Selected Challenges of Software Evolution for Automated Production Systems

B. Vogel-Heuser, S. Feldmann, J. Folmer
Institute of Automation and Information Systems,
Technische Universität München, Germany
{ vogel-heuser ; feldmann ; folmer }@ais.mw.tum.de

M. Kowal, I. Schaefer
Institute of Software Engineering and Automotive Informatics, University of Braunschweig, Germany
m.kowal@tu-bs.de ; i.schaefer@tu-braunschweig.de

J. Ladiges, A. Fay
Automation Technology Institute,
Helmut Schmidt University, Germany
{ ladiges ; fay }@hsu-hh.de

C. Haubeck, W. Lamersdorf
Distributed Systems and Information Systems,
Universität Hamburg, Germany
{ haubeck ; lamersd }@informatik.uni-hamburg.de

S. Lity
Institute for Programming and Reactive Systems,
Technische Universität Braunschweig, Germany
lity@ips.cs.tu-bs.de

T. Kehrer
Software Engineering Group,
University of Siegen, Germany
kehrer@informatik.uni-siegen.de

M. Tichy
Software Engineering Division, Chalmers, Sweden |
University of Gothenburg, Sweden
matthias.tichy@cse.gu.se

S. Getir
Reliable Software Systems,
University of Stuttgart, Germany
sinem.getir@informatik.uni-stuttgart.de

M. Ulbrich, V. Klebanov, B. Beckert
Application-oriented Formal Verification, Karlsruhe Institute of Technology, Germany
{ ulbrich ; klebanov ; beckert}@kit.edu

***Abstract*—Automated machines and plants are operated for some decades and undergo an everlasting evolution during this time. In this paper, we present three related open evolution challenges focusing on software evolution in the domain of automated production systems, i.e. evolution and co-evolution of (interdisciplinary) engineering models and code, quality assurance as well as variant and version management during evolution.**

Keywords—automated production systems, evolution, variability, production automation, software engineerring

I. INTRODUCTION

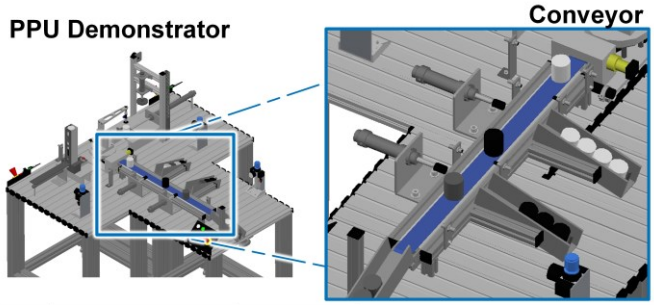
Automated machines and plants are often in operation for up to three decades [1]. As a consequence, such automated production systems (aPS) are subject to changes in customer requirements such as additional types of products to be produced as well as novel technological developments. Besides these changes that can be anticipated beforehand and that are typically addressed following a well-defined engineering process, e.g., the V-Model XT, also unanticipated changes according to Buckley's taxonomy [2] occur, e.g., during operation. Anticipated changes "can be foreseen during the initial development of the systems and, as such, can be accommodated in the design decision taken" [2] whereas unanticipated changes according to [2] are not foreseen during the development phase, but frequently undertaken at short notice, e.g., during commissioning and operation. In addition to these incremental *versions* that arise due to sequential evolution of the aPS, parallel evolution results in different *variants* of the system [3]. To stay competitive in global markets, companies that operate aPS are increasingly forced to address these challenges. Therefore, approaches are necessary to incorporate the (co-)evolution of aPS (*challenge I*), to ensure the quality of these systems (*challenge II*) and to manage their variability (*challenge III*).

To address challenges regarding evolution of such long-living systems from a software perspective, various approaches already exist. Although these approaches support different aspects of variant-rich software-intensive systems, e.g., verification and validation, consistency management and modeling, such approaches focusing merely on the software aspect are often not sufficient to address the challenges in the aPS domain focusing on machines henceforth. Therefore, the goal of this research is three-fold: (1) to determine why this is the case (section III), (2) to show the research goals and approaches to address the challenges in the aPS (section IV) and

(3) to identify how these approaches can lead to synergetic research goals and results (section V) focusing on the evolution of long-living aPS. The contributions presented in this paper are developed within the German Priority Programme SPP 1593 "Design For Future – Managed Software Evolution". We derive the challenges and illustrate our findings at a joined case study machine (section II), which was developed for the purposes of exploring and investigating the applicability of research results [4] and, by that, to facilitate the comparability of different approaches, ease exchange of research ideas, and enable research collaboration among different research groups and disciplines. We conclude with a summary in section VI.

II. An Open Demonstrator for the Machine Manufacturing Domain

To explore and investigate the applicability of fundamental research results for the challenges in the aPS domain, the Pick and Place Unit (PPU) serves as an open case study [4]. Although being a bench-scale, academic demonstration case, the PPU is complex enough to demonstrate selected challenges that arise during engineering of aPS. Especially to explore and investigate evolution of aPS, sixteen scenarios (that is: variants of the PPU) are defined [3], including both sequential evolution (e.g., to enlarge the PPU's functionality and performance) and parallel evolution (e.g., same functionality on different types of platforms, i.e. PLCs). An excerpt of these scenarios is illustrated in Fig. 1. Besides pure software changes that do not incorporate adaptations in the mechanics and automation hardware of the PPU, also changes to these disciplines are considered in the scenarios (e.g., increasing the work piece throughput in Sc12a up to replacing the control platform in Sc12d [3]).



Sce-nario	Cause triggering requirement	Conveyor			Remarks
		M	AH	S	
Sc12a	Higher throughput of workpieces	x	0	x	Faster pushers: increased dynamics of pneumatics ↔ different time constraints
Sc12c	Different control voltage	x	x	0	Different field bus components (e.g., I/O modules) required
Sc12d	Different platform supplier	0	x	(x)	Difference in automation components, i.e. drives, fieldbus and I/O modules
Sc12f	Additional functionality self-healing / diagnosis	x	0	x	Additional sensors and software required, automatic mode enlarged

M: Mechanics, **AH**: Automation Hardware, **S**: Software

Fig. 1. Excerpt of the PPU's scenarios for parallel evolution [3]

For the scenarios developed for the PPU, both the structure and the behavior are documented using Systems Modeling Language (SysML) [4]. The documentation is available online – as Eclipse Papyrus[1] models and as textual documentations [4]. Besides, automation software code for Programmable Logic Controllers (PLC) is available for each scenario – implemented in CODESYS[2], which is an industrial development tool for PLC control software. For testing purposes, MATLAB/Simulink models and 3D visualizations based on Virtual Reality Modeling Language (VRML) were developed. By that, PLC code can be tested against a simulation of the respective PPU scenario. The PPU's documentation is constantly being developed, improved and provided within the PPU community[3].

The PPU with its 22 digital input, 13 digital output, 3 analog output signals provides simple discrete event automation tasks [5], which have already been used for several controlled usability experiments with both technicians and students applying UML, SysML and IEC 61131-3 [5].

III. Selected Challenges for Evolving aPS

Engineering and operation of aPS involves changes. Therefore, besides the system itself, the models created during aPS' engineering are subject to (co-)evolution (*challenge I*, section III.A). In addition, the quality of aPS must be ensured after a change (*challenge II*, section III.B). Finally, besides sequential evolution of an aPS, parallel evolution results in different variants of the aPS that require approaches to manage aPS' variability (*challenge III*, section III.C). These challenges are highlighted in the following (cp. Fig. 2, top). A holistic overview on challenges that arise during software engineering and operation for evolving aPS as well as approaches that address these challenges can be found in [6].

A. Challenge I: Evolving and Co-Evolving (Interdisciplinary) Engineering Models

The engineering process in the aPS domain involves stakeholders from different disciplines, e.g. mechanical, electrical/electronic and software engineering. As a result, a variety of models that represent different views on the machine under consideration is created. Obviously, these models depend on each other because they overlap at certain points (e.g., sensors of the PPU can have representations as *hardware addresses* in an electrical view on the system and as *variables* in a software view on the system), and in case of evolution, they have to co-evolve consistently. Synchronizing these changes adequately and identifying the impacts of changes (e.g., if a hardware address is changed in the electrical view and must be synchronized with the respective control software) requires approaches for managing co-evolution of the different views.

Additionally, co-evolution of these different models is hampered by the infrequency of changes in the different disciplines involved: whereas changes are performed rarely to the mechanical subsystem (e.g., every 20 to 50 years), there are more frequent changes being performed to the electrical/electronic (every 10 to 15 years) and software subsystem (1.5 years). The development of the mechanical subsystem and automation hardware (electrics/electronics) therefore faces the rapid possibilities to change the respective control software. Besides, models of a machine exist to specify functional and non-functional aspects, e.g. regarding

[1] http://eclipse.org/papyrus/, retrieved on 5/4/2015.

[2] http://www.codesys.com, retrieved on 5/4/2015.

[3] http://www.ppu-demonstrator.org, retrieved on 5/4/2015.

dependability and fault handling in scenario Sc12f of the PPU demonstrator [3]. Therefore, to ensure that the design of an interdisciplinary aPS is correct, the consistent co-evolution of these engineering models has to be ensured.

Whereas such anticipated changes typically follow a well-defined procedure, also unanticipated changes exist [3]. Reasons for such unanticipated changes in the automation software of a machine are, e.g., the compensation of unexpected raw material during the commissioning phase of the aPS. Often, these changes are not well documented resulting in an insufficient co-evolution of machine and model [7]. In some cases, updates to the control software are even necessary while the machine is running, which requires appropriate techniques to verify such on-the-fly modifications in aPS [8]. Furthermore, ad-hoc adaptations are necessary, e.g., though self-organization in case of changing product requirements [9] or through self-healing in case of failures of system parts [3]. Additionally, due to high time and cost pressure, requirements elicitation and specification seldom follows an entirely model-based procedure, resulting in informal requirements specifications [10]. This is especially true for minor evolutionary changes in the software and physical subsystem of aPS [11] and underlines the need for consistently co-evolving the (formal) models with the realization of the aPS.

B. Challenge II: Quality Assurance for Evolving aPS

In order to ensure a correct and reliable design that fulfils the requirements on functional safety and availability of an aPS, methods to assure the quality of evolving aPS are inevitable. However, the interdisciplinary nature of an aPS implies that changes to one discipline (e.g., changing a sensor's hardware address in the electrical/electronic subsystem) affect other disciplines (e.g., the respective software view of the PPU) but also effects within one discipline arise. Quality assurance approaches therefore need to incorporate these dependencies – e.g., by assuring that the system's behavior is compliant to the intended behavior as specified in the requirements on the aPS.

However, shortened evolution cycles in the machine manufacturing domain imply a multitude of different variants and versions of an aPS. For instance, for the PPU demonstrator, measuring the crane module's angular position can be realized through respective micro switches mounted to the bottom of the crane or through a potentiometer [4]. Certainly, ensuring the quality of all variants and versions of the aPS is time- and cost-intensive. Thus, assuring quality for highly variant-rich aPS is up to now an open challenge. Furthermore, aPS are special systems with regard to their error-proneness for events that occur seldom (e.g., after long operation times or under specific environmental conditions). Therefore, approaches are necessary to ensure the quality of the system and its models during evolution and to verify that such critical events do not occur.

C. Challenge III: Variant and Version Management

One means to increase efficiency during engineering of aPS is the increase of modularity and reuse of already existing components [12]. Although approaches and investigations in academia and industry already exist to address such modularization issues, e.g. [12, 13, 14], the realization of module libraries that ultimately increase reuse during engineering of aPS are not yet state of the art [14]. aPS designs are hence often created from scratch or by adapting existing solutions as reuse opportunities seem to be expensive and inefficient at first sight. Amongst others, reasons for that are missing support to find the appropriate module from the library and identifying the correct level of granularity of available solutions [12].

Furthermore, as customer and system requirements are changing rapidly and continuously, aPS are highly variant-rich. Therefore, there exists a variety of simultaneously created module variants (e.g., scenarios Sc12a to Sc12f of the PPU demonstrator [4]) as well as versions of these variants, which are created over time [3]. Mechanisms to support the management of variants and versions are therefore inevitable. Variability can result from different sources: one is that customers can have different requirements on the machine. These requirements may lead to features to be implemented from a developer's point of view. For instance, a customer requires a specific type of controller for the PPU, which leads implications on the control software. Another source for variability is that the demanded features can be implemented by different solutions. For instance, the requirement to measure the current angular position of the crane module (customer's point of view) can be implemented through, e.g., micro switches or a potentiometer (developer's point of view). Therefore, both the variability from the customer's and from the developer's point of view need to be defined. It is hence inevitable to provide novel approaches to manage module variants and versions and, finally, to support reuse of such variants and versions for aPS designs.

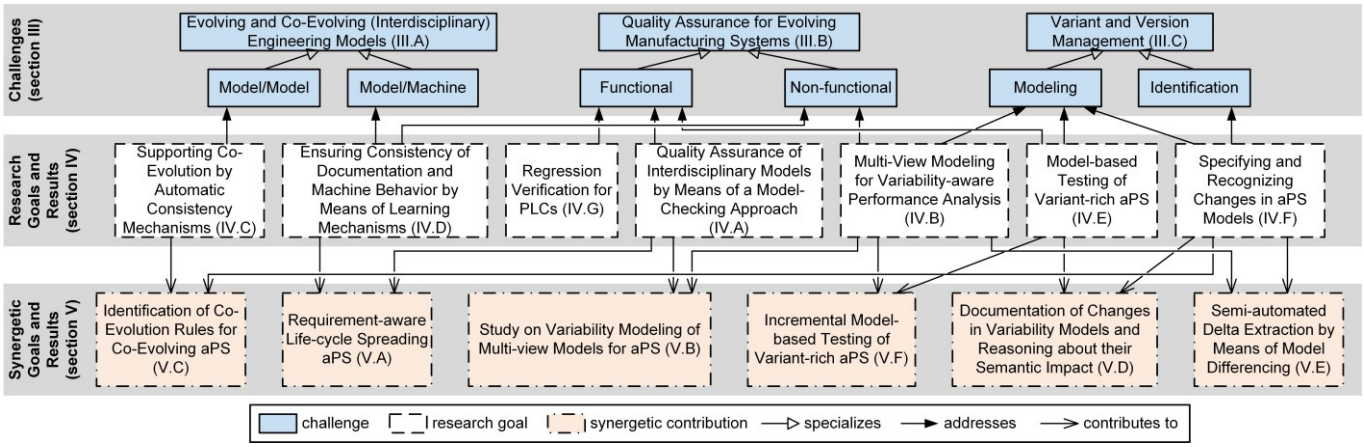


Fig. 2. Dependencies and relations between challenges (section III), research goals (section IV) and synergetic contributions (section V)

IV. Research Goals and Results

In this section, we present a brief overview on research goals that can be derived from the challenges that have been illustrated in section III. An overview on these research goals and their relations to the challenges are shown in Fig. 2.

A. Reasearch Goal A: Quality Assurance of Interdisciplinary Models by Means of a Model-checking Approach

To simplify the engineering of aPS, model-driven engineering is increasingly adopted in machine manufacturing industry [15]. Nevertheless, it is indispensable to ensure the correct functionality of the machine under consideration (cf. *challenge II*). When focusing on functional aspects of an evolving machine, mechanical aspects are typically of major interest and questions such as "Are work pieces correctly handled by the PPU?" arise. For instance, for handover positions at the PPU, it must be ensured that work pieces are handled correctly. To answer such questions and, by that, to ensure the correct functionality of the aPS, the overall system architecture and behavior of sensors/actuators, bus, the automation hardware (especially PLCs), and the control software have to be taken into account [16]. Various model-checking approaches exist for specific components, e.g. for verifying real-time behavior of the field bus behavior [17] or the control software [18, 19, 20], but an approach, which integrates the different disciplines does not exist until now. To address this challenge, a Model-Driven Evolution Management Framework for aPS software is currently being developed, which will support an interdisciplinary approach for automatically verifying functional requirements by means of model-checking [21] to assure quality during evolution. The framework provides different views for the involved stakeholders on a strict formal base: the *requirements viewpoint* supports the engineer in gaining a comprehensive understanding of the system under construction during requirement engineering. Refining the requirements viewpoint, the *process viewpoint* is used to model the technical process of an aPS on an abstract level. It describes a solution for the fulfillment of requirements with respect to the handling of material, e.g. work pieces for the PPU [22]. It enables to verify the technical process' correctness on an abstract level and to reason about the functionality of the aPS in an integrated fashion. The *system viewpoint* contains the model of an aPS' components, i.e. the components' structural description as well as their behavior by means of automata. This viewpoint facilitates the verification of functional requirements [21].

B. Research Goal B: Multi-view Modeling for Variability-aware Performance Analysis

Variability of aPS is increasing (*challenge III*) due to continuously evolving software and hardware over long periods of time. At the same time, quality of these different variants or versions needs to be assured (*challenge II*).

A two-fold approach is used to alleviate these challenges. First, a design-level modeling approach is developed consisting of three individual perspectives taking advantage of the principle of separation of concerns. A system is completely modeled within the *workflow*, *architecture* and *behavior* perspectives, which are based on Unified Modeling Language (UML) diagrams. The workflow represents the actual life cycle of a work piece throughout the system, e.g. which operations (such as transporting or stamping for the PPU's work pieces) are performed on it. The hardware elements of the machine are abstracted at the architecture as components and the behavior of the system components is captured by a state-based modeling formalism [23]. The creation of new variants or versions is often done by the well-known copy & paste approach, since no explicit variability management method is available. Duplicating all perspectives and conducting the variant-specific changes afterwards produces lots of redundancies, is ineffective and leads to an increased maintenance effort [24]. Second, in order to seamlessly handle variability, *delta modeling* is applied to each perspective. In delta modeling, the system has a core and sets of modifications (the *deltas*). Hence, a new variant/version is generated by applying the required deltas to the core [25]. The notion of delta modeling can be used on all three modeling viewpoints in a consistent fashion. By separation of concerns in the different perspectives, the analysis of non-functional system properties, such as performance requirements (*utilization, throughput, average queue length*), is enabled already in early development phases on the workflow level. This allows to assess the different variants and versions during the first representation of the intended manufacturing process before spending more effort in its detailed technical realization in the architecture and behavior perspectives [26].

C. Research Goal C: Supporting Co-Evolution by Automatic Consistency Mechanisms

During the entire lifecycle of an aPS, different aspects of the aPS need to be kept consistent (*challenge I*) so that the machine can correctly fulfill its requirements. Usually, different aspects are the three disciplines of mechanical hardware, electrical hardware, and software. However, also other development artifacts need to be kept consistent. Specifically, quality evaluation models are important in this regard as they allow for analyzing the quality of a machine (even before operation) with respect to properties such as throughput of work pieces in the PPU, safety as well as reliability of the PPU demonstrator.

Therefore, it is focused on supporting the co-evolution of these different aspects by using model-transformations and probabilistic analysis approaches, particularly, for probabilistic quality evaluation models like Fault Trees and Markov Chains. Based on the Henshin Model Transformation Language [27], the CoWolf[4] framework has been developed, which supports the consistent co-evolution of system architecture models, behavior model in form of state machines, and Fault Trees and as Petri Nets as modeling languages for quality evaluation models.

Complementary, a lightweight adaptive filtering technique has been developed to accurately learn time-varying transition probabilities of discrete time Markov models that provide robustness to noise and fast adaptation to changes with a very low overhead [28]. This allows for also keeping the quality evaluation models consistent with the running system by monitoring the running system and updating the quality evaluation models.

[4] http://cowolf.github.io/, retrieved on 5/4/2015.

D. Research Goal D: Ensuring Consistency of Documentation and Machine Behavior by Means of Learning Mechanisms

Despite the already existing model-driven engineering methods and tools supporting formalization of requirements, direct implementation of informal requirements often takes place in industrial practice [10]. Expert surveys have shown a lack of structured and systematic working during aPS development mainly due to high time pressure [29]. Especially changes performed during operation are insufficiently (formally) specified [6]. The result of such performed adaptations is a missing documentation (*challenge I*) and a lack of (quality) evaluation of changes performed to the aPS (*challenge II*).

Addressing these challenges, especially under consideration of shortened evolution cycles, requires a higher degree of self-awareness of the aPS [30]. One approach to realize self-awareness while reducing the influence of such countermeasures is to observe the aPS' behavior by means of the input-output signals of controlling PLCs (i.e., the observable behavior of the PPU) and, by that, to learn machine behavior models [11]. In terms of evolution, a proper execution of these models allows to recognize anomalies in the machine behavior by constantly comparing the actual behavior with the previously learned models. If an anomaly is detected, a semi-automated evaluation process could take place [31]. Such a process is able to analyze models regarding (non-functional) properties, which have been operationalized properly [32]. With these properties, an operator can determine with regard to the system requirement, whether the detected anomaly is part of an intended evolutionary change and the models should be updated by learning the evolved behavior. This ensures consistent documentation in steadily updated models as well as a constant quality evaluation with regard to requirements of undocumented performed changes during an evolution process.

E. Research Goal E: Model-based Testing of Variant-rich aPS

Quality assurance (*challenge II*) is an important and resource-intensive task during system development. Due to the increasing application of model-based engineering in the automation domain [15], also model-based testing [33] can be applied allowing for automatic test case generation based on a behavioral test model specification. In the context of variant-rich machines (*challenge III*) applying model-based testing for each variant in isolation is not feasible in general. Although some approaches for model-based testing of machines exist [34, 35, 36, 37], the application of model-based testing for variant-rich aPS is still an open challenge.

Based on the software product line (SPL) paradigm [38], a model-based testing approach for variant-rich machines is developed [39] to address the quality assurance of variant-rich systems. An SPL [38] represents a family of similar software systems with the specification of commonality and variability by means of customer-visible properties called features. A common core as well as reusable development artifacts build the basis for SPL engineering. Based on a 150% state chart test model, i.e., a model comprising all variant-specific test models, where elements are annotated with features to define for which variants an element is valid, a complete test suite generation for all variants is realized. For each generated test case, the set of variants is derived for which the test case is reusable by analyzing the feature annotations. This leads to an effort reduction, i.e., the test case generation time and the test suite size are reduced compared to the test suite generation for each variant in isolation, where (reusable) test cases are generated for each variant anew.

F. Research Goal F: Specifying and Recognizing Changes in aPS Models

Model-based engineering of machines implies that different types of models are systematically used as primary development artifacts, which are iteratively developed and which heavily evolve throughout the lifecycle of an aPS. Thus, a clear picture of the changes between versions of a model is one of the most essential preconditions to understand and plan the evolution of such a model-based system (*challenge III*). Thereby, exact and meaningful specifications of changes play two different roles; (1) a prescriptive role, i.e. as specification of modifications to be performed on an existing model version, and (2) a descriptive role as a means to describe the observed difference between two versions, notably past changes in the history of a model. Such specifications of changes are a basis for many further tasks, e.g. for propagating changes in variants of a model [40].

A variety of concepts, technologies and tools are involved in the specification and the analysis of model evolution; their development is another research goal. Firstly, we need sophisticated approaches from the domain of model transformation which enable the specification of high-level edit operations (also known as delta operations), e.g. as offered by visual editors or modern refactoring tools. Secondly, model comparison techniques are required that deliver model differences (or deltas) which are based on these high-level edit operations. In other words, an obtained difference specifies how the base version can be transformed to the revised one in a step-wise manner, each step refers to an invocation of a high-level edit operation.

Formalisms to define such operations and methods to recognize the resulting changes mutually depend on each other. First advances in this field, which are based on graph transformation concepts, can be found in [41, 42].

G. Research Goal G: Regression Verification for PLCs

In the safety-critical context of aPS, evolution of software must not introduce faults into the system behavior (*challenge II*). If a complete specification of the desired system behavior is not available, one possibility to ensure that the quality of the PLC software is preserved during an evolution step is to compare the new system version against the old version.

At this point, regression verification comes to the aid: using formal methods (based on state-of-the-art model checkers), regression verification statically analyzes the two software versions and verifies that the new version does not introduce unwanted behavior. This is done by conducting a formal proof that the new plant software behaves equivalently to the old one. Obviously, an evolution step usually also introduces desired changes (optimizations, corrections, adaptations) into the system behavior. It must hence be possible to specify precisely which parts of the behavior are to be retained.

By design, regression verification cannot make a statement about features newly introduced since there is no counterpart in the old version to compare against. Regression verification is thus a valuable approach that is complementary to other quality assurance methods during evolution without replacing them. For the parts where behavior is preserved, regression verification has advantages over functional verification and testing: No formal specification is needed to describe the desired safety property of the new system. The old version, as the reference implementation, serves this purpose. Moreover, no individual test cases are needed and the state space is fully explored.

Based on an approach for regression verification on C-code [43], a tool was implemented [44] that computes a model regression verification condition for two IEC 61131-3 PLC implementations. The tool has been applied successfully for the PPU case study revealing a few flaws in the defined evolution steps. One research challenge was the efficient encoding of the differences between PLC software versions, between which many things have not changed, into a formula that can serve as model checking input. The approach allows for conditional verification where equivalence only needs to be shown for those parts of the state space satisfying a certain condition and for relational verification where it is not equality of system states but an equivalence relation specified by the user that is checked.

In many cases, considering the PLC software code as sole input is not sufficient since the equivalence of the system versions depends also on behavioral patterns of the aPS. In these cases, behavioral models of physical entities are needed. It remains an open research question how equivalence proofs found during regression verification can be incorporated into the holistic verification workflow as proposed in *research goal A*.

V. Synergetic Research Goals and Results

Based on the research goals and approaches described in section IV, several synergetic research goals and results towards managing the evolution of long-living aPS can be achieved by combining these approaches (cp. Fig. 2, bottom).

A. Synergetic Research Goal A: Requirement-aware, Life-cycle Spreading aPS

Due to market requests, aPS undergo an evolution process that increasingly shifts traditional development activities for aPS to later phases of their lifecycle, i.e. a machine is evolved in both ways, a model-driven (anticipated) way in the development phase as well as an unanticipated way during the operational phase. Accordingly, the co-evolution challenges between different models as well as towards the actual system arise (*challenge I*). Furthermore, (semi-automated) requirement verification mechanisms during evolution in various phases of a machine's life cycle are required to assure quality (*challenge II*).

An approach combining a posteriori and a priori verification and highlighting synergetic effects is presented in [45] towards requirement-aware, life-cycle spreading automation systems. An interdisciplinary modeling framework for model-driven engineering (*research goal A*) as well as a PLC-signal based monitoring technique (*research goal C*) are applied that enhance each other by comparing estimated and actual system characteristics as verifiable requirement description. The resulting, combined approach and its synergies are illustrated in [45] by means of two scenarios of the PPU. Firstly, the combined approach can be used for correcting estimations of model-driven engineering by signal monitoring of actual values. This facilitates to identify potentially dangerous situations which might result from misestimated values in MDE but can be identified by applying updated values for verifying functional aspects. Secondly, it was shown that the combined approach can be beneficial for proving intention and identifying change sources. To identify the source of the change precisely, sophisticated models are required which are not available for monitoring approaches on the signal level, which is able to detect anomalies precisely. A combined approach leverages the identification of change sources over different lifecycle phases.

B. Synergetic Research Goal B: Study on Variability Modeling of Multi-view Models for aPS

Both *research goals A* and *B* investigate different aspects of applying multi-view models. Understanding each other's approaches in detail facilitates novel, synergetic aspects: a collaborative study on variability modeling for multi-view models in the aPS domain by means of delta modeling [25] (*challenge III*). This synergetic research goal is driven by knowledge exchange between different involved domain experts: variability modeling experts and automation engineers.

In delta modeling, system variants and versions are represented by explicit sets of modifications (the deltas). In order to obtain the software for a particular version or variant of an automation system, the necessary modifications captured in the deltas are automatically applied to a designated core system model. Until now, two case studies have been undertaken: Firstly, the multi-view model of *research goal A* was applied to the PPU [23]. It was shown that the multi-view model can be used to model different variants of the PPU for generating PLC state charts. Secondly, delta modeling is applied to the interdisciplinary model [23] which was developed under *research goal A*. The result is two-fold: (1) applying delta modeling to such formal, interdisciplinary models is possible and (2) for applying delta modeling, an eased way for defining deltas in such models is necessary (cf. section E).

C. Synergetic Research Goal C: Identification of Co-Evolution Rules for Co-Evolving aPS Models

Iterative development and changing requirements lead to continuously changing models. In particular, this leads to the problem of consistently co-evolving (*challenge I*) different views of a model-based system. Whenever one model undergoes changes, related models should evolve with respect to this change. Domain engineers are faced with the huge challenge to find proper co-evolution rules which can be finally used to assist developers in the co-evolution process. By integrating aspects from *research goals C* and *F*, an approach to learn about co-evolution steps from a given history using an extensive analysis framework [46] can be achieved. The approach uses models and model transformations as described in *research goal C*, the SiLift [47] framework as a result from the work described in *research goal F* and the jointly developed co-evolution framework. The approach is generic as it can be adapted to study co-evolution of other types of models. For this purpose, the framework needs the available model transformation rules for each of the involved types of models as input.

D. Synergetic Research Goal D:Documentation of Changes in Variability Models and Reasoning about their Semantic Impact

By integrating *research goals E* and *F*, an automated approach to (1) document the evolution of feature models and (2) semantically reason about changes between feature models by using complex edit operations in order to describe the structural changes between two versions of a feature diagram can be achieved. Therefore, a set of typical edit operations on feature diagrams using the model transformation language Henshin was specified. Henshin is based on graph transformation concepts and allows to precisely specify edit operations as so-called edit rules. These edit rules are used by the model differencing engine SiLift [47] to get a meaningful description of feature diagram differences. Furthermore, a logic-based formal framework was developed which allows to reason about the semantic impact of feature diagram changes.

In particular, complex edit operations and differences which are described in terms of these edit operations were categorized using the categories proposed in [48] to reason about the semantic impact of syntactic changes, i.e., classifying them as refactoring, specialization, generalization, or arbitrary edit. Later, this approach will be used in the field of regression testing to indicate how to change a test suite after changing the SPL.

E. Synergetic Research Goal E: Semi-automated Delta Extraction by Means of Model Differencing

A delta is a specification of how to transform one valid variant (called the source variant) of a model being written in some modeling language into another variant (called target variant). The manual specification of a large set of deltas is tedious and prone to errors. To that end, *research goal B and research goal F* can be combined to a synergetic contribution to semi-automated delta extraction. The basic idea is that valid variants of the SPL are provided by the results from *research goal B* and deltas are defined by using techniques from the domain of model differencing (*research goal F*). With this approach, the definition of a delta is achieved in two steps: First, a particular source variant is modified using a standard editor for that language (e.g. *workflow*, *architecture* and *behavior* perspectives as introduced in section IV.B) such that it finally becomes the desired target variant. Secondly, a delta is automatically extracted by comparing the original with the revised version of the model. The result of the comparison is transformed into an executable delta. Thus, the definition of a delta becomes much easier and far more reliable.

F. Synergetic Research Goal F: Incremental Model-based Testing of Variant-rich aPS

aPS evolve continuously, e.g., based on software or hardware updates. To guarantee quality assurance after evolution (*challenge II*), testing strategies are required focusing on changes and their impacts as, e.g., regression testing. In the context of variant-rich systems (*challenge III*), regression testing further allows for an incremental testing process by exploiting the reuse potentials when stepping from one variant to another.

Lochau et al. [49] present an approach for incremental testing of variant-rich systems by combining model-based and regression testing on the component as well as integration testing level. Delta modeling [25] (*research goal B*) is used to specify changes between component state chart models and architectural models by means of regression model deltas. Those regression deltas build the basis for the derivation of changes also for the remaining test artifacts, i.e., the set of test goals, the test suite and a test plan captured in a test artifact regression delta. Based on a complete test suite comprising reusable test cases for each variant (*research goal E*), the test artifact regression deltas allows for an incremental test artifact application when stepping from one variant to a subsequent one. By exploiting the reuse potentials of test artifacts and test results, the incremental testing process leads to a reduction of the testing effort and builds the basis for quality assurance after evolution. Lity et al. [50] propose an approach for incremental model slicing based on the concepts of delta modeling applicable as test case selection technique for regression testing of variant-rich systems.

VI. Conclusion and Outlook

As aPS are often in operation for up to three decades, they evolve during their lifecycle. In this paper, we illustrated some challenges and selected research goals regarding evolving aPS, showed exemplary approaches to address these challenges and identified how these research goals can lead to synergetic contributions in the field of software evolution for aPS.

First results regarding managing the (co-)evolution of long-living aPS could be achieved – e.g., in the fields of consistency mechanisms, verification, validation and testing as well as variability modeling – and transferred and evaluated at a simple lab size aPS. The Pick and Place Unit [4] as an open case study served as a valuable demonstration case for the approaches. In between sixteen options for the adaptation of the PPU case study for future research, online model and program changes, safety aspects, deployment, i.e. distributed automation systems, and a more challenging technical process will be prioritized. Online changes will address the evolution in the operation phase of the aPS as it is well known on classical PLC platforms.

Acknowledgements

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP 1593: Design For Future – Managed Software Evolution (http://www.dfg-spp1593.de/).

References

[1] ZVEI, "Life-Cycle-Management für Produkte und Systeme in der Automation," Online: http://www.zvei.org/Publikationen/Leitfaden_LifeCycle.pdf.

[2] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *J. Softw. Maintenance Evolution: Research and Practice*, vol. 7, no. 5, pp. 309–332, 2005.

[3] B. Vogel-Heuser, C. Legat, J. Folmer, and S. Rösch, "Challenges of parallel evolution in production automation focusing on requirements specification and fault handling," *Automatisierungstechnik*, vol. 62, no. 11, pp. 755–826, 2014.

[4] 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," TUM-AIS-TR-01-14-02, Tech. Rep., 2014.

[5] B. Vogel-Heuser, "Usability experiments to evaluate UML/SysML-based model driven software engineering notations for logic control in manufacturing automation," *J. Softw. Eng. Appl.*, vol. 7, no. 11, pp. 943–973, 2014.

[6] B. Vogel-Heuser, C. Diedrich, A. Fay, S. Jeschke, S. Kowalewski, M. Wollschlaeger, and P.Göhner, "Challenges for software engineering in automation," *J. Softw. Eng. Appl.*, vol. 7, no. 5, pp. 440–451, 2014.

[7] C. Haubeck, I. Wior, L. Braubach, A. Pokahr, J. Ladiges, A. Fay, and W. Lamersdorf, "Keeping pace with changes – towards supporting continuous improvements and extensive updates in production automation software," *Electron. Commun. EASST*, vol. 56, 2013.

[8] C. Sünder, V. Vyatkin, and A. Zoitl, "Formal verification of downtimeless system evolution in embedded automation controllers," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1, pp. 17:1–17:17, 2013.

[9] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux, "Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution," *Computers in Industry*, vol. 66, pp. 99–111, 2015.

[10] G. Frey and L. Litz, "Formal methods in PLC programming," in *IEEE Int. Conf. Syst. Man Cybern.*, vol. 4, 2000.

[11] J. Ladiges, C. Haubeck, A. Fay, and W. Lamersdorf, "Evolution management of production facilities by semi-automated requirement verification," *Automatisierungstechnik*, vol. 62, no. 11, 2014.

[12] N. Jazdi, C. Maga, and P. Göhner, "Reusable models in industrial automation: Experiences in defining appropriate levels of granularity," in *IFAC World Congr.*, 2011.

[13] K. Thramboulidis, "Overcoming mechatronic design challenges: The 3+1 SysML-view model," *J. Comput. Sci. Technol.*, pp. 6–14, 2013.

[14] S. Feldmann, J. Fuchs, and B. Vogel-Heuser, "Modularity, variant and version management in plant automation – future challenges and state of the art," in *Int. Design Conf.*, Dubrovnik, Croatia, 2012, pp. 1689–1689.

[15] V. Vyatkin, "Software engineering in factory and energy automation: State of the art review," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1234–1249, 2013.

[16] B. Vogel-Heuser, S. Feldmann, T. Werner, and C. Diedrich, "Modeling network architecture and time behavior of distributed control systems in industrial plant automation," in *IEEE Ann. Conf. Ind. Electron. Soc.*, 2011.

[17] D. Witsch, B. Vogel-Heuser, J. M. Faure, and G. Marsal, "Performance analysis of industrial ethernet networks by means of timed model-checking," in *IFAC Symp. Inf. Control Problems Manuf.*, 2006.

[18] S. Biallas, J. Brauer, and S. Kowalewski, "Arcade.PLC: A verification platform for programmable logic controllers," in *IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2012, pp. 338–341.

[19] T. Mertke and G. Frey, "Formal verification of PLC programs generated from signal interpreted petri nets," in *IEEE Int. Conf. Syst. Man Cybern.*, vol. 4, 2001, pp. 2700–2705.

[20] N. Bauer, S. Engell, R. Huuck, S. Lohmann, B. Lukoschus, M. Remelhe, and O. Stursberg, "Verification of PLC programs given as sequential function charts," ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3147, pp. 517–540.

[21] C. Legat, J. Mund, A. Campetelli, G. Hackenberg, J. Folmer, D. Schütz, M. Broy, and B. Vogel-Heuser, "Interface behavior modeling for automatic verification of industrial automation systems' functional conformance," *Automatisierungstechnik*, vol. 62, pp. 815–825, 2014.

[22] G. Hackenberg, A. Campetelli, C. Legat, J. Mund, S. Teufl, and B. Vogel-Heuser, "Formal technical process specification and verification for automated production systems," ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8769, pp. 287–303.

[23] M. Kowal, C. Legat, D. Lorefice, C. Prehofer, I. Schaefer, and B. Vogel-Heuser, "Delta modeling for variant-rich and evolving manufacturing systems," in *Int. Works. Modern Softw. Eng. Methods Ind. Autom.*, 2014.

[24] M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *J. Syst. Softw.*, vol. 1, pp. 213–221, 1980.

[25] I. Schaefer, "Variability modelling for model-driven development of software product lines," in *Int. Workshop Variability Modeling Softw.-intensive Syst.*, 2010.

[26] M. Kowal, I. Schaefer, and M. Tribastone, "Family-based performance analysis of variant-rich software systems," in *Int. Conf. Fundamental Approaches Softw. Eng.*, 2014.

[27] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: Advanced concepts and tools for in-place EMF model transformations," in *ACM/IEEE Int. Conf. Model Driven Eng. Languages Syst.*, 2010.

[28] A. Filieri, L. Grunske, and A. Leva, "Lightweight adaptive filtering for efficient learning and updating of probabilistic models," in *Int. Conf. Softw. Eng.*, 2015.

[29] M. Bellgran and E. K. Säfsten, *Production Development: Design and Operation of Production Systems*. London: Springer, 2010.

[30] C. Haubeck, W. Lamersdorf, J. Ladiges, and A. Fay, "An active service-component architecture to enable self-awareness of evolving production systems," in *IEEE Int. Conf. Emerg. Technol. Factory Autom.*, 2014.

[31] J. Ladiges, C. Haubeck, A. Fay, and W. Lamersdorf, "Semiautomated decision making support for undocumented evolutionary changes," in *Workshop Softw. Reeng. Evolution*, 2014.

[32] ——, "Operationalized definitions of non-functional requirements on automated production facilities to measure evolution effects with an automation system," in *IEEE Int. Conf. Emerg. Technol. Factory Autom.*, 2013.

[33] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2007.

[34] R. Hametner, B. Kormann, B. Vogel-Heuser, Winkler, and A. Zoitl, "Automated test case generation for industrial control applications," in *Recent Advances in Robotics and Automation*, ser. Studies in Computational Intelligence. Springer, 2013, vol. 480, pp. 263–273.

[35] T. Hussain and G. Frey, "UML-based development process for IEC 61499 with automatic test-case generation," in *IEEE Int. Conf. Emerg. Technol. Factory Autom.*, 2006.

[36] B. Kumar, B. Czybik, and J. Jasperneite, "Model-based TTCN-3 testing of industrial automation systems – first results," in *IEEE Int. Conf. Emerg. Technol. Factory Autom.*, 2011.

[37] S. Rösch, D. Tikhonov, D. Schütz, and B. Vogel-Heuser, "Model-based testing of PLC software: Test of plants' reliability by using fault injection on component level," in *IFAC World Congr.*, 2014.

[38] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[39] M. Lochau, J. Bürdek, S. Lity, M. Hagner, C. Legat, U. Goltz, and A. Schütt, "Applying model-based software product line testing approaches to the automation engineering domain," *Automatisierungstechnik*, vol. 62, no. 11, pp. 771–780, 2014.

[40] T. Kehrer, U. Kelter, and G. Taentzer, "Propagation of software model changes in the context of industrial plant automation," *Automatisierungstechnik*, vol. 62, no. 11, pp. 803–814, 2014.

[41] ——, "A rule-based approach to the semantic lifting of model differences in the context of model versioning," in *IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2011.

[42] ——, "Consistency-preserving edit scripts in model versioning," in *IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2013.

[43] D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, and M. Ulbrich, "Automating regression verification," in *IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2014.

[44] A. Weigl, "Regression verification for programmable logic control software," Master's thesis, Karlsruhe Institute of Technologie, 2015.

[45] C. Haubeck, J. Ladiges, J. Fuchs, C. Legat, W. Lammersdorf, A. Fay, and B. Vogel-Heuser, "Interaction of model-driven engineering and signal-based online monitoring of production systems," in *IEEE Ann. Conf. Ind. Electron. Soc.*, 2014.

[46] S. Getir, M. Rindt, and T. Kehrer, "A generic framework for analyzing model co-evolution," in *Int. Workshop Models Evolution*, 2014.

[47] T. Kehrer, U. Kelter, M. Ohrndort, and T. Sollbach, "Understanding model evolution through semantically lifting model differences with silift," in *IEEE Int. Conf. Softw. Maintenance*, 2012.

[48] T. Thüm, D. Batory, and C. Kästner, "Reasoning about edits to feature models," in *IEEE Int. Conf. Softw. Eng.*, 2009.

[49] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz, "Delta-oriented model-based integration testing of large-scale systems," *J. Syst. Softw.*, vol. 91, pp. 63–84, 2014.

[50] S. Lity, H. Baller, and I. Schaefer, "Towards incremental model slicing for delta-oriented software product lines," in *IEEE Int. Conf. Softw. Anal. Evolution Reeng.*, 2015.