VarApp: Variant Management App for IEC 61131-3 Compliant Legacy Software

Juliane Fischer
Chair of Automation and Information Systems
Technical University of Munich
Garching, Germany
juliane.fischer@tum.de

Birgit Vogel-Heuser
Chair of Automation and Information Systems
Technical University Munich
Garching, Germany
vogel-heuser@tum.de

Elisabet Estévez Estévez
Ingeniería de Electrónica y Automática
Universidad de Jaén
Jaén, Spain
eestevez@ujaen.es

Markus Male
Chair of Automation and Information Systems
Technical University of Munich
Garching, Germany
markus.male@tum.de

***Abstract*—Despite its many drawbacks, clone & own is still frequently used for control software development in the domain of automated production systems (aPSs). However, this procedure leads to a high amount of so-called legacy software variants, which are usually unmanaged and, thus, difficult to maintain or reuse. To facilitate the reuse of existing software, this paper presents a model-based approach for documentation, configuration, and derivation of IEC 61131-3 compliant control software. Thereby, the required models are stored in a NoSQL database and represent the structure and the variability of the considered aPS from a functional viewpoint. A visualization of the variability in the style of a software product line, which is commonly used in the informatics domain for the development of variant-rich systems, enables the user to configure a variant and the respective IEC 61131-3 compliant control software is generated automatically through model transformations. The developed concept is evaluated with a prototypically implemented web application and the control software variants of a laboratory demonstrator. Thereby, the web application supports the integration of variants from existing legacy software into the database model.**

Keywords—variant management, code configuration, legacy software, IEC 61131-3, model-based development, planned reuse, software product line

I. Control Software Development

Automated Production Systems (aPS) are long-living and highly variant-rich systems, which usually are controlled by programmable logic controllers (PLCs) programmed following the IEC 61131-3. Thereby, variants may arise from customers, who request different or additional functionality, and from different disciplines, e.g., mechanics, electrics/electronics, and software. Moreover, globally operating manufacturers often have to deliver aPSs with different PLCs for different markets, resulting in additional variants. Thereby, the interdisciplinary character of aPSs complicates the variant and version management of these systems, as variants from all disciplines need to be taken into account [1]. A recent study confirms that variant and version management of automation software controlling these aPSs is still a challenge [2]. For example, companies in the aPS domain often do not use any variant management or face challenges in the recognition and administration of variants. These challenges include, among others, arising of new variants during the lifecycle of a machine type, the lack of a global knowledge base including existing variants, and the high demand for customer-specific variants, which often influence all disciplines involved. Current variant management tools available on the market do not cope sufficiently with these challenges [2].

Although PLC platform developers start offering cloud-services linked to their development environment for central storage of projects including variants and versions, it is not yet state-of-practice in industry. This in combination with the lack of suitable variant management approaches hampers the planned reuse of control software artifacts. Thus, in the development of new control software variants or the adjustment of existing variants, the so-called "clone and own" approach is still commonly used despite its well-known drawbacks. In this unplanned reuse approach, parts of existing control software projects are copied manually to the project under development and are modified according to the respective requirements. However, this task is time-consuming and requires detailed knowledge of existing product variants [3]. To avoid the error-prone, unplanned development with clone and own, which results in legacy software systems that are hard to maintain, paradigms such as model-based engineering or model-based development present alternative solutions. However, the focus of both often lies in the development of aPSs from scratch and not on the development based on legacy software reuse.

For managing variants and versions arising throughout the development of software-intensive, embedded systems, software product line engineering (SPLE) has proven a successful approach in computer science and the automotive domain [4]. The approach is usually applied to systems programmed with high programming languages. Thereby, the commonalities and variabilities of a system are defined in a software product line (SPL) to enhance the planned reuse of software artifacts. Attempts to derive SPLs of aPS legacy software are promising. However, their current focus is variability documentation rather than reuse of software [5].

To address the challenges regarding the documentation and planned reuse of legacy control software, this paper introduces a database-assisted variant management approach, which enables the documentation, configuration, and generation of IEC 61131-3 based control software. Thereby, the model-based concept includes two general types of variability models: firs, a machine product line model (PLM), which enables the functionality-oriented modeling of aPS variants to document discipline-independent variability. Secondly, discipline-specific models (DSMs), e.g., the control

software variants focused in this paper, which are linked to the machine PLM. This link contains information on the dependencies between the function-oriented variability and the resulting, discipline-specific variability in the software DSM. It is the basis for planned reuse from DSM parts, i.e., legacy software, according to the configured PLM functionality. The developed concept is evaluated with a prototypically implemented web application.

The main contribution of this paper is an approach to support the centralized variability management of IEC 61131-3 legacy control software, including variant documentation, configuration, and code generation (thereby reusing existing code snippets). The remainder of this paper is structured as follows: Section 2 derives and presents the targeted requirements. Subsequently, section 3 provides background information. State of the art is summarized in section 4, followed by the concept presented in section 5 and its evaluation. Section 7 concludes the paper concludes with summary and outlook.

II. Derived Requirements

The absence of a global knowledge base containing all variants hinders the planned reuse of legacy software due to the lack of overview [2]. Thus, requirement 1 is the establishment of a centralized variability management functioning as a global knowledge base. As usually multiple projects are under development simultaneously, it is crucial that this knowledge base is centrally accessible for various developers (RQ1 global, central knowledge base).

To develop variant-rich systems in the automotive domain, the SPLE approach is successfully applied and, thereby, the used SPLs, which represent mandatory, optional, and alternative software parts, convince through their simplicity. Further, SPLs are the basis for user interaction when developing new variants from existing, partial solutions. Therefore, the documentation of variability in legacy control software is based on the SPL concept, which requires a comprehensive visualization of the product line (RQ2 comprehensible visualization).

To enable the variant management concept application to PLC software from different vendors, to guarantee the long-term readability of the used format and to reduce the amount of required transformations for documenting, configuring and generating (legacy) control software, the use of open formats and technologies is crucial, resulting in requirement 3 (RQ3 open formats and technologies).

According to a recent survey [6], the benefits users expect from variability management include apart from managing existing variability, also *product configuration*. In the considered context, product refers to one particular PLC software (not including variability) for a specific aPS out of an SPL of similar PLC software variants (including variability). This requires selecting, saving, and validating a configuration in the discipline-independent machine PLM (RQ4 configuration of a product variant).

Moreover, the survey [6] revealed that more than half of the participants expect variability management to support the *derivation of products*. Regarding the scope of this paper, this means, that from a selected configuration, a correspondingly configured machine model, and the respective DSM (both models without variability) have to be derived automatically. (RQ5 enabling derivation of DSMs, e.g., software).

Finally, variant management aims to enable the reuse of legacy software in a planned manner to decrease the development time while increasing or, at least, maintaining software quality. Further, when establishing SPLs for planned reuse, the reuse of existing developments is common [6]. Hence, the concept needs to enable a developer to enrich existing legacy software with variability information (RQ6 planned reuse of existing legacy software).

III. Preliminaries On Software Models, Exchange Formats and Variability Modeling

To facilitate reuse, extension, and modifications to software systems, the object-oriented paradigm (OOP) was introduced. It aims at defining modular classes and objects, including their interfaces, to enable their reuse and extension by instantiation and inheritance. An established standard for the description of OOP systems is the graphical Unified Modeling Language (UML), which defines 13 diagrams and their containing elements to model software structure and behavior (cf. [7] for details). In order to support the modeling of more general systems, the Systems Modeling Language SysML was defined. Modeled information can be exchanged with the open, platform-independent exchange format *XML Metadata Interchange* (XMI), which is supported by various software development tools. XMI enables the exchange of development data via *eXtensible Markup Language* (XML), which is not limited to the aforementioned models, but generally encodes documents in a flexible, machine and human-readable, textual data format. Thereby, the text file is set up in a hierarchical style with so-called *tags* defining *element*s, their *attribute*s, and enclosing child nodes, i.e., child elements or text. In the scope of aPS, PLCopen XML (part of IEC 61131 [8]) specifies an XML-based exchange of PLC software between different PLC programming environments.

As introduced above, SPLE is an approach for the development of variant-rich systems in the informatics domain [4]. It contains two main steps: *domain engineering*, which includes the development of an SPL containing commonality and variability of a product, and *application engineering*, where an individualized product is built from the SPL. Thereby, variability is divided into *problem* and *solution space*: Lettner et al. define problem space as "systems' specifications established during domain analysis and requirements engineering" (comparable to the product requirements document in aPS development) and the solution space as the "concrete systems created during development" (comparable to scope statements in aPS development) [9]. To model a system's variability, various variability categories are defined: the feature-oriented domain analysis (FODA) [10] distinguishes *mandatory features*, *optional features*, and *alternative groups*. As mandatory features are not variable, two variability types remain: A binary presence/absence variability (optional feature) and a selection of "specializations of a more general category" (groups) [9]. Cardinality-based feature models (CBFM) add two additional conceptions [11]. The multiplicity (cardinality) of a single feature, and groups out of which more or less than one feature can be selected. Both feature and group cardinality are constraint by cardinality ranges. Finally, the SimPL methodology provides a very compact taxonomy of variability types in OOP models [12]. Thereby, each system variation can be expressed by one of the following variability classes: Cardinality variability, attribute variability, topology variability, and type variability [12].

With this background on models, exchange formats, and variability categories, the following section presents the state of the art in PLC software variant modeling, configuration, and derivation.

IV. Related Literature Regarding variant Management in the Domain of aPS

To support variability modeling in the aPS domain, a so-called delta modeling (DM) approach is presented in [1], which is applicable to any existing modeling language. The authors use *MoDEMAS* (an interdisciplinary modeling approach based on [13]) and extend it with deltas, which are used to explicitly model changes to a designated base system. Thereby, a delta is able to either add or remove modeling elements. A complete product family is modeled according to DM by specifying a base model and, for each variant or version, a set of delta operations, which are applied to the base sequentially to create a new variant. Although this approach supports variant management of models, it is not centralized. The connection to or integration of DSMs or at least software is not mentioned; thus, neither an automated derivation of configured software nor utilization of legacy software is supported. The utilized modeling techniques (cf. [13]) are openly available, and special applications or tools are not specified. When combining the delta approach with SPLs, visualization, configuration, and potentially even derivation of software are enabled. However, legacy software variants are only reusable if their differences are stored as delta sets.

Biffl et al. [14] propose an approach for interdisciplinary version management in the domain of aPS based on the XML-based standard AutomationML [15], designed initially to exchange engineering data between different tools. An enlargement of this approach includes cardinality-based variability modeling [16]. Variability and product lines are described in system unit classes and based on plant topology information, which is described in the XML-based format CAEX. To represent variation points, the authors define a CAEX role class library containing role classes such as *Cardinality*, *OR_Choice*, *XOR_Choice*, *Mandatory*, *Optional*, *Implies*, and *Restricts* from FODA. Configurations out of a product line are enabled via Instance Hierarchies and verifiable with OCL constraints, thus, supporting the user, but a suitable visualization is not explicitly targeted. The variability management is based on open formats. However, the authors neither specify a concept for the reuse of legacy software (albeit AutomationML utilizes PLCopen XML) nor for automated software derivation. Furthermore, the concept itself does not provide centralized variability models.

An approach to apply SPLE in the domain of aPS while considering the multi-disciplinary character of machines and plants is the use of interdisciplinary product lines (IPLs) described in [17]. The authors differentiate the function-based variability of aPS *from a customer's point of view* and the discipline-specific *variability from a developer's point of view*, with an individual SPL representing each view. The configuration of a variant in the customer SPL results in the respective selection of features in the discipline-specific SPLs, which are linked via a mapping matrix. For the SPL implementation, open formats are useable. The approach suggests connecting the discipline-specific SPL with a global knowledge base of the system containing discipline-specific components, e.g., the software SPL with a library containing PLC software module variants. Thus, the variant management is not centralized but split between a central SPL and tools used in the considered disciplines. The configuration of a variant leads to the indication of required library modules, but the software is not derived automatically. Further, the considered granularity is limited to the coarseness of library modules, which hampers the reuse of legacy software that is often not well-structured into reusable modules.

Fang et al. present a model-based approach [18] that aims at supporting the derivation of software in the domain of industrial automation. The authors consider different stakeholders and, therefore, use different models to enable feature, topology, and process configuration. During the derivation process, the models are instantiated based on a feature configuration. Subsequently, they can be adapted and refined and, finally, are used to generate code. The approach specifies repositories to hold software fragments but does not detail if these are centralized. However, the combination of all models can be considered as a global knowledge base. The feature model provides a comprehensible representation of the product line and enables the configuration of a product variant via feature selection. From the configuration, the software is derived automatically. Although the reuse of legacy software is contemplated, no specific support is given. The approach does not utilize open formats and technologies; instead, plugins for a commercial tool are purpose-built.

To handle the demand for customer-specific product variants, software ecosystems (SECOs), i.e., interrelated SPLs, are used. Thereby, specialized SPLs are derived from (a subset of) features of a core SPL and enlarged, modified, and configured according to customer needs [19]. Hinterreiter et al. present an approach including version management of features to support the evolution of these SECOs and, therein, reuse of existing software artifacts [19]. The approach bases on open formats and technologies. Furthermore, it applies feature models to configure and derive software variants from legacy software artifacts, which are stored in a repository. To enable feature-to-code mappings, the ECCO (Extraction and Composition for Clone-and-Own) approach is used, which supports software engineers in applying clone-and-own [3]. Albeit ECCO enables the reuse of legacy software through feature-to-code mappings, the granularity of the features and, thus, reusable software artifacts depends on the granularity of commonalities and differences in legacy software variants. Furthermore, the approach presents a global, but not centralized knowledge base.

An approach for interdisciplinary reuse utilizing feature models in process engineering is presented in [20]. Thereby, the authors distinguish between project-dependent and project-independent engineering and propose a function-oriented methodology. In the approach, a feature model represents reusable components in the problem-space that are functionally decomposed. Furthermore, a family model (SPL) is used to represent discipline-specific, reusable artifacts. Links between feature and family model are established, whereby the concept theoretically enables links to several discipline-specific family models. By utilizing feature and family models, the concept uses open formats. However, the aspect of central accessibility is not addressed. Variability visualization and variant configuration are supported by a commercial tool, which generally enables the derivation of software. However, neither the reuse of legacy software nor the derivation of software is specifically targeted within [20].

The family mining approach presented by Schlie et al. [5] compares two IEC 61131-3 variants using tailored metrics to

reverse engineer an SPL visualizing commonalities and differences of two compared legacy software projects. The resulting SPL represents a global knowledge base on the considered legacy software in a comprehensive manner and is based on the open format PLCopen XML. However, the approach is currently limited to pairwise variant comparisons, and neither considers the configuration nor the derivation of PLC software as its focus is the automatic documentation of variability in legacy software. Furthermore, the variability information is not centrally accessible.

Overall, no approach fulfills all requirements derived in Section II (cf. Table I). To bridge this gap between state of the art and (industrial) requirements, the subsequent section presents a concept for centralized variant management, configuration, and reuse of IEC 61131-3 legacy software.

V. Model-based Concept for Documentation, Configuration and Derivation of Software Variants

This section presents an application example, introduces the developed concept, including its models and their interrelation, and, further, provides details on the developed *VarML* Profile for XML-based variability modeling.

A. Introduction of Pick and Place Unit (PPU)

The pick and place unit (PPU) is a laboratory demonstrator designed to study variability and evolution during the life cycle of aPS [21]. The PPU consists, depending on the required functionality, of several modules, such as stack, crane, conveyor, and stamp. In the scope of this paper, the crane and its variation examples are focused (cf. Fig. 1, left). It is used to transport workpieces (WPs) among the other PPU modules. Thus, depending on the presence or absence of modules and the required workflow, it must reach two to four positions. Crane positioning can be realized either by multiple, discrete position sensors or by an analog position sensor, i.e., a potentiometer. In the first case, the required number of contact switches depends on the number of crane positions. Generally, both hardware solutions realizing the positioning functionality influence the required control software.

B. Used Models and their interrelations

In the style of SPLE, the models used within the developed concept are separated into problem space (supra-disciplinary product line model (PLM)) and solution space (DSMs), cf. Fig. 2. Thus, there is no knowledge about DSMs required to understand a system's structure and variability and to configure a customer-specific variant. Further, it is distinguished between domain and application engineering.

The concept consists of four interrelated model types (cf. Fig. 2). A PLM describes the machine line, such as the crane, including both its functional structure (representing all variants) and its variability. For the crane, e.g., the PLM

TABLE I. Fulfillment of Derived requirements In Literature

	Derived Requirements					
	R1	***R2***	***R3***	***R4***	***R5***	***R6***
Delta Modeling [1]	o	o	+	o	o	-/o
AutomationML[16]	-	o	+	(+)	-	-
Interdisciplinary SPLs [17]	o	+	(+)	+	-	-
Model-Based Derivation [18]	(+)	+	-	+	+	o
Software ecosystems [19]	o	+	+	+	+	(+)
Feature Models [20]	o	+	(+)	+	(+)	o
Family Mining [5]	o	+	+	-	-	-

Legend: fulfilled: +; partially fulfilled: o; not fulfilled / targeted: -

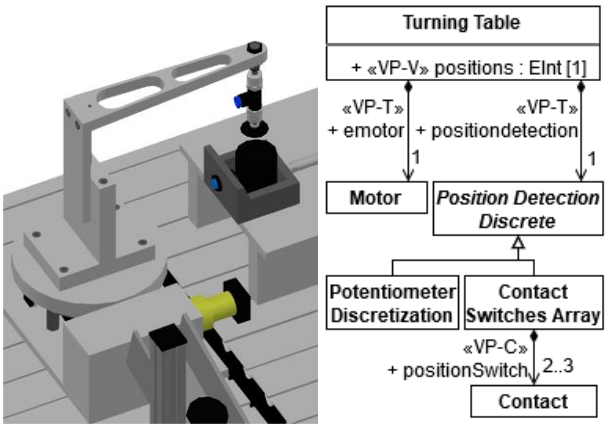

Fig. 1 Overview on PPU module crane and its variability model

includes a variation point *position detection* with the variants *contact switches* or *potentiometer*. The PLM is interrelated with one or more product line DSMs representing a specific discipline each, which define the discipline-specific fragments within the machine line and how they are applied according to the machine line's variability. For the crane example, one product line DSM (from the software discipline) defines software for both, *contact switches* and *potentiometer* and how it relates to the PLM variants. Thereby, software elements may represent legacy software. In the tool-supported DSM reuse assistance, a user is enabled to reuse this legacy software and establish the links between DSM and PLM (RQ6).

Based on the PLM, configuration assistance supports the user in choosing a valid configuration (machine variant) (RQ4). To present the variability modeled in the object-oriented, well-defined, and specification-conform PLM to the user in a comprehensible manner, it is displayed in a feature-model like form in a user-interface (RQ2). In this view, the user may choose features for all variation points (in the crane example, either contact switches or potentiometer) and is equipped with functionality to validate the configuration and to derive a product DSM automatically (RQ5). Afterward, the derivation function retains all DSM fragments related to chosen variants and discards other fragments. For the crane, this function assembles control software, including either software for potentiometer discretization or switches. The

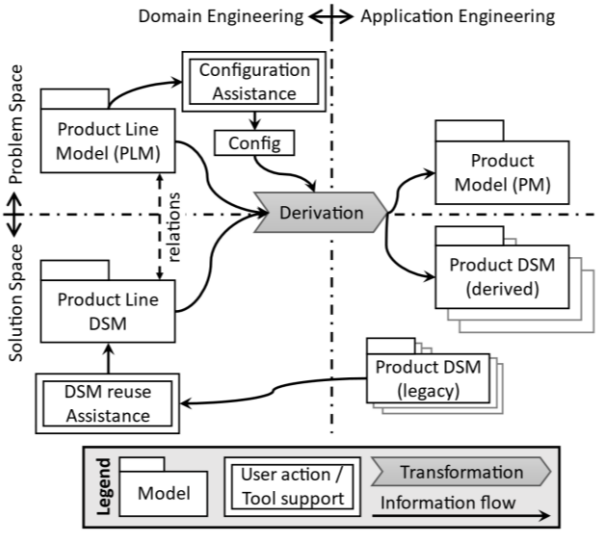

Fig. 2 Structure of the elaborated concept

models, at least PLM and product line DSMs as well as their mappings, are database-stored to fulfill the requirement for a global, central knowledge base (RQ1).

C. SysML-based Product Line Model (VarML Profile)

Although feature models are well recognized in domain engineering, they have limitations concerning the reuse (i.e., multiple uses) of model parts: To use a feature multiple times within a model, it has to be copied, whereby copies hold no reference to their source. Thus, if a feature is changed, every occurrence (copy) has to be changed manually. Applying OOP, this can be addressed by the concept of multiple instantiations of the same class. Therefore, within the developed concept, OOP is used to model the variability in problem space. However, despite their drawbacks regarding reuse of parts, feature models have an advantage regarding comprehensibility compared to other models. Thus, the developed concept merges both notions by modeling the system according to OOP and providing a feature-model-like view to configure the product. For this purpose, the profile *VarML* is introduced, which enhances the object-oriented modeling language SysML with variability mechanisms. Details on the developed *VarML* profile are presented below.

1) Classification: Cardinality, type and value variability

Different approaches to express variability have been defined (cf. sec. 2), each of them introducing different variability forms. Roughly, four different variability types can frequently be identified using different wordings for similar variability mechanisms (cf. Table II). In this concept, three kinds of variability, similar to those of SimPL[12], plus constraints are regarded sufficiently. In the context of OOP, considering structure variability within an object-oriented class, the possible variations are concentrated in the class' properties, which can differ in their type, classifier, and their value, defining the three types of variability used in this concept. Relations between variation points can be expressed using constraints (with group constraints as a special type).

As each variable property represents a variation point in the product line, an abstract stereotype *VP* (variation point) on properties is introduced as a base for the stereotypes for cardinality (*VP-C*), type (*VP-T*) or value (*VP-V*) variability (see Fig. 3). The VP stereotype owns an optional tag definition *name* to differentiate variation points within one product line from each other. Specializing *VP*, *VP-C* defines a variation point on a property's cardinality and owns an optional association *default* to ValueSpecification. Upper and lower cardinality boundaries are defined by standard UML cardinality's upper and lower values (cf. *positionSwitch* in Fig. 1, right). The *VP-T* stereotype expresses the variation of a property's classifier and is associated with an optional *default* classifier and several *options*. In Fig. 1, the type VP on

TABLE II. DIFFERENT APPROACHES FOR VARIABILITY EXPRESSION

FODA [10]	CBFM [11]	SimPL[12]	*VarML*
optional feature	feature cardinality	cardinality variability	cardinality variability
alternative group	group <1..1>	type variability	type variability
	group <m..n>		group constraint / type variability
-	-	attribute variability	value variability

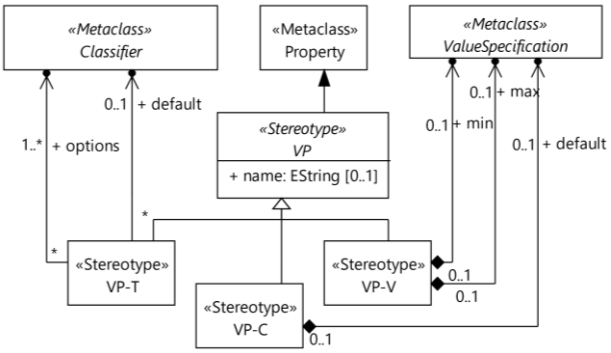

Fig. 3 VarML profile: Cardinality, type and value variability

positiondetection has the classifier options *Potentiometer Discretization* and *Contact Switches Array*. Both specialize the original classifier *Position Detection Discrete* of the property. Finally, *VP-V* expresses a variation on a property's value, especially for primitive types, using the default value from the property and specifying a *min* and *max* value (cf. *Turning Table.positions* in Fig. 1).

2) Constraints

Constraints are further restrictions for the configuration of a product, i.e., a machine from a PLM. For example, the presence of a particular variant may require or forbid the choice of another variant. They are essential in the configuration of a product in order to prevent invalid or physically impossible configurations (RQ 4). UML provides the metaclass constraints, which allows defining constraints in any language, thus providing more functionality than *require* and *exclude* constraints. The stereotype *Variability Constraint* marks constraints considered during the product configuration.

3) Groups

According to FODA [10], alternative groups consist of sub-features representing different variants of the same functionality (such as vacuum gripper or pneumatic gripper for workpiece picking). These groups are expressed in *VarML* by Type-Variability (*VP-T*). In contrast, other approaches consider alternative groups of elements with similar interfaces regardless of their functionality. As these groups are used frequently throughout literature, a special constraint is introduced to represent them. Fig. 4 shows the stereotype *VP-Group*, which extends the metaclass constraint. *VP-Group* additionally refers to *VP-T* properties as group members and *min* and *max* values for the number of group members.

4) XML representation of the product line model (PLM)

Variability models and system models are, even if based on a common metamodel, often stored differently in different tools. To find a common basis, which also allows for further processing of the models, a common storage format is necessary. A system model in SysML can be stored in an

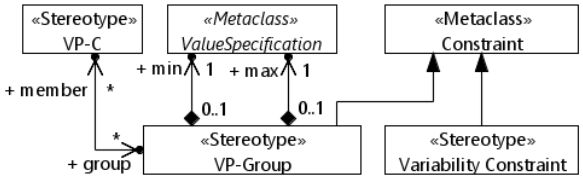

Fig. 4 Extract of the *VarML* profile illustrating group constraints

XML-based XMI file, but this file type is not very comprehensible and includes much information irrelevant for the variability model. Therefore, a simplified structure and variability model are expressed in an XML dialect, which can, later on, be used to store the model in the database. The introduced XML schema *VarML* specifies the metamodel of the XML representation and also is used for DSMs.

D. XML-based discipline-specific models (DSMs)

As introduced above, the discipline-independent PLM is linked to DSMs, which represent the discipline-specific implementations of the modeled variants, e.g., PLC legacy software. To enable the software's reuse, the XML-based, open exchange format PLCopen XML is used (RQ 3). For this reason, the integration of DSMs focusses on XML based model formats, which can also be found in other engineering disciplines such as Collada for CAD models. This section illustrates the storage of XML DSMs in a database and their enrichment with variability.

1) Document structure

To keep information about the original model file, the original format and namespace, and other metadata, the proposed *VarML* XML specification uses a header section (<varml:header/>). The header is only used in the databased, variability-incorporating product line DSM, and omitted when deriving a product DSM (configured variant). Nevertheless, data included in the header is used for derivation, for example, to set the proper file extension. The actual model is saved in a body (<varml:body/>) that can contain any namespace.

2) Variability Enrichment and Variant Derivation

To form a product line DSM, fragments of the original DSM documents are enriched with their variability within the body section. Starting with a plain DSM (legacy software), varying document parts are identified, marked, and linked to related variants in the PLM (DSM reuse, RQ6). Likewise, additional variants from different product DSMs can be added and linked later on to support model evolution. The XML elements for variability attribution are expressed in the *VarML* schema and, therefore, violate the original document schema temporarily: the resulting product line DSM contains XML fragments in the original DSM namespace, embedded in supplementary variability elements where necessary. During product derivation, fragments linked to a required variant are retained, while the others and the varml elements are omitted, resulting in a namespace-clear DSM combining variants in an all-new configuration (RQ5).

A *varml:reference* element connects parts of the XML document to a variation point in the PLM and states that the document part within varies with the linked variation point. For example, a reference element containing software for position detection by micro switches and by potentiometer is linked to the VP-T *positiondetection* (cf. Fig. 1, right). The *reference* element contains the attribute *id*, *variation-point*, which refers to a variation point in the PLM, and *xpath*. The attribute *xpath* defines the insertion point (the „where“) for contained document variants during variant derivation via a relative XPath. It is default ".", determining to insert the *reference* element. For example, "../@*{attributeName}*" states that the chosen value is to be inserted into the attribute *{attributeName}* of the parent element.

The insertion value (the "what"), on the other hand, is determined by several *varml:valueSelect* elements to choose from. Within a valueSelect, the child nodes represent the DSM fragments related to a variant, e.g., a software module for potentiometer discretization or a binary variable for a contact switch. The attribute *value* defines a possible value of the variation point (type, cardinality, or value of the property, according to the stereotype) for which the *valueSelect* element is selected. The definition of more sophisticated selection rules as XPath expression (return type xs:Boolean) is possible.

Fig. 5 illustrates an example of enrichment. The reference *ref1* points to the variation point *vpPosDet* (VP-T on property *positionDetection*, cf. Fig. 1). The insertion point is right within the *localVars* element (as *xpath* is default "."). During derivation, the *valueSelect* element *vs1* is selected if the variation point is configured to the value *cPotiDisc*. As *vpPosDet* is a type variation point, the property *VP-T positiondetection* is of type *Potentiometer Discretization* (id = cPotiDisc). In this case, the derivation process inserts the child elements of *vs1* in the place of *ref1*.

VI. Evaluation with Prototypical Implementation

This section introduces the developed prototypical web application VarApp, which is used together with the PPU to evaluate the presented concept. Based on the evaluation, the requirement fulfillment is discussed.

A. Prototypical Implementation: Web Application VarApp

For evaluation purposes, the presented concept is implemented as the web application VarApp. VarApp separates front- and backend to allow multiple users to access the same projects. The frontend (cf. Fig. 6 for an overview) implements model and document presentation as an HTML5 page, including javascript, to implement user actions and the connection to the backend. The web frontend allows using the application without any installation.

The backend is built upon the open-source NoSQL database eXist-db. EXist-db is a document-oriented XML database that supports not only the organized central storage of XML documents but also their processing via, e.g., the transformation language XSLT and the query languages XPath and XQuery. Especially XQuery scripts are used to implement functionality such as the generation and adaption of HTML pages as well as for the manipulation of XML documents. XSLT is used to implement transformations like the derivation of product DSMs. VarApp does not support the creation of PLMs as model editors more suitable for this task are freely available. Instead, an import transformation for XMI-stored models is provided.

B. Sample Configurations from PPU demonstrator

To evaluate the concept and its applicability, the implemented software prototype is applied on the PPU

```
<pou name="TurningTable" pouType="functionBlock">
  <interface>
    <localVars>
      <varml:reference id="ref1" variation-point="vpPosDet">
        <varml:valueSelect id="vs1" value="cPotiDisc">
          <variable name="Potentiometer">
            <type>
              <derived name="Potentiometer"/>
            </type>
          </variable>
        </varml:valueSelect>
          [...]
      </varml:reference>
```

Fig. 5 Detail of the variability-enriched PLC OpenXML of PPU

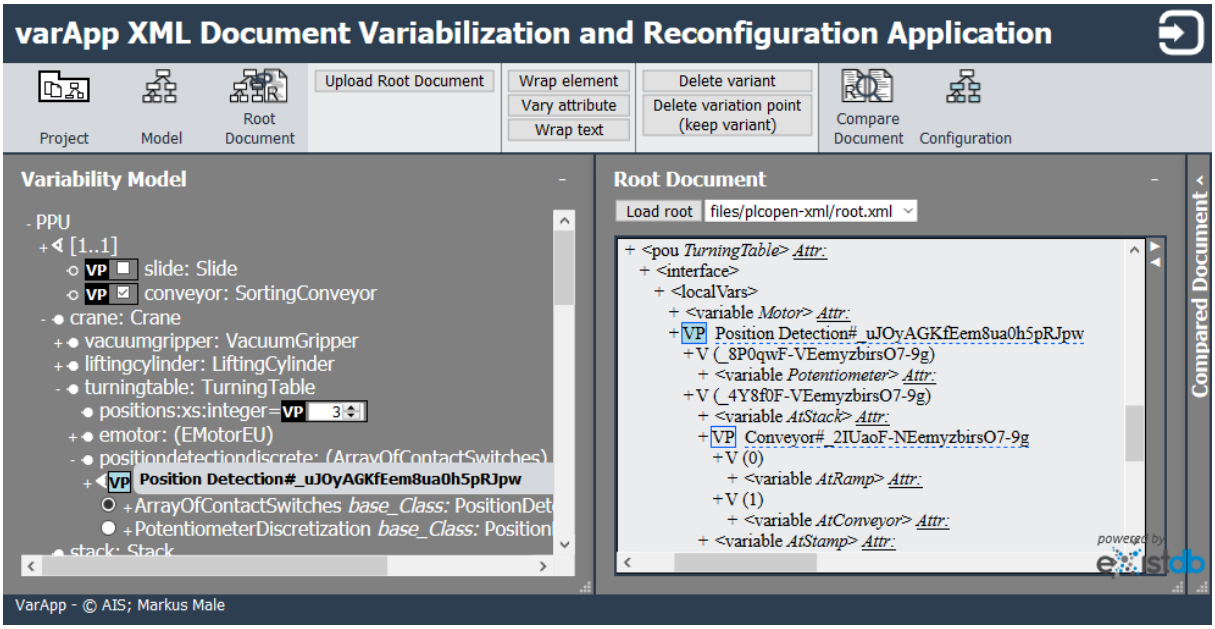

Fig. 6 VarApp frontend with loaded PPU example: Variability Model view (PLM) and Root Document view (product line DSM (enriched PLCopen XML))

example (cf. section V. A). It is assumed that a *VarML* compliant PLM and existing PPU legacy software variants are a sufficient base to build a product line DSM, which supports the derivation of software similar to existing configurations (variants) as well as new ones. Thereby, the re-configured, existing configurations (variants) should resemble the original software unchanged, while new configurations should be syntactically and semantically correct.

Apart from the afore-mentioned PPU variation point (crane position detection by potentiometer or micro switches; VP4), additional variation points could be identified in existing legacy software. VP1 describes the presence or absence of a stamp, VP2 whether the processed WPs are stored in a slide or further sorted via a sorting conveyor (cardinality VPs with group constraint), and VP3 expresses the number of pick up positions for the crane (value VP, dependent on VP1). Table III shows the existing legacy software configurations used to create the product line DSM and the derived configurations to evaluate the concept.

C. *Execution Steps and Requirement Fulfillment*

The PLM of the PPU has been modeled in Eclipse Papyrus compliant to the *VarML* profile, including the variation points and variants of each existing configuration as well as constraints. An excerpt from the model is presented in Fig. 1, right. Subsequently, the model (or respectively its XMI representation) has been transformed and loaded to the database via the VarApp frontend. In VarApp, it is presented in a feature-model like way (cf. Fig. 6, left side), thus fulfilling RQ2. The model is centrally stored in the backend and accessible from multiple clients as required by RQ1.

TABLE III. OVERVIEW ON VARIATION POINTS AND CONFIGURATIONS

Configuration	VP1	VP2	VP3	VP4
A	no stamp	slide	2 pos.	switches
B	stamp	conveyor	3 pos.	switches
C	stamp	conveyor	3 pos.	potentiometer
C*	stamp	conveyor	3 pos.	potentiometer
D*	stamp	slide	3 pos.	switches

* configurations derived from product line DSM with *VarApp*

Then, the legacy software of configuration B has been chosen to form the basis for the product line DSM (the root document, cf. Fig. 6., right). It is loaded to the backend via the VarApp frontend. Afterward, the varying parts of the document may be outlined via the frontend menu bar (Fig. 6, top). Accordingly, a reference and valueSelect element are created in the backend and shown in the frontend (cf. Fig. 6, right). Subsequently, other DSMs (Table II, configurations A and C) have been loaded and compared to the variability-enriched root document. Thereby, different variants have been included via the VarApp menu *Compare Document*. Selecting the element, attribute, or text in the compared document and the reference node in the root document, a new valueSelect node is generated in the backend. Fig. 6 shows, amongst others, the variation point *Position Detection* (marked in PLM on the left) and a corresponding reference node with two valueSelects (blue marked in the root document on the right). In this manner, all variants of existing legacy code are reusable, thus satisfying RQ6.

With the PLM and the product line DSM stored in the database, a new configuration can be chosen in the PLM by selecting an option for each variation point. User functions allow to validate the configuration against the constraints in the model and to derive a configured product DSM (RQ 4). In the example, this has been done for Configurations C* and D*. Using text comparison of the derived document C* and the original C did not detect any significant differences. Furthermore, the novel configuration D* has been validated by importing it to the original PLC development environment TwinCAT without errors. Hence, RQ5 is satisfied too, as a generic XSLT transformation enables the automated derivation of a product DSM, which can be downloaded and stored in the original DSM format (e.g., PLCOpen XML). As the concept, as well as the prototypical implementation, use exclusively open formats and tools, RQ 3 is fulfilled.

VII. Summary and Outlook

The presented concept enables SPL-based documentation and a tool-supported configuration of control software. The resulting database functions as a central, global knowledge base on existing variants and can be enlarged with additional software variants continuously. The configured variants are derived from legacy software conform to the standardized exchange format PLCopen XML. Thus, the concept supports the planned reuse of legacy software. The evaluation with VarApp confirmed the requirements are fulfilled.

Despite all achieved advantages, the manual comparison of two variant documents, though facilitated by a document tree view and side-by-side presentation, requires good knowledge of the document types and their composition as well as their variabilities. Thus, the addition of a new variant to the database requires a high initial, manual workload, which domain experts must perform. To reduce the initial effort, the presented concept could be combined with concepts that derive SPLs from legacy PLC software automatically, as described in [5]. Furthermore, although the XML-format is, in principle, human-readable, means to display the contained information more comprehensively are required. The same holds for the tree-like SPL visualization, which should be supplemented with a more sophisticated user interface and visualization strategies to represent the implementation details of multiple, industrial-sized control software variants simultaneously. So far, the concept has been evaluated with demonstrator software only; thus, to rate its scalability, industrial software should be considered in future work. Regarding the concept functionality, feature enlargements are expected after application to industrial documents.

With open formats and technologies such as XML for DSMs, the concept enables discipline-independent management of development documents in the database. The particular focus of this work was on IEC 61131-3 compliant control software as a special type of DSM. Further work includes the concept's application to other document types, including those from other disciplines, like development documents from mechanical or electrical engineering.

Acknowledgment

This work was supported by the DFG (German Research Foundation) (VO 937/31-1 and SCHA 1635/12-1).

References

[1] B. Vogel-Heuser *et al.*, "Towards interdisciplinary variability modeling for automated production systems: Opportunities and challenges when applying delta modeling: A case study," in *IEEE 13th International Conference on Industrial Informatics (INDIN)*, Cambridge, United Kingdom, 2015, pp. 322–328.

[2] J. Fischer, S. Bougouffa, A. Schlie, I. Schaefer, and B. Vogel-Heuser, "A Qualitative Study of Variability Management of Control Software for Industrial Automation Systems," in *IEEE International Conference on Software Maintenance and Evolution*, Madrid, 2018, pp. 615–624.

[3] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants," in *IEEE International Conference on Software Maintenance and Evolution*, Victoria, BC, 2014, pp. 391–400.

[4] K. Pohl, G. Böckle, and F. van der Linden, *Software product line engineering: foundations, principles and techniques*: Springer Science & Business Media, 2005.

[5] A. Schlie, K. Rosiak, O. Urbaniak, I. Schaefer, and B. Vogel-Heuser, "Analyzing variability in automation software with the variability analysis toolkit," in *23rd International Systems and Software Product Line Conference (SPLC)*, Paris, France, 2019, pp. 1–8.

[6] T. Berger *et al.*, "A survey of variability modeling in industrial practice," in *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, Pisa, Italy, 2013, pp. 1–8.

[7] OMG, *Unified Modeling Language (UML), Version 2.5*.

[8] *Programmable controllers – Part 10: PLC open XML exchange format*, International Electrotechnical Comission (IEC) 61131-10, 2019.

[9] D. Lettner, K. Eder, P. Grunbacher, and H. Prahofer, "Feature modeling of two large-scale industrial software systems: Experiences and lessons learned," in *ACM/IEEE 18th Int. Conf. on Model Driven Engineering Languages and Systems*, Ottawa, Canada, 2015, pp. 386–395.

[10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[11] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Softw. Process: Improve. Pract.*, vol. 10, no. 1, pp. 7–29, 2005.

[12] R. Behjati, T. Yue, L. Briand, and B. Selic, "SimPL: A product-line modeling methodology for families of integrated control systems," *Information and Software Technology*, vol. 55, no. 3, pp. 607–629, 2013.

[13] M. Broy and K. Stølen, *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. New York, NY: Springer, 2001.

[14] S. Biffl, E. Maetzler, M. Wimmer, A. Lueder, and N. Schmidt, "Linking and versioning support for AutomationML: A model-driven engineering perspective," in *IEEE 13th International Conference on Industrial Informatics (INDIN)*, Cambridge, United Kingdom, 2015, pp. 499–506.

[15] *Engineering data exchange format for use in industrial automation systems engineering - AutomationML*, IEC 62714, 2014.

[16] M. Wimmer *et al.*, "Cardinality-based variability modeling with AutomationML," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation*, Limassol, 2017.

[17] S. Feldmann and B. Vogel-Heuser, "Interdisciplinary product lines to support the engineering in the machine manufacturing domain," *International Journal of Production Research*, vol. 55, no. 13, pp. 3701–3714, 2017.

[18] M. Fang, G. Leyh, J. Doerr, C. Elsner, and J. Zhao, "Towards model-based derivation of systems in the industrial automation domain," in *Proceedings of the 19th International Conference on Software Product Line*, Nashville, Tennessee, 2015, pp. 283–292.

[19] D. Hinterreiter *et al.*, "Feature-Oriented Evolution of Automation Software Systems in Industrial Software Ecosystems," in *IEEE 23rd International Conference on Emerging Technologies & Factory Automation (ETFA)*, Turin, 2018, pp. 107–114.

[20] S. Schröck, A. Fay, and T. Jäger, "Systematic interdisciplinary reuse within the engineering of automated plants," in *9th Annual IEEE International Systems Conference (SysCon)*, Vancouver, BC, Canada, 2015, pp. 508–515.

[21] B. Vogel-Heuser, C. Legat, J. Folmer, and S. Feldmann, "Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the Pick and Place Unit.," Technical Report No. TUM-AIS-TR-01-14-02, 2014. [Online] Available: https://mediatum.ub.tum.de/node?id=1208973.