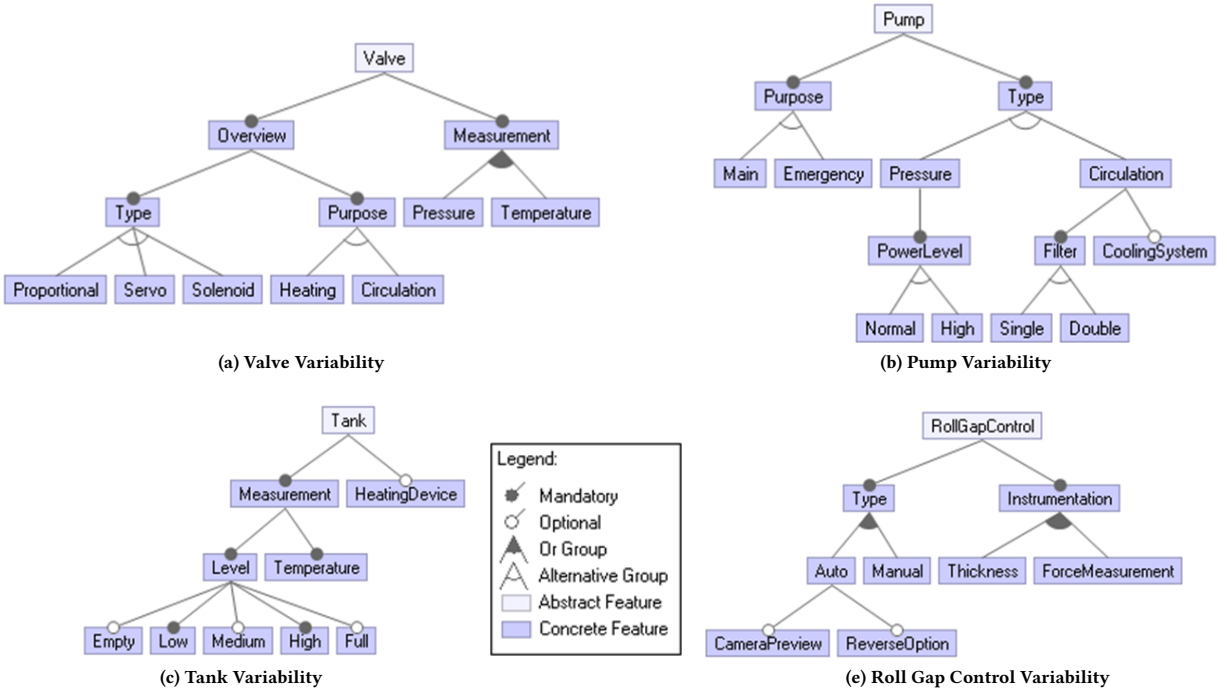
We use the example of a *metallurgical production plant* to illustrate the challenges of managing CPPS variability and our approach to address these challenges. A metallurgical production plant aims to produce a steel product (e.g., steel coils or slabs) with a desired thickness and/or one or more mechanical properties. Such a plant comprises various highly complex machinery such as casting machines, hot rolling mills, and cold rolling mills. These machines typically have different hardware components, such as *Valves*, *Pumps*, *Fluid Tanks*, or *Roll Gap Controls*. To build such a system, the customer and the business department first discuss the requirements of such a plant and later create a Bill of Material (BOM) written in textual documents (e.g., spreadsheets and/or word documents). Based on these requirements, engineers from multiple disciplines (e.g., signal, metallurgical, electrical, and process) define the overall production strategy. These engineers typically use artifacts such as CAD drawings, P&I diagrams, and/or DTS to describe their engineering knowledge. Control software engineers use such artifacts as input to develop the control software of a plant.

The components' properties in each machine vary depending on customer and technical requirements (cf. Fig. 1). For example, the engineers' decision to use a `Proportional`, `Servo`, or `Solenoid` valve influences energy efficiency and the overall project cost of the metallurgical production plant. Engineers can also decide which `Measurement` type is used in the valve component. Also, the engineers can decide which `Type` of pump is used for each machine in the plant. A `Circulation` pump is used to circulate residual fluid after being used in the machine while a `Pressure` pump is used to circulate fluid from the tank component. Additionally, engineers can decide whether a pump is used as a `Main` or backup (`Emergency`) pump. Engineers require different granularities of `Level` measurement inside tank components depending on the fluid's importance in the metallurgical process. Engineers must also decide whether to keep certain liquids warm or hot inside the tank component by using `HeatingDevices`. To ensure the steel product matches certain metallurgical characteristics, the engineers must decide the type of `Instrumentation` used in roll gap control component. A roll gap control can also be designed to be fully automatic (`Auto`) and/or have an interface to control it (`Manual`). Furthermore, there can be multiple instances of components in each machine. A casting machine can have two pressure pumps and one circulation pump while a hot-rolling mill machine might have two circulation pumps. Decisions about the number of components and their properties later influence which sensors, actuators, electrical components, and signals can be used in the machines.

In our research, we are currently focusing on providing variability support for control software implemented using a DSML defined in IEC 61499 [18]. IEC 61499-based control software is defined by creating a network of interconnected Function Blocks (FBs) of certain types. FB types and instances could be compared to classes and objects in an object-oriented programming language such as Java. Each FB type encapsulates an algorithm and provides an event and data communication interface. Types can be stored in a type library, thus creating a collection of reusable software components. The standard also defines additional concepts such as *subapplication* and *adapter* to improve the modularity and reusability of the FB types and its instances [40].

The development of control software with IEC 61499 can be challenging. First, there is no explicit mechanism to represent system variability. Decisions regarding variability in each software development stage (requirements, design, and implementation) are mostly based on implicit knowledge of the engineers and information from documents and spreadsheets. This leads to a high dependence on experts (with several years of experience) in project development. On the other hand, the lack of explicit models also prevents the formalization of dependencies between disciplines that would allow change impact analysis (e.g., assessing the effects of changing a valve model on the software). Employing change impact analysis is especially important after the deployment phase and in commissioning because machine downtime costs are high. Finally, the size and complexity of control software are also challenging. Although certain aspects of software generation have been automated, they still depend on manual actions such as integrating new components

[1] https://www.eclipse.org/4diac/
[2] https://featureide.github.io/



(a) Valve Variability **(b) Pump Variability**

(c) Tank Variability **(e) Roll Gap Control Variability**

Figure 1: (Partial) Variability models of typical components of a CPPS, illustrated using FeatureIDE feature model [23].

or manually adapting certain parameters. Industrial-scale control software can have ten up to a hundred thousand interconnected FB instances. Following a clone-and-own approach for creating variants for customers thus quickly becomes infeasible. Thus, the industry comes up with custom-developed approaches to manage variability, e.g., by defining configuration parameters allowing to remove or deactivate parts of the software. Nevertheless, development and maintenance remain challenging and thus further motivate a more systematic approach to variability management in CPPS.

3 V4RDIAC

We use our existing multidisciplinary variability management approach V4rdiac [6, 7] to address the described challenges.

3.1 Process Activities

Using V4rdiac, engineers can use any type of variability model to describe the variability in their domain. Engineers can later define cross-discipline constraints using propositional logic for specifying include/exclude relations between variation points from multiple variability models. V4rdiac uses delta modeling [33] as a mechanism to define variability in the control software. The delta models describe changes from an existing application model (e.g., adding or removing FBs) according to one or more variation points. V4rdiac also supports a stepwise product configuration process, i.e., the engineer can define one or more configuration steps where each step shows the configuration options defined in one variability model. After the product configuration is finished, V4rdiac generates the control software that reflects the selected configuration options.

Fig. 2 illustrates the process activities to perform domain engineering using V4rdiac and the stakeholders involved in each of them. The first activity is to *(1) define multidisciplinary CPPS variability* to formalize engineers' knowledge into one or more variability models. In this activity, each organizational unit or engineering discipline can use its engineering artifacts and technical documents as a source of variability knowledge within their domain. We allow them to define the variability in a variability model of their choice. For example, hierarchical variability in mechanical engineering might be described using a feature model while behavioral variability in process engineering might better be described using a decision model [24].

In the second activity, engineers *(2) define the stepwise configuration*, i.e., define a sequence of steps and which variability model is configured in each step. The purpose of the stepwise configuration is to provide a flexible configuration that adapts to the state of practice. For instance, the engineers may define a stepwise configuration to configure variability models from different engineering disciplines (process, electrical, etc.) according to their daily engineering procedure. We encourage only one variability model to be configured per step, although V4rdiac would allow configuring multiple variability models per step. Additionally, the users can create multiple stepwise configuration setups in V4rdiac to describe different ways to perform stepwise configuration.

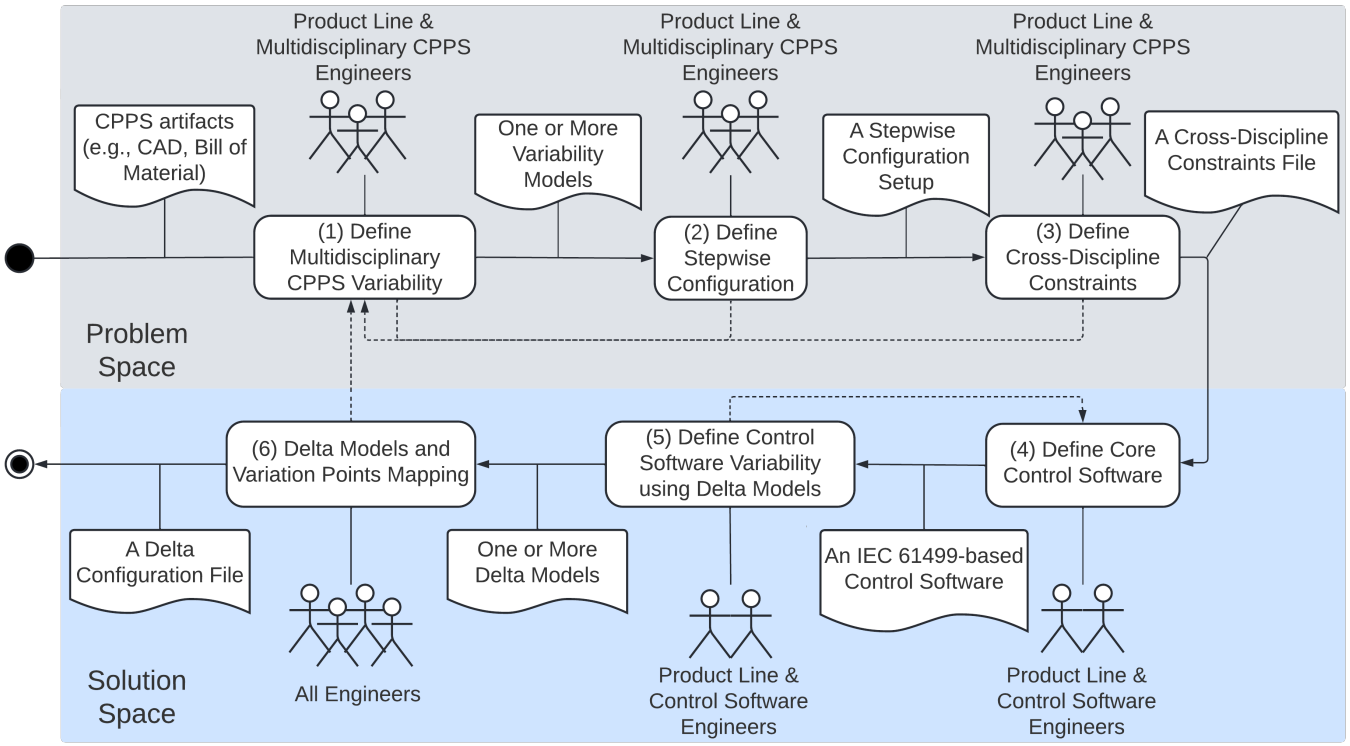


Figure 2: Domain Engineering process activities for expressing CPPS variability using V4rdiac, extended from [7].

In the third activity, engineers must *(3) define cross-discipline constraints* to formalize include/exclude relations between variation points from different organizational units and engineering disciplines. Some additional variability can arise in the second and third activities based on the discussions between all organization units and engineering disciplines. Engineers can still incorporate new variability into their variability models while defining the configuration steps and cross-discipline constraints.

In the fourth activity, the control software engineers *(4) define the core control software*, which shall be used as a basis. Then, they *(5) define control software variability using delta models* that reflect the changes to adjust this core control software according to the variation points from the variability models. In the last activity, engineers *(6) define a mapping between the variation points and the delta models* (delta configuration).

Based on these artifacts, engineers can perform the application engineering using the V4rdiac product configuration tool. Users first need to select the stepwise configuration setup that will be used when performing product configuration (cf. Fig. 6). After the product configuration process is finished, the tool can automatically generate a control software variant based on the user's selection of configuration options defining which changes (specified in delta model) to execute.

3.2 V4rdiac Architecture

V4rdiac has diverse components (cf. Fig 3) to support the process activities in both domain and application engineering.

In **domain engineering**, V4rdiac mainly uses a *Standardized Models Provider*, a *Cross-Discipline Constraints Reader*, a *Multi-Level Configuration Provider*, and an *IEC 61499 Delta Modeling* component. V4rdiac uses the *Standardized Models Provider* to load all variability models for further usage. These variability models are first transformed into a single standardized representation by utilizing the variability model transformation framework TRAVART [11]. We currently use FeatureIDE feature models [23] as our standardized representation. Currently, the standardized representation is hidden from the user and used for creating cross-discipline constraints and enacting configuration options. Then, the *Cross-Discipline Constraints Reader* component loads all the cross-discipline constraints written by the users into V4rdiac. Later, these constraints are used by *Multi-Level Configuration Propagator* to validate the selected configuration options. The *Multi-Level Configuration Provider* component provides a wizard to create the stepwise configuration setup. Specifically, in this wizard, the user can define all the product configuration steps and assign variability models into these configuration steps. Furthermore, the *IEC 61499 Delta Modeling* component provides the infrastructure to support users when creating delta models and delta configuration files. Currently, we provide a *textual* language to create delta models in V4rdiac [8]. This component mainly utilizes several components from Eclipse 4diac. Specifically, we use Eclipse 4diac IDE and its default Type Library, which contains all the necessary functionalities to develop IEC 61499-based control software. Control software engineers are allowed to add additional FB types that are necessary when developing their CPPS control software into the Type Library.

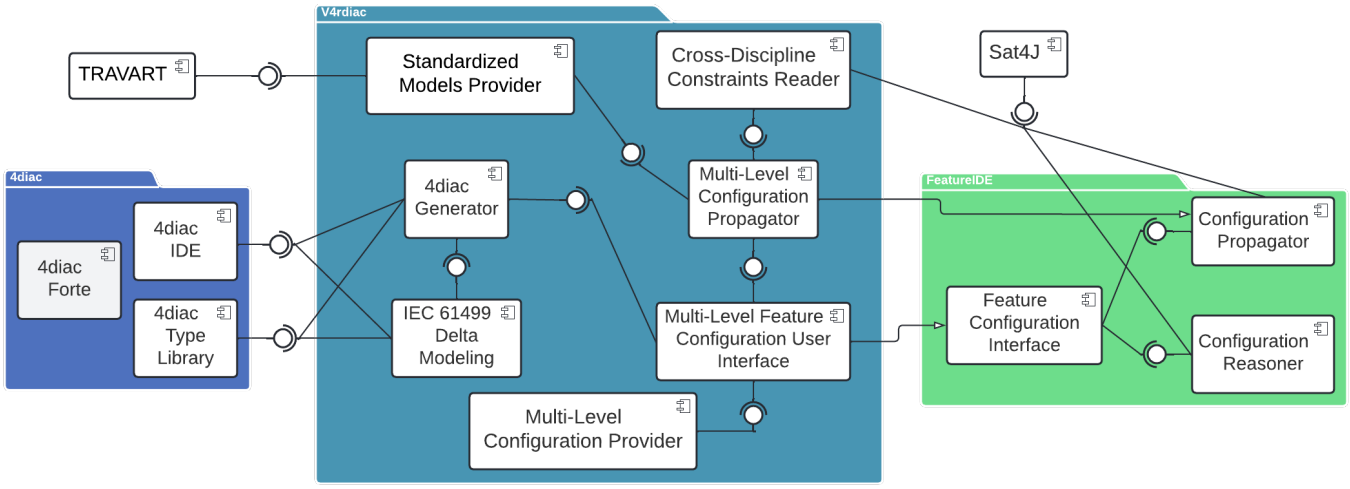


Figure 3: V4rdiac Architecture.

During **application engineering**, V4rdiac uses a *Multi-Level Feature Configuration User Interface*, a *Multi-Level Configuration Propagator*, and the *4diac Generator* component. The *Multi-Level Configuration Propagator* component works: (1) as a "*global validator*", i.e., it validates the selected configuration options according to all the variability models as well as the cross-discipline constraints loaded by the *Cross-Discipline Constraints Reader* and (2) as a *local validator*, i.e., it validates the selected configuration options according to the variability models configured in a particular step. This component also provides a functionality to automatically select one or more configuration options according to the local and global validator. As a result, users are guided to select feasible configuration options that are valid for both validators. To deliver these functionalities, the *Multi-Level Configuration Propagator* component extends the FeatureIDE *Configuration Propagator* component [23] and SAT4j [3] to deliver all its functionality. The *Multi-Level Feature Configuration User Interface* presents configuration options in a stepwise manner according to the stepwise configuration setup. We extended FeatureIDE's *Feature Configuration Interface* component [23] for this purpose. Furthermore, we also use the *Configuration Reasoner* embedded in the *Feature Configuration Interface* to provide a textual description why one or more configuration options are automatically selected. Users can only access a new step if the selected options are valid according to the local validator. Additionally, users can only finish the product configuration if the global validator confirms validity. After the product configuration is finished, V4rdiac invokes the *4diac Generator* component to generate the control software variant.

4 USING V4RDIAC TO MANAGE CPPS VARIABILITY IN INDUSTRY

Together with engineers from our industry partner, we conducted several workshops and proof of concept studies to manage control software variability in their industrial CPPS using V4rdiac [6, 7]. Based on this, we describe our experiences in terms of ***Benefits (B)*** and ***Lessons Learned (LL)***. Each benefit describe a positive aspect experienced by our industry partner when using V4rdiac in their engineering process. On the other hand, each lesson learned describes a necessary improvement for V4rdiac to further increase its usability in the CPPS engineering process. In each subsection, we explain the benefits and lessons learned related to domain engineering process activities (cf. Fig 2) and product configuration to derive control software variants.

4.1 Define Multidisciplinary CPPS Variability

We used two approaches to capture the overall metallurgical production plant variability. First, we conducted multiple workshops with engineers to gather the plant's variability. We created initial variability models based on these workshops and by analyzing engineering artifacts provided to us by the engineers. Also, we manually reviewed control software together with engineers. We allowed engineers to refine the initial variability models and discussed the feedback on the models in multiple workshops. We could successfully define variability models for several components in a metallurgical production plant (cf. Fig 1). As a further example, Fig. 4 depicts the motor control variability, which acts as an interface between hardware and control software. Additionally, a motor control can also include an optional temperature sensor to guarantee its operation depending on the temperature and an operator panel to configure operation in runtime.

***B1* - Reaffirm engineers' variability knowledge**. Current documents to describe the systems (CAD, spreadsheets, etc.) do not or only partially describe the systems' variants. Configuration decisions about the system are made based on expert knowledge and certain documents (e.g., specification documents). It is thus hard to assess the impacts of changes. Formalizing the variability using V4rdiac facilitates a direct understanding of the existing variants for each part of the system (e.g., a motor) and also considers the relationships between elements of the same part (e.g., a motor with a specific hardware and the possible controller devices for that hardware).

[3] https://www.sat4j.org/

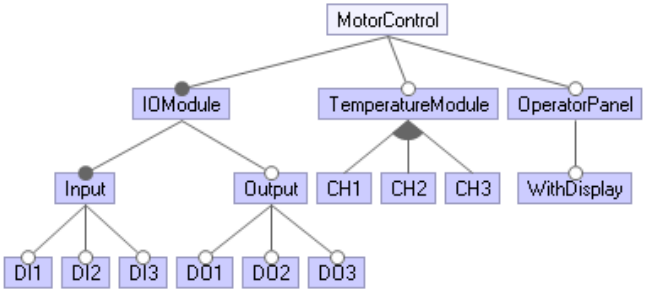


Figure 4: Motor control variability, illustrated using a FeatureIDE feature model [23].

Furthermore, creating variability models allows engineers to confirm their understanding of the variability in their engineering discipline. During one of the workshops, we created an initial feature model that represents the overall variability of a certain machine based on the BOM document. Following the BOM document, we define that a machine can only have 1-6 `Pressure` and 1-3 `Circulation` pumps. However, in practice, the engineers commented that only up to 4 `Pressure` and 2 `Circulation` pumps are possible. Thus, the BOM document could be refined in addition to now having a formal document describing the variability of the machine.

***LL1* - Variability modeling requires experience**. In the first workshops, we introduced feature modeling and showed engineers how to describe variability using a feature model. In addition, we showed the expressiveness of the feature model and the configuration support provided based on feature models. However, engineers instinctively treated the variability model as a software configurator for each software variant rather than capturing the overall systems' variability. For instance, they created one variability model for each control software variant to describe its parameters. To address such issues, we performed multiple workshops to refine the variability models created by the engineers. We conclude that variability modeling requires some experience and approaches must provide clear guidelines [22], e.g., on creating variability models.

4.2 Define Stepwise Configuration and Cross-Discipline Constraints

After creating the variability models, we interviewed several senior CPPS engineers about the plant's overall variability. We demonstrated how to create the stepwise product configuration setup using V4rdiac. In this demonstration, we use a configuration wizard provided by V4rdiac to define the configuration steps. For instance, the engineers used the configuration wizard to define the tank to be configured in the first step followed by pump, valve, roll gap control, Human-Machine Interface (HMI), and motor control as illustrated in Fig. 5. Additionally, we asked them about include/exclude relations between variation points from different variability models and documented these relations in a cross-discipline constraints file (cf. Listing 1). We simulated how configuration options can be shown step-by-step according to the stepwise product configuration setup and the cross-discipline constraint file. Then, we asked

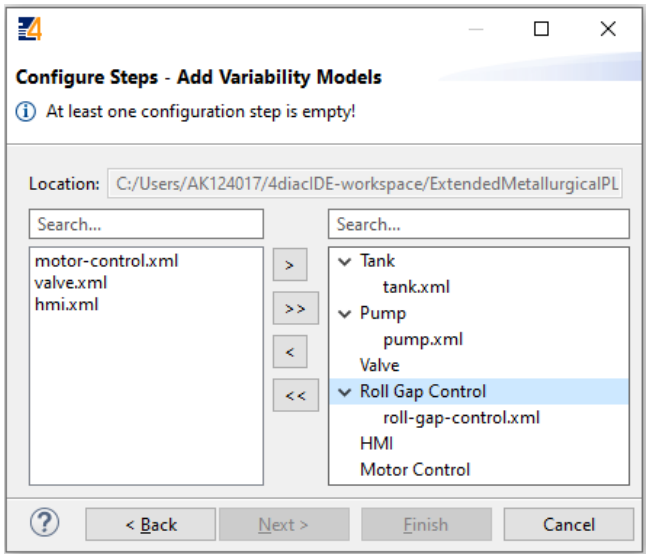


Figure 5: Defining Stepwise Configuration Steps in V4rdiac.

the engineers for feedback about adopting the stepwise product configuration as a common design practice.

***B2* - Detecting inconsistencies between components**. Decisions in one component can directly affect other components. For instance, it is necessary to equip a display monitor in a motor control (by selecting `WithDisplay` feature) if `Full` and `Medium` measurement levels are required for monitoring liquid in the tank during the machine operation (cf. `CD4` in Listing 1). Such knowledge is often communicated manually, e.g., through emails or meetings, making synchronizing changes across disciplines or tracing decisions difficult or at least harder to manage. Explicitly describing such relations (as cross-discipline constraints) in a centralized place allows all engineers to trace relations without or with less manual communication.

```
CD1) RollGapControl#Type#Auto => Valve#Overview#Type#
     Servo;
CD2) Valve#Measurement#Temperature || Tank#Measurement#
     Temperature => MotorControl#TemperatureModule;
CD3) Tank#HeatingDevice || Pump#Type#Circulation#
     CoolingSystem => MotorControl#OperatorPanel;
CD4) Tank#Measurement#Level#Medium && Tank#Measurement#
     Level#Full <=> MotorControl#OperatorPanel#
     WithDisplay;
```

Listing 1: Cross-discipline constraint example.

Formalizing relations between different components, potentially from different engineering disciplines in CPPSs, also facilitates detecting inconsistencies. For instance, a process engineer commented that the `Circulation` valve could only be paired with `Circulation` pump. This knowledge is derived based on the process engineer's experience from the ongoing development of several plants. However, senior engineers from electrical engineering described that the `Circulation` valve could also be paired with `Pressure` pump for some old plants. Based on this discussion, we could refine the cross-discipline constraints.

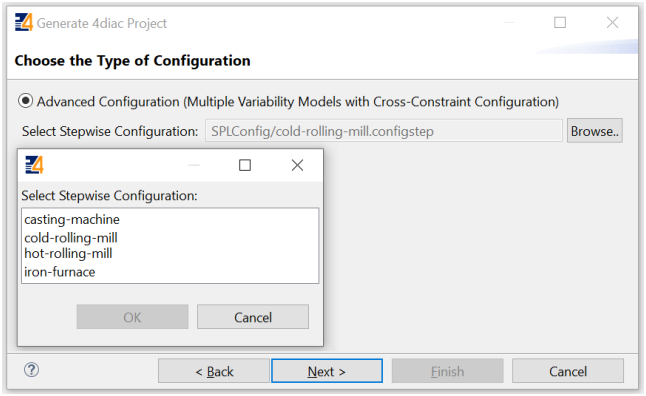


Figure 6: Choosing a Stepwise Configuration Setup in V4rdiac.

***LL2* - Incremental configuration processes are more realistic**. We learned that configuring a CPPS cannot be done in a strictly sequential manner. For instance, the configuration process could start by selecting/deselecting some configuration options in the Type subtree of pump, roll gap control, and valve components while the rest remains unsolved. A process engineer then might configure a subset of production processes related to these components. The configuration process would then go back to resolve the remaining pump, valve, roll gap control, and tank configuration options before proceeding back to the process engineer or moving on to the next configurations step.

In V4rdiac, the users can create multiple stepwise configuration setups allowing them to select different variants (cf. Fig 6). However, V4rdiac only allows its users to configure the next configuration step if the local validator in the current step confirms validity. In most cases, the engineers need to resolve all the configuration options to complete the current step and proceed to the next configuration step. Thus, V4rdiac can only partially support such an incremental configuration process. Providing an additional mechanism to mark some constraints as soft [21] could help to refine the validation process and enable engineers to access all steps without satisfying all local constraints.

***LL3* - A rich semantic language is needed for defining complex constraints**. Constraints in industrial CPPSs can become quite complex. For example, in the motor control (cf. Fig. 4), the `IOModule` acts as a communication interface between the machine components and the control software. Each input and output module within the `IOModule` provides a limited number of input/output ports. Therefore, to ensure that we can connect all the ports required by the machine components, an adequate number of `IOModules` must be provided. Each component requires a different number of `Input` ports, for instance: (1) `Pressure` measurement of a valve requires 2 ports, (2) `Level` measurement of a tank needs 1-4 ports, depending on the selected granularity `Level`, and (3) the `Instrumentation` of a roll gap control requires 2 ports for each instrument. Additionally, each `Input` in the `MotorControl` also has a different number of ports. The `DI1` contains 4 ports while both `DI2` and `DI3` only have 2 ports. Furthermore, one `Input` can only be paired with one unique device from a component. For instance, we can pair `Pressure` measurement with 2 ports from `DI1` albeit leave the other 2 ports unused. As a result, selecting different variation points for each machine component influences the number of motor controls required to build a metallurgical production plant. Finding an optimal number of motor controls (cf. Listing 2) is necessary to optimize the cost of building such metallurgical production plants. Unfortunately, V4rdiac currently does not provide a mechanism to express and evaluate such expressions. Providing a language such as pvSCL [4, 29] instead or in addition to the simple propositional logic constraints that V4rdiac currently supports could be a starting point to address this issue.

```
LET allLevelFeat = getSelectedFeature(Level)
LET granularLevels = allLevelFeat.filter(this.childrens.
    eval(isOptional()).count() > 1).count()
LET pressureFeat = getSelectedFeature(Pressure).filter(
    this.root == Valve)
LET forceMeasurement = getSelectedFeature(
    ForceMeasurement)
LET thickness = getSelectedFeature(Thickness)
LET motorControl = granularLevels
LET 2PortsReq = (allLevelFeat.count() - granularLevels) +
     pressureFeat.count() + forceMeasurement.count() +
    thickness.count()
2PortsReq = 2PortsReq - granularLevels * 2
motorControl = motorControl + 2PortsReq / 3
RETURN motorControl
```

Listing 2: Complex constraint example. Calculating the optimal number of MotorControls given a set of selected machine components.

4.3 Define Core Control Software and Delta Models

In this activity, we did a series of workshops with a senior CPPS engineer and a control software engineer. Both engineers have knowledge of the relationship between variation points and a subset of interconnected FBs. In the first interview, we asked these engineers to provide a core control software of the valve component (cf. Fig. 1). We discovered that the core control software of the valve uses an architectural pattern, i.e., a 150% model [17] where all variants are implemented (cf. Fig. 7). Consequently, our delta models then have to remove unnecessary FB instances or connections based on unselected variation points (cf. Listing 3, which is applied when `Proportional` is unselected when configuring a valve component). On the other hand, the core control software for roll gap control is a minimum variant containing a minimum set of features. In this case, the delta models mainly add and modify the necessary FB instances or connections to reflect the selected variation points. We created the delta models after the first workshop. Then, we used follow-up workshops to validate and refine our delta models. After the delta models were completed, we asked the engineers for their opinion on the delta models. The engineers stated some benefits of using delta models to express control software variability. Additionally, they are interested in experimenting more with delta modeling since it enables them to automatically generate (part of) the plant's control software (cf. Subsection 4.5). However, the engineers also expressed concerns that must be addressed before using the delta modeling approach in their engineering process.

***B3* - Delta models allow to discuss software architecture**. Engineers discovered that they could use delta models as a basis

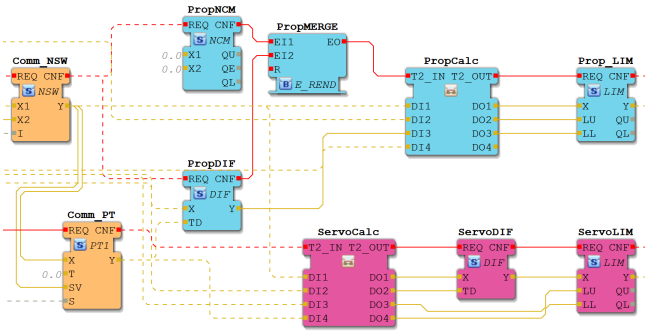


Figure 7: Core control software of a valve component. Colors assigned to each FB indicate its purpose (Orange is mandatory, Light Blue is unique for `Proportional`, and Purple is unique for `Servo`).

to discuss control software architecture. The number of changes stored in a delta model increases if there are unnecessary FB instances that are duplicated/cloned. These clones typically exist for variants of a core control software representing a 150% model. To introduce a variant of certain functionality, the engineers first copy a subset of FB instances related to the original functionality. Then, the engineers adjust the copied FB instances by connecting them to new FB instances or removing existing FB instances. As a result, some FB instances are completely the same yet exist multiple times in the core control software. For instance, Fig. 7 shows that FBs with type `DIF` and `LIM` are used in both the `Proportional` and `Servo` valve. Each variant instantiates these FB types separately and they are also wired differently. Thus, delta models for both `Proportional` and `Servo` valves will contain the same operations to remove FBs with type `DIF` and `LIM`. Engineers could investigate how to use the same instance of FB type `DIF` and `LIM` for both `Proportional` and `Servo`. Additionally, the number of delta models also increases if the changes related to a variation point are scattered across different hierarchies. IEC 61499 provides a mechanism to decompose an application model into multiple hierarchies. In our approach, each delta model only modifies one hierarchy at a time. As a result, having changes in multiple hierarchies will lead to creating multiple delta models. In comparison, populating the changes in one hierarchy requires creating only one delta model.

***LL4* - Visualization support for delta models**. After several workshops, we learned that control software engineers found it difficult to use a textual language when creating delta models. Engineers in practice mostly use a graphical DSML for developing the control software. Thus, using a textual language particularly makes it hard to see the relations between delta operations and function blocks. Furthermore, the engineers are already accustomed to the usability of a graphical editor. For instance, the engineers are unable to see in the delta model text file where the new `EvtMerger` FB and its connections will be added when the delta model in Listing 3 is applied to the core control software. To address this challenge, we plan to provide a visualization in V4rdiac which allows engineers to see which FB instances or connections will be added or removed by deltas and where.

***LL5* - Tool support for creating delta models**. Creating delta models for real-world control software is error-prone and cumbersome [37]. For instance, the engineer must identify a valid pair of source and destination interfaces to define a connection. All these interfaces must be manually typed in a textual representation (cf. Listing 3). In comparison, users only need to drag a line from a source to a destination interface in the 4diac IDE graphical editor. Tool support to create delta models using a graphical editor could significantly improve V4rdiac's usability. Another feedback from the control software engineers was to (semi-)automatically mine the delta models by comparing multiple control software variants. We have already started to address this feedback by implementing a modification recording operation in 4diac IDE and investigating different graph and model comparison algorithms [9].

```
delta DRemoveProp;
uses Valve.MainFunc.MachineAlg;
{
    <Remove> NetworkElement name=PropNCM;
    <Remove> NetworkElement name=PropCalc;
    <Remove> NetworkElement name=PropDIF;
    <Remove> NetworkElement name=PropMERGE;
    <Remove> NetworkElement name=Prop_LIM;
    <Add> FB name=EvtMerger type=RT_E_REND;
    <Add> EventConnection source=Comm_NSW.CNF
            dest=EvtMerger.E1
    <Add> EventConnection source=ServoLIM.CNF
            dest=EvtMerger.E2
}
```

Listing 3: IEC 61499 textual delta model example to derive a Servo Valve from the Valve Core Control Software.

4.4 Delta Models and Variation Points Mapping

We performed this step together with the same engineers from previous activities. At first, we demonstrated how to create the mapping between delta models and variation points (cf. Listing 4). After that, we gathered engineers' feedback on the usability of creating this mapping in V4rdiac.

```
productline MetallurgicalPL;
variations Valve#Overview#Type#Servo,
           Valve#Overview#Type#Proportional,
           Pump#Type#Circulation#CoolingSystem;

delta DServo when Valve#Overview#Type#Servo;
delta DProportional when Valve#Overview#Type#Proportional
    ;
delta DPumpCoolingSystem after (DValveAssembly &&
    DTankAssembly) || (DPressure || DCirculation) when
    Pump#Type#Circulation#CoolingSystem;
```

Listing 4: A delta configuration file example specifying several application conditions.

***LL6* - Visual support for creating the delta configuration**. The control software engineers commented that defining this mapping in V4rdiac is a cumbersome task and makes them reluctant to adopt our approach for daily use. Currently, several delta models are related to multiple variation points defined in variability models with complex relations. Furthermore, these complex relations also exist when one or more delta models must be executed in a certain order. For instance, in Listing 4, `DPumpCoolingSystem` can be executed if one of the following conditions is satisfied: (1) `DValveAssembly` and `DTankAssembly` have been executed, or

(2) `DPressure` or `DCirculation` have been executed. Additionally, the engineers commented that it is difficult to trace the configuration and delta modules using the current textual representation. To address these issues, engineers asked us to provide a visualization in which they can observe and create the mapping between variation points in variability models and delta models.

4.5 Application Engineering

Most of our collaboration effort is currently allocated to domain engineering (cf. Fig 2). We frequently perform interviews and workshops to create the variability models and the delta models for the industrial CPPSs. However, this process prevents us from fully performing the application engineering. Thus, so far, we have mainly demonstrated how V4rdiac's configurator can generate a part of the control software based on already existing variability models and delta models. After the demonstration, we captured the engineers' feedback.

***B4* - Configuring control software based on explicitly defined rules/constraints**. The engineers commented that having a software configurator based on a variability model ensures their configuration process is valid according to predefined rules/constraints. Currently, engineers perform the validation process manually by looking at the engineering artifacts produced for each variant. Thus, automating this process can guide the engineers to minimize inconsistencies when selecting variation points from multiple components and disciplines. Fig. 8 depicts V4rdiac's product configuration interface, showing the step to configure the pump component. The users can go to the next configuration step using the `Next` button when the selected options are valid according to the *local validator* (`Local` label displaying true). Furthermore, the `Finish` button will be enabled when the selected options from all the configuration steps are valid according to the *global validator*. When the users click the `Finish` button, the *4diac Generator* component immediately generates a control software according to the selected configuration options (cf. Fig. 3).

***B5* - Automatically generating control software variants**. Having a software generator is beneficial for our industry partner to speed up their development process. The engineers commented that V4rdiac could at least assist them in providing an initial template for developing control software. Since this initial template already reflects the selected configuration options, the development effort will be reduced. The engineers can focus more on fine-tuning the generated control software in further development phases (e.g., commissioning). In practice, the fine-tuning process depends on the project requirements and can range from changing parameters in one or more FBs, to minor adjustments in the FB network, to additional development.

***LL7* - Positioning of function blocks is important**. We learned that the position of FB instances and connections in the network affects control software engineers' understanding of the control software. When developing control software, control software engineers apply a certain graphical pattern defining where they put the FB instances in the network. For instance, the engineers put all FB instances related to the `Proportional` valve in the same area (cf. Fig 7). V4rdiac currently does not provide any mechanism to specify a specific location where newly added FBs will be positioned in a network. As a result, the engineers commented that they could not easily see where the changes from the delta models are located, increasing the complexity of verifying whether the generated variants are valid. We thus added two solutions to address this particular issue: (1) we now allow specifying a *relative* location in an <Add> FB delta operation and (2) engineers can use a clustering algorithm to group FB instances that are related in the same area. Our general approach is to place newly added FB instances nearby existing FBs in the core control software. Additionally, the placement of existing FB instances must remain unchanged to ensure the pattern imposed by the engineers still exists after the generation process.

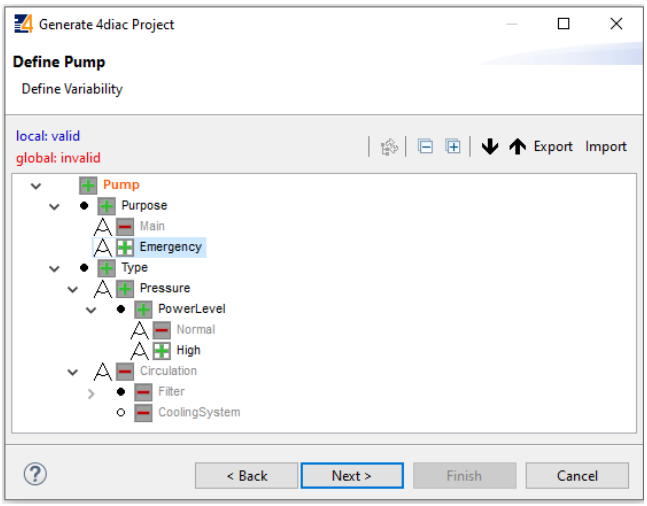


Figure 8: V4rdiac product configuration interface.

***LL8* - Generate control software variants based on multiple configuration files**. This lesson also compliments *LL2* regarding the need for supporting incremental product configuration. In particular, we need to better separate the product configuration and the control software generation processes. We will modify V4rdiac to create a configuration file representing a set of selected options whenever the product configuration process for each stepwise configuration file has been finished. The users can then input all the created configuration files to automatically generate a control software variant for the entire system. By doing this, the engineers can perform the incremental configuration process for each machine component and store the result in a configuration file. After all the machine components are configured, the engineers can invoke the control software generation process to generate the control software for the overall machine's functionality.

5 RELATED WORK

Several works have proposed describing industrial systems' variability using multiple variability models. Safdar et al. [32] developed a conceptual framework that supports multi-step and multi-stage production configuration for CPSs. This conceptual framework defined a conceptual model where we can relate one or more variability models with a base model and resolution models that form all the necessary artifacts to build the CPS. Fang [10] proposed a multi-view modeling approach for expressing the variability of industrial

automation management systems. This approach uses a feature model to extend the existing model commonly used for expressing topological and process views. As a result, users can now express variability within the topological and process model. Meixner et al. [24, 25] extend the Product-Process-Resource (PPR) language also to describe product, production process, and production resource variability in the CPPS domain. Existing interdisciplinary product line approaches propose using multiple feature models when describing the overall variability in the manufacturing [12, 20] and industrial automation [30] domains.

Additionally, existing tools and approaches can express software variability using multiple variability models. For instance, pure::variants [4] is a commercial tool allowing users to create multiple feature models to describe the overall system's variability. Clafer [2, 19] is a unified modeling language that unifies feature and class modeling to describe structural and behavioral variability. InVar [15] is a toolchain to integrate heterogeneous (types of) variability models for expressing system variability.

In summary, all these works allow using one or more variability models when expressing a system's variability. Each variability model represents a specific view, dimension, aspect, or engineering discipline involved in the system's development. However, no approach except InVar and our approach allows using different types of variability models. Our V4rdiac approach specifically focuses on CPPS variability. Each engineering discipline can use its preferred variability model type. Compared to InVar [15], we use a different mechanism to relate variability models. In InVar, each variability model must have an Inter-Model Dependency Information (IMDI) file for expressing the relation between different variability models. V4rdiac relies on an existing variability transformation framework TRAVART [11] to automatically transform different (types of) variability models. We use this framework to transform the different (types of) variability models into a standardized representation. The standardized representation is then used for defining cross-discipline constraints and enacting configuration options. V4rdiac can be extended to deal with custom variability representations by relying on TRAVART.

Several variability realization mechanisms for model-based systems engineering have been applied in industry [34]. While clone-and-own, parameterization, or custom-made mechanisms are still widely used in industry to deal with variability [3, 13], adopting a systematic variability mechanism for industrial control software domain is still at the early phase. For instance, existing work uses a feature model to describe variability within AutomationML [38] to express variability within multiple engineering artifacts or models. Another work [28] uses a feature model and augmented feature-to-code mapping to express control software variants developed using the IEC 61131-3 standard. Existing work also extended decision modeling to be used in IEC 61499-based control software to configure which FBs can exist inside the application model [14]. Furthermore, an approach that uses feature modeling and delta modeling has also been proposed to express Matlab/Simulink-based control software variability [17]. In comparison, our V4rdiac approach uses delta modeling to express IEC 61499-based control software. We also provide a mechanism to relate the delta models and variation points from the variability models. This relation is later used to derive IEC 61499-based control software given a set of configuration options selected during V4rdiac product configuration.

6 CONCLUSION & FUTURE WORK

In this paper, we described our experiences of managing CPPS control software variability using V4rdiac approach. We performed a series of workshops with engineers of our industry partner to assess how V4rdiac can be used in industry. These workshops focused on performing domain engineering for expressing the overall CPPS variability and the related control software variability using V4rdiac. We also demonstrated how V4rdiac could generate (parts of) CPPS control software. We conclude that V4rdiac provides several benefits including improvement of CPPS engineering knowledge, formalizing tacit variability knowledge, and a mechanism to generate variants of control software. We also identified several lessons learned, including the need for visualization support and incremental product configuration. These lessons learned are important for enhancing V4rdiac's usability and further ease CPPS engineering process.

In our future work, we will prioritize addressing the challenges from our lessons learned. We will provide a mechanism to express and evaluate more complex constraints between variation points defined in multiple variability models. We will implement visualization support for IEC 61499 delta modeling. Additionally, we will improve the language and provide editor support for creating cross-discipline constraints. We also plan to further improve the process of variability elicitation for different CPPS engineers. To achieve this, we investigate possible mechanisms to derive feature models based on engineering artifacts automatically. In parallel, we are investigating different graph-based and model-based comparison algorithms to provide a (semi-)automatic delta model mining mechanism based on comparing multiple variants of IEC 61499-based control software. Furthermore, we are also investigating product line evolution mechanism to reflect changes made during commissioning back to delta and variability models.

ACKNOWLEDGMENTS

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged. We explicitly want to thank our industry partner for their continuous support. In writing this paper, we used Grammarly and ChatGPT to correct and improve the grammar of our writing. We have rephrased, corrected, and extended this generated text.

REFERENCES

[1] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation.* Springer.
[2] Kacper Bak, Zinovy Diskin, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. 2016. Clafer: unifying class and feature modeling. *Softw. Syst. Model.* 15, 3 (2016), 811–845. https://doi.org/10.1007/s10270-014-0441-1
[3] Thorsten Berger, Jan Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. 2020. The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering* 25, 3 (2020), 1755–1797.
[4] Danilo Beuche. 2019. Industrial Variant Management with Pure::Variants. In *Proceedings of the 23rd Int'l Systems and Software Product Line Conf. - Volume B*

(Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 37–39.

[5] Stefan Biffl, Detlef Gerhard, and Arndt Lüder. 2017. *Introduction to the Multi-Disciplinary Engineering for Cyber-Physical Production Systems*. Springer International Publishing, Cham, 1–24.

[6] Hafiyyan Sayyid Fadhlillah, Kevin Feichtinger, Philipp Bauer, Elene Kutsia, and Rick Rabiser. 2022. V4rdiac: Tooling for Multidisciplinary Delta-Oriented Variability Management in Cyber-Physical Production Systems. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B* (Graz, Austria) *(SPLC '22)*. Association for Computing Machinery, New York, NY, USA, 34–37.

[7] Hafiyyan Sayyid Fadhlillah, Kevin Feichtinger, Kristof Meixner, Lisa Sonnleithner, Rick Rabiser, and Alois Zoitl. 2022. Towards Multidisciplinary Delta-Oriented Variability Management in Cyber-Physical Production Systems. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* (Florence, Italy) *(VaMoS '22)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages.

[8] Hafiyyan Sayyid Fadhlillah, Shubham Sharma, Antonio Manuel Gutiérrez Fernández, Rick Rabiser, and Alois Zoitl. 2023. Delta Modeling in IEC 61499: Expressing Control Software Variability in Cyber-Physical Production Systems. In *28th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2023, Sinaia, Romania, September 12-15, 2023*. IEEE.

[9] Hafiyyan Sayyid Fadhlillah, Shubham Sharma, Rick Rabiser, and Alois Zoitl. 2022. Supporting Variability Management in Cyber-Physical Production Systems: Towards Semi-Automatic Delta Model Mining for IEC 61499. In *27th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2022, Stuttgart, Germany, September 6-9, 2022*. IEEE, 1–4. https://doi.org/10.1109/ETFA52439.2022.9921660

[10] Miao Fang. 2019. *Model-Based Software Derivation for Industrial Automation Management Systems*. Ph. D. Dissertation. Technische Universität Kaiserslautern.

[11] Kevin Feichtinger, Johann Stöbich, Dario Romano, and Rick Rabiser. 2021. TRAVART: An Approach for Transforming Variability Models. In *15th Int'l Working Conf. on Variability Modelling of Software-Intensive Systems* (Krems, Austria) *(VaMoS'21)*. ACM, New York, NY, USA, 8:1–8:10.

[12] Stefan Feldmann, Christoph Legat, and Birgit Vogel-Heuser. 2015. Engineering support in the machine manufacturing domain through interdisciplinary product lines: An applicability analysis. *IFAC-PapersOnLine* 48, 3 (2015), 211–218.

[13] Juliane Fischer, Safa Bougouffa, Alexander Schlie, Ina Schaefer, and Birgit Vogel-Heuser. 2018. A Qualitative Study of Variability Management of Control Software for Industrial Automation Systems. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 615–624. https://doi.org/10.1109/ICSME.2018.00071

[14] Roman Froschauer, Alois Zoitl, and Paul Grünbacher. 2009. Development and adaptation of IEC 61499 automation and control applications with runtime variability models. In *Proc. of the 2009 IEEE Int'l Conf. on Industrial Informatics*. IEEE, 905–910.

[15] José A. Galindo, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, and Paul Grünbacher. 2015. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology* 62, 1 (2015), 78–100.

[16] Volkan Gunes, Steffen Peter, Tony Givargis, and Frank Vahid and. 2014. A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems. *KSII Transactions on Internet and Information Systems* 8, 12 (December 2014), 4242–4268. https://doi.org/10.3837/tiis.2014.12.001

[17] Arne Haber, Carsten Kolassa, Peter Manhart, Pedram Mir Seyed Nazari, Bernhard Rumpe, and Ina Schaefer. 2013. First-Class Variability Modeling in Matlab/Simulink. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-Intensive Systems* (Pisa, Italy) *(VaMoS '13)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages.

[18] International Electrotechnical Commission (IEC), TC65/WG6. 2012. IEC 61499-1, Function Blocks - part 1: Architecture: Loveneion 2.0.

[19] Paulius Juodisius, Atrisha Sarkar, Raghava Rao Mukkamala, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. 2018. Clafer: Lightweight Modeling of Structure, Behaviour, and Variability. *CoRR* abs/1807.08576 (2018). arXiv:1807.08576 http://arxiv.org/abs/1807.08576

[20] Matthias Kowal, Sofia Ananieva, Thomas Thüm, and Ina Schaefer. 2017. Supporting the Development of Interdisciplinary Product Lines in the Manufacturing Domain. *IFAC-PapersOnLine* 50, 1 (2017), 4336–4341.

[21] Sami Lazreg, Vladyslav Bohlachov, Loveneesh Rana, Andreas Hein, and Maxime Cordy. 2022. Variability-Aware Design of Space Systems: Variability Modelling, Configuration Workflow and Research Directions. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* (Florence, Italy) *(VaMoS '22)*. Association for Computing Machinery, New York, NY, USA, Article 4, 10 pages.

[22] Kwanwoo Lee, Kyo Chul Kang, and Jaejoon Lee. 2002. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In *Software Reuse: Methods, Techniques, and Tools, 7th International Conference, ICSR-7, Austin, TX, USA, April 15-19, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2319)*, Cristina Gacek (Ed.). Springer, 62–77.

[23] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.

[24] Kristof Meixner, Kevin Feichtinger, Rick Rabiser, and Stefan Biffl. 2022. Efficient Production Process Variability Exploration. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* (Florence, Italy) *(VaMoS '22)*. Association for Computing Machinery, New York, NY, USA, Article 14, 9 pages.

[25] Kristof Meixner, Rick Rabiser, and Stefan Biffl. 2019. Towards modeling variability of products, processes and resources in cyber-physical production systems engineering. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B*. ACM, 49–56.

[26] László Monostori. 2014. Cyber-physical Production Systems: Roots, Expectations and R&D Challenges. *Procedia CIRP* 17 (2014), 9–13. Variety Management in Manufacturing.

[27] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.

[28] Herbert Prähofer, Daniela Rabiser, Florian Angerer, Paul Grünbacher, and Peter Feichtinger. 2016. Feature-oriented development in industrial automation software ecosystems: Development scenarios and tool support. In *2016 IEEE 14th Int'l Conf. on Industrial Informatics (INDIN)*. IEEE, 1218–1223.

[29] pure-systems GmbH. 2022. pure::variants User's Guide Version 5.0.10.685. Online Document. https://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf Last Access 2023-04-02.

[30] Daniela Rabiser, Herbert Prähofer, Paul Grünbacher, Michael Petruzelka, Klaus Eder, Florian Angerer, Mario Kromoser, and Andreas Grimmer. 2018. Multi-purpose, multi-level feature modeling of large-scale industrial software systems. *Software and Systems Modeling* 17, 3 (2018), 913–938.

[31] Rick Rabiser and Alois Zoitl. 2021. Towards Mastering Variability in Software-Intensive Cyber-Physical Production Systems. *Procedia Computer Science* 180 (2021), 50–59.

[32] Safdar Aqeel Safdar, Hong Lu, Tao Yue, Shaukat Ali, and Kunming Nie. 2021. A framework for automated multi-stage and multi-step product configuration of cyber-physical systems. 20, 1 (2021), 211–265.

[33] Ina Schaefer. 2010. Variability Modelling for Model-Driven Development of Software Product Lines.. In *Proc. of the 4th Int'l Workshop on Variability Modelling of Software-Intensive Systems*. ICB-Research Report 37, Universität Duisburg-Essen 2010, 85–92.

[34] Andreas Schäfer, Martin Becker, Markus Andres, Tim Kistenfeger, and Florian Rohlf. 2021. Variability Realization in Model-Based System Engineering Using Software Product Line Techniques: An Industrial Perspective. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A*. Association for Computing Machinery, New York, NY, USA, 25–34.

[35] Daniel Stock, Daniel Schel, and Thomas Bauernhansl. 2019. Cyber-Physical Production System Self-Description-Based Data Access Layer. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 168–175.

[36] Michael Tiegelkamp and Karl-Heinz John. 2010. *IEC 61131-3: Programming industrial automation systems*. Springer.

[37] Birgit Vogel-Heuser, Jakob Mund, Matthias Kowal, Christoph Legat, Jens Folmer, Sabine Teufl, and Ina Schaefer. 2015. Towards interdisciplinary variability modeling for automated production systems: Opportunities and challenges when applying delta modeling: A case study. *Proc. of the 2015 IEEE Int'l Conf. on Industrial Informatics* (2015), 322–328.

[38] Manuel Wimmer, Petr Novák, Radek Šindelár, Luca Berardinelli, Tanja Mayerhofer, and Alexandra Mazak. 2017. Cardinality-based variability modeling with AutomationML. In *2017 22nd IEEE Int'l Conf. on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 1–4.

[39] Tao Yue, Shaukat Ali, and Bran Selic. 2015. Cyber-Physical System Product Line Engineering: Comprehensive Domain Analysis and Experience Report. In *Proceedings of the 19th International Conference on Software Product Line* (Nashville, Tennessee) *(SPLC '15)*. Association for Computing Machinery, New York, NY, USA, 338–347.

[40] Alois Zoitl and Robert Lewis. 2014. *Modelling control systems using IEC 61499. 2nd Edition*. Institute of Electrical Engineers.

[41] Alois Zoitl, Thomas Strasser, and Antonio Valentini. 2010. Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study. In *2010 IEEE Int'l Symp. on Industrial Electronics*. IEEE, 3817–3819.