Feature-Oriented Evolution of Automation Software Systems in Industrial Software Ecosystems

Daniel Hinterreiter
CD Lab MEVSS, ISSE
Johannes Kepler University
Linz, Austria
Email: daniel.hinterreiter@jku.at

Lukas Linsbauer
CD Lab MEVSS, ISSE
Johannes Kepler University
Linz, Austria
Email: lukas.linsbauer@jku.at

Florian Reisinger
CD Lab MEVSS, ISSE
Johannes Kepler University
Linz, Austria
Email: florian.reisinger@jku.at

Herbert Prähofer
Institute System Software
Johannes Kepler University
Linz, Austria
Email: herbert.praehofer@jku.at

Paul Grünbacher
CD Lab MEVSS, ISSE
Johannes Kepler University
Linz, Austria
Email: paul.gruenbacher@jku.at

Alexander Egyed
Software Systems Engineering (ISSE)
Johannes Kepler University
Linz, Austria
Email: alexander.egyed@jku.at

***Abstract*—In the domain of industrial automation many companies nowadays need to serve a mass market while at the same time customers demand individual customer-specific solutions. Such customizations often apply to individual products only but may also be needed at the level of product lines for whole market segments. To handle this problem, development is frequently organized in software ecosystems (SECOs), i.e., interrelated software product lines involving internal and external developers. This paper introduces an approach supporting feature-oriented, distributed development and evolution in industrial SECOs. It is common industrial practice to first derive initial products from a product line, then adding and adapting features to satisfy individual customer requirements, possibly followed by merging back these changes into the original product line. Our approach goes beyond this practice and also allows to share new or updated features by transferring them to other product lines in the ecosystem. This is for instance useful when a feature developed in an individual customer project becomes relevant for another market segment or when updates of features need to be transferred to related products in the ecosystem. We describe and motivate research challenges based on the industrial ecosystem of an industry partner. We outline the key elements and operations of our approach, including an implementation in our FORCE^2 development environment. We demonstrate application scenarios from the well-known Pick-and-Place Unit (PPU) system as a proof of concept.**

I. Introduction

The automation industry is facing an increasing market demand for machine-specific solutions [1], [2]. Therefore, industrial software solutions are nowadays often developed and evolved in software ecosystems (SECOs), i.e., interrelated software product lines [3] involving internal and external developers. An example of such a SECO is KePlast, a SECO in the domain of industrial automation maintained by our industry partner Keba AG. Our analysis of the evolution in this SECO [4] showed that development is based on a multi-stage [5] and clone-and-own [6]–[11] process involving multiple product lines: the core product line provides a comprehensive set of features to create a wide range of solutions for automating injection molding machines. It is controlled by an in-house development team. Individual products for customers are derived and adapted by adding new features or creating new versions of existing features to meet the customer-specific requirements. Furthermore, in the KePlast SECO the core product line is also the basis for deriving yet other product lines, which are then used by Keba and OEMs to develop products for specific market segments. That means, the development in the SECO is performed in a globally distributed fashion, only weakly integrated with the in-house team.

The KePlast example shows that managing the evolution in SECOs is challenging as developers continuously and independently evolve features of the core product line, the cloned product lines, and individual customer products. In particular, it remains hard to track and understand evolution at the level of features. Development teams typically use version control systems to track fine-grained, implementation-level changes to product lines and products. However, it is difficult to relate such low-level changes to features and their evolution in the SECO. Overall, this bears the risk that knowledge about commonalities and variability of existing product lines as well as products created by different business units or organizations is lost, thus increasing the risk of re-inventing the wheel in distributed development. This is confirmed by Duc et al.'s [12] investigation of multi-platform development practices, which showed that diverged code bases frequently lead to redundant development.

The aim of our ongoing research is thus to support the development and evolution in SECOs at the level of features. Lifting the focus to features is essential, as they are widely used by product management, software architects, and developers in SECOs to communicate about systems and changes to systems [13]–[15]. Features also provide a useful abstraction and common view of variability in a wide range of development artifacts [16], [17].

In our earlier research we introduced a multi-purpose, multi-level feature modeling environment [18], augmented with feature-to-code mappings [14] and configuration-aware program analysis [10]. We have now been evolving this environment to support distributed and feature-oriented development and evolution in SECOs. Our approach and its FORCE^2 tool implementation go beyond the scope of single integrated product lines as often assumed in conventional product line engineering approaches and aims to support engineering in a network of interrelated product lines.

The main contribution of this paper is a description of our approach with key elements and operations for supporting feature-oriented, distributed development in SECOs and its implementation in the FORCE^2 development environment. We also provide a preliminary evaluation of our concepts and tool based on application scenarios from the Pick-and-Place Unit (PPU) system [19].

II. Industrial Challenges

Our analysis of software evolution challenges in the industrial SECO of Keba AG [4] revealed four main research challenges for feature-oriented evolution of interrelated product lines:

C1) How can we create a product line for a specific development task from another product line? It has been shown that the development of products for specific customers or market segments require specialized product lines, which are created from an original product line by selecting, extending and adapting the required features. For example, in the KePlast SECO, a specific product line has been created for the Chinese market from the original product line. Such cloned product lines, however, then evolve independently and keeping them up-to-date becomes extremely difficult, e.g., when new of modified features of the original product line become available.

C2) How can we transfer features from one product line to another product line? It has been shown that 'transplanting' code and related artifacts from one system to another in a largely automated fashion is extremely challenging [20]. Even if features have been developed in one product line with the intention to reuse them in another product line, transferring them remains hard. As a result engineers often develop identical or similar features in different product lines, thus re-inventing the wheel as shown by Gousios et al. [21]. This increases the risk of losing the knowledge of feature implementations in the different product lines [22].

C3) How can we merge features back into the original product line? If a new feature or a new version of an existing feature developed for one specific purpose becomes relevant for other customers or market segments, it should be merged back into the original product line. This is difficult, as the customer-specific adaptations often prevent the straightforward reuse of the feature. Some approaches have been proposed to reduce this gap between clone-and-own and product lines. For instance, the VariantSync approach [23] aims at automating the synchronization of software variants to simplify their later integration in the product line. However, in [24] the authors show that new features (pull requests) are also often rejected due to problems with code quality, or if they become outdated due to even newer features. Overall, evolving a product line remains challenging, particularly, if the features in the product line are tightly tangled, dependent, or interacting with each other.

C4) How can we pull updated features from the original product line into cloned product lines or derived products? A fourth challenge is to update derived products and cloned product lines, which typically evolved independently from the original product line, by pulling updated features from the original product lines. Such updates usually result in inconsistencies and conflicts which cannot be resolved easily as pointed out by Schulze *et al.* [25].

III. Approach

To address these challenges we have been developing a platform supporting a feature-based clone-and-own approach for managing multiple distributed product lines in SECOs. The proposed approach follows the idea of a 'virtual platform strategy' [11] and provides support for managing multiple product lines and products in a SECO as shown in Fig. 1.

Specifically, a SECO comprises multiple interdependent *platforms*, each embodying a product line with variability. Each platform provides an independent working environment for feature-oriented development and includes feature models, feature-to-code mappings, a feature-oriented version control system, and one or more derived products. A new platform can be created based on an existing one by selecting a coherent subset of features and *cloning* the platform based on the selection, i.e., the clone will only contain that subset of features and code. Features can be *transferred* to other platforms, they can be *pushed* to their original platform, or they can be *updated* from their original platform.

A. Platform Elements

A platform for product line development comprises four main building blocks (cf. Fig. 2): a *feature model* defining the common and variable features of a product line; a

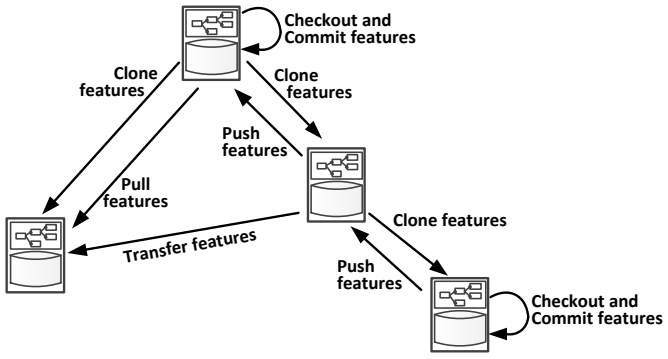

Fig. 1: Multiple platforms form an ecosystem. Various operations support feature-oriented evolution in the SECO.

feature-oriented code and artifact repository for managing the implementation artifacts of the product line, with *feature-to-code mappings* linking features to source code and other artifacts; as well as derived *products*, i.e., compositions of code and artifacts for specific feature configurations.

Feature models. We use a feature modeling approach with support for grouping features in components and arranging them in different modeling spaces as described in [14]. We further support multiple versions of features in feature models to support tracking evolution at the level of features.

Feature-oriented code artifact repository and feature-to-code mappings. In distinction to common source code repositories and version control solutions such as Git or Subversion, our approach is based on the variability-aware version control system ECCO [26]–[30], which was originally designed to recover feature-to-code mappings from independently maintained products [8]. ECCO stores and versions features and maintains mappings of features to artifacts in the repository. Mappings are maintained in the form of presence conditions determining whether the artifacts ought to be included if a specific feature is selected in a product configuration. A presence condition is a propositional logic formula with feature versions as literals. It supports the evolution of features over time by considering feature versions in the computed presence conditions [31].

Products. A product then is a concrete composition of artifacts from the repository. In fact, a product is created based on a valid selection of features from the feature model by composing the artifacts mapped to the selected features. Specifically, the feature-to-code mappings stored in the ECCO repository allow composing products from a selection of features in alignment with the feature model. The derived product is then the basis for extending and adapting the product solution and its corresponding feature model.

B. Internal Operations

For supporting SECO development and evolution, we distinguish *internal* operations only affecting one platform, and *externa*l operations involving different platforms in the SECO. The internal (intra-platform) operations *Checkout Features* and *Commit Features* allow to develop and evolve features within one platform (cf. Fig. 1 and 2).

Checkout Features. This operation allows deriving and composing a product from within a platform. Thus, a checkout requires a valid feature configuration without unresolved variability. Then, the feature-to-code mappings stored in the repositories are used to compose the required code artifacts based on the provided configuration. That means, the composition uses the feature-to-code mappings to construct the source files, and possibly other artifacts, based on the configuration.

Commit Features. A developer uses this operation to submit the changes made in the product development to the repository. The developers are supposed to provide their 'intention', i.e., information about the feature or set of features that have been changed [30]. Based on this information, the feature-to-code mappings will be created and updated. Similar to distributed code repositories like Git, *Commit Features* is just a local operation and does not create any conflicts. If the committed code artifacts affects already existing features, new feature versions are created automatically.

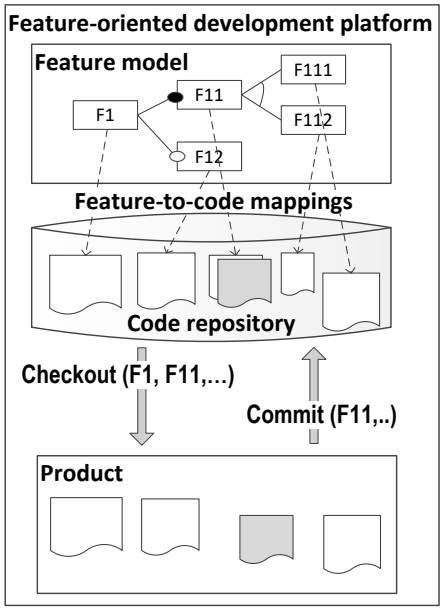

Fig. 2: A platform model consists of a feature model, feature-to-code mappings, a code repository, and derived products.

C. External Operations

External (inter-platform) operations are for exchanging features between platforms and the operations *Clone Features*, *Push Feature*, *Pull Feature* and *Transfer Feature* (cf. Fig. 1) are supported.

Clone Features. The operation *Clone Features* allows creating a new platform from an existing for a subset of its features (cf. challenge *C1*). After selecting a set of features available in the original platform, the operation clones the feature-to-code mappings and code artifacts of the selected feature subset to a new repository. An important difference to clone operations in common version control systems is that the customized platforms contain only the subset of the features required for a specific development task. During this operation the feature model in the original platform is pruned to just contain the features provided in the configuration. After the clone is completed, the developer will start adapting and extending the source code and feature model by checking out and committing features as shown above.

Push Features. The *Push Features* operation allows pushing one or more features developed in a cloned platform back to the original platform. At the surface this operation looks similar to a push command in Git, however, pushing again works at the level of features. The operation adds code artifacts to the repository of the original platform and also merges feature models, possibly resulting in new feature versions (cf. challenge *C3*). Merge conflicts may occur in this distributed

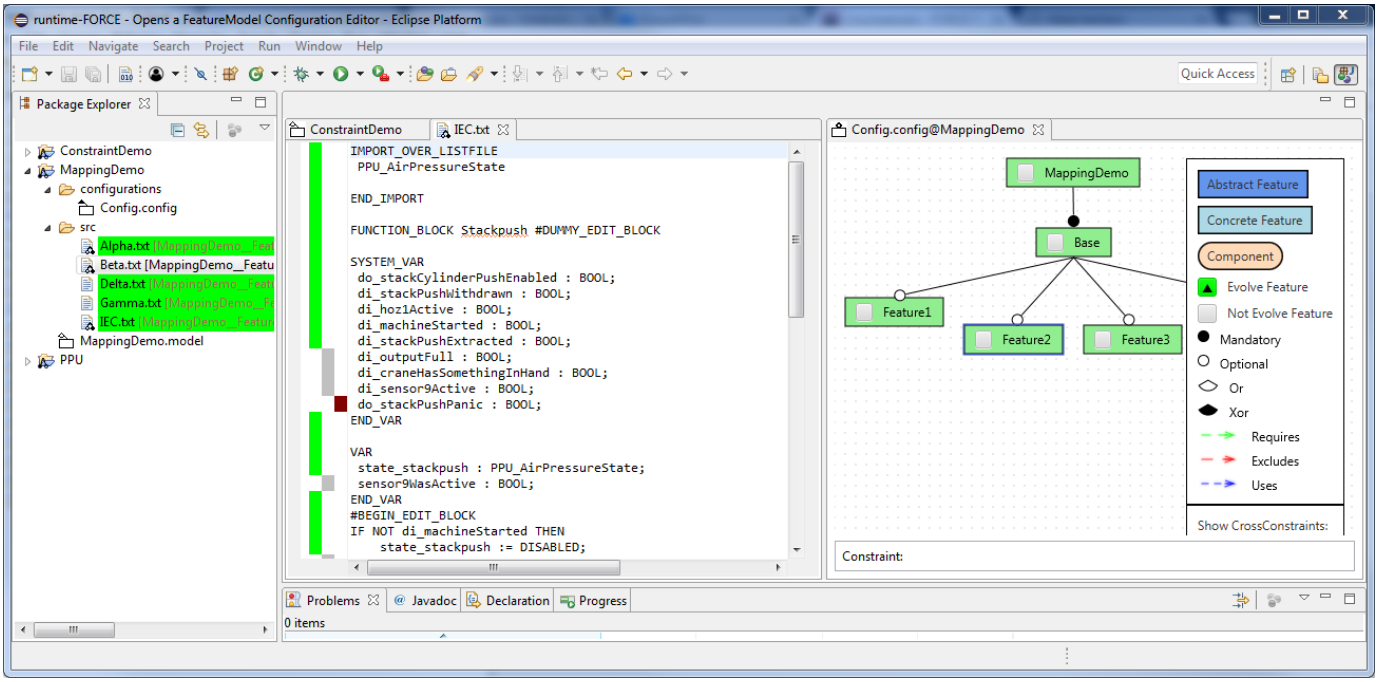

Fig. 3: Screenshot of the FORCE^2 development environment.

scenarios: for instance, a feature pushed from some platform back to its origin may already exist in the origin platform after a push of a similar feature with the same name from a sibling platform. Furthermore, beside pushing all changes of a cloned platform back to its origin platform, the operation supports pushing only a selected feature or feature set.

Pull Features. A key challenge in industrial SECOs is to update features of cloned platforms and derived products after new versions of the origin platform are released. The *Pull Features* operation allows a developer of a cloned platform to update features to new versions provided by the original platform (cf. challenge *C4*). Again, a fine-grained selective approach is supported, which allows to update only the desired features. Difficulties can be expected in this case if the feature on the cloned platform has been changed significantly, resulting in conflicts with the feature update. Also, changes could have been made to the feature model and feature dependencies in the origin or the cloned platform, which would result in conflicts that need to be resolved.

Transfer Features. Another evolution challenge described above (cf. challenge *C2*) is to transfer features from one derived product to another one, i.e., features implemented in one platform may be reusable in other platforms, even when pushing the feature to the original platform may not make sense. The *Transfer Features* operation supports developers who want to exchange some features between platforms. The operation copies the transferred feature's implementation from the source platform repository to the target repository and adds the feature to the feature model. In our ongoing research we investigate how to support developers in placing the transferred feature in the target feature model and how to adapt the feature-to-artifact traces in the repository based on the new position in the feature model. Furthermore, we plan to raise awareness of dependencies and constraints that might be violated.

IV. Tool Implementation

We have implemented the approach as proposed in Section III in the FORCE^2 Eclipse-based development environment. The main components of FORCE^2 are (i) ECCO which serves as the feature-oriented and variability-aware version control system, (ii) a reader/writer framework for parsing and persisting the different program artifacts, i.e., source code in different languages as well as configuration settings, (iii) a graphical feature modeling framework, and (iv) a project explorer for managing the various platforms and their artifacts. Currently the tool supports various artifacts, i.e., PLC code in a dialect of the IEC 61131-3 languages from our industry partner, Java, and different configuration settings.

The screenshot in Fig. 3 shows FORCE^2 with its feature modeling tool, a tree view showing the artifact tree of a currently checked out product configuration, and a source code editor. The editor highlights the code elements which are mapped to a selected feature of the control component.

V. Application Scenarios

As a proof of concept we illustrate the developement operations and its implementation in FORCE^2 on the Pick-and-Place Unit (PPU), a well-known example of a manufacturing system for transporting and sorting different workpieces taken from [19]. In [19] and [2] the PPU system is used as a case study system illustrating the evolution of automation systems by describing different evolution steps leading to

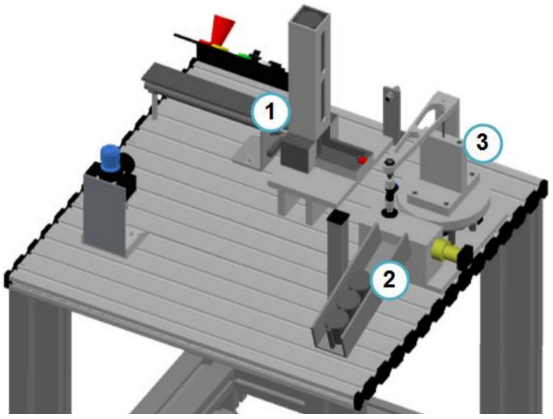

Fig. 4: Structure and components of the basic PPU system: (1) Stack, (2) Ramp, (3) Crane (taken from [19]).

various variations of the system. The basic version of the PPU comprises three components: a *Stack*, a *Crane*, and a *Ramp* (cf. Fig. 4). In further evolution steps, further components *Stamp* and *Sorter* are added to the system.

We have been implementing various versions and variants of the PPU for demonstrating evolution support in FORCE[2]. Our PPU systems are built up from several subsystems using different technologies: a control program written in Structured Text, an operator visualization panel implemented in Java, a Java component simulating the physics of the machine, as well as the configuration of the system with variable mappings connecting the different subsystems in simple text files. Thus, our implementation of the PPU system comprises artifacts of various types, which is common in industrial automation systems. Moreover, the implementation of the PPU features usually spans different subsystems and technologies. This means that checking out a PPU product by selecting features results in composing a system containing different technologies. Starting from a base version, we have been evolving the PPU to include 14 versions, allowing us to validate the different operations of our approach.

Based on the evolution scenarios described in [19], we present representative examples to illustrate how our approach supports feature-oriented and distributed development and evolution. Fig. 5 depicts the evolution timeline of a scenario we selected for demonstrating the key operations. Each platform in the scenario is shown using a lifeline and snapshots of its feature model after model-changing operations. The figure then shows the different operations used for performing the evolution steps.

The original platform *Base* represents the starting point. The initial version (*Base, v1*) contains features for controlling the three basic components and the optional *Stamp* component. In a first scenario, the developer starts the evolution by using the *Clone Features* operation (operation 1) to create a new platform *Clone A*, which is used to extend the control feature for *Crane*. The clone is created from a set of features, but does not include the feature *Stamp*. After cloning the own platform, the developer performs a checkout (operation 2) and generates the concrete product *P1* for implementing an extension. The developer adds the feature *InductiveSensor* to the feature model and writes the code for the inductive sensor feature. By committing the changes (operation 3) the added code will be added in the repository and mapped to the new feature *InductiveSensor* (*Clone A, v2*).

Using a second clone operation, a new platform *Clone B* is created from the base platform (operation 4). In this new platform, a new optional feature *MicroSwitch* is added to the *Crane* feature by the developer (*Clone B, v2*). This again happens on a concrete product *P2* in a checkout-code-commit cycle (operations 5 and 6). However, the developer also has to add an *InductiveSensor* feature and would like to reuse the feature from the sibling platform. The *Transfer Feature* operation allows to reuse the feature from the sibling platform (operation 7). It takes the feature together with the mapped code artifacts and integrates them into the platform, i.e., the feature will be added to the feature model, the code will be added to the repository, and the feature mappings will represent the resulting dependencies (*Clone B, v3*).

However, there might be some inconsistencies and conflicts due to the changes already made in the platform when implementing the feature *MicroSwitch*. Therefore, the developer has to resolve the conflicts in the implementation. For that purpose, the developers checks out a configuration containing both new features (operation 8) *InductiveSensor* and *MicroSwitch* to obtain concrete product *P3* and changes the implementation so that conflicts are resolved. When committing the changes, new versions of the features *InductiveSensor* and *MicroSwitch* are added to the platform (operation 9). Support for resolving such conflicts is essential for developers and will be part of our future research (c.f. Section VII).

Further, the team decides to make the feature *InductiveSensor* in the version of platform *(Clone B, v3)* generally available via the original platform. The developer adds and merges the new features back to platform *Base* using *Push* (operation 10). Again, the *Push Feature* operation will extract all the code artifacts for feature *InductiveSensor*, i.e., feature, code and traces, and integrate them into the base platform. Possibly, the developer responsible for the base platform will have to resolve conflicts and/or make some improvements by implementing a new version of the feature in a checkout-code-commit cycle. The result is a new version of platform *Base, v2*.

Finally, the developer of the first cloned platform *Clone A* notices that the version of feature *InductiveSensor* in the *Base* platform contains some improvements, which would also be beneficial. Therefore, the developer updates the solution of *InductiveSensor*. The *Feature Update* operation allows to selectively update feature *InductiveSensor* from the base platform (operation 11) resulting in a new version of platform (*Clone A, v3*) with a new version *v2* of feature *InductiveSensor*.

VI. Related Work

We discuss related research in the areas of product line engineering, clone-and-own reuse, variation control systems,

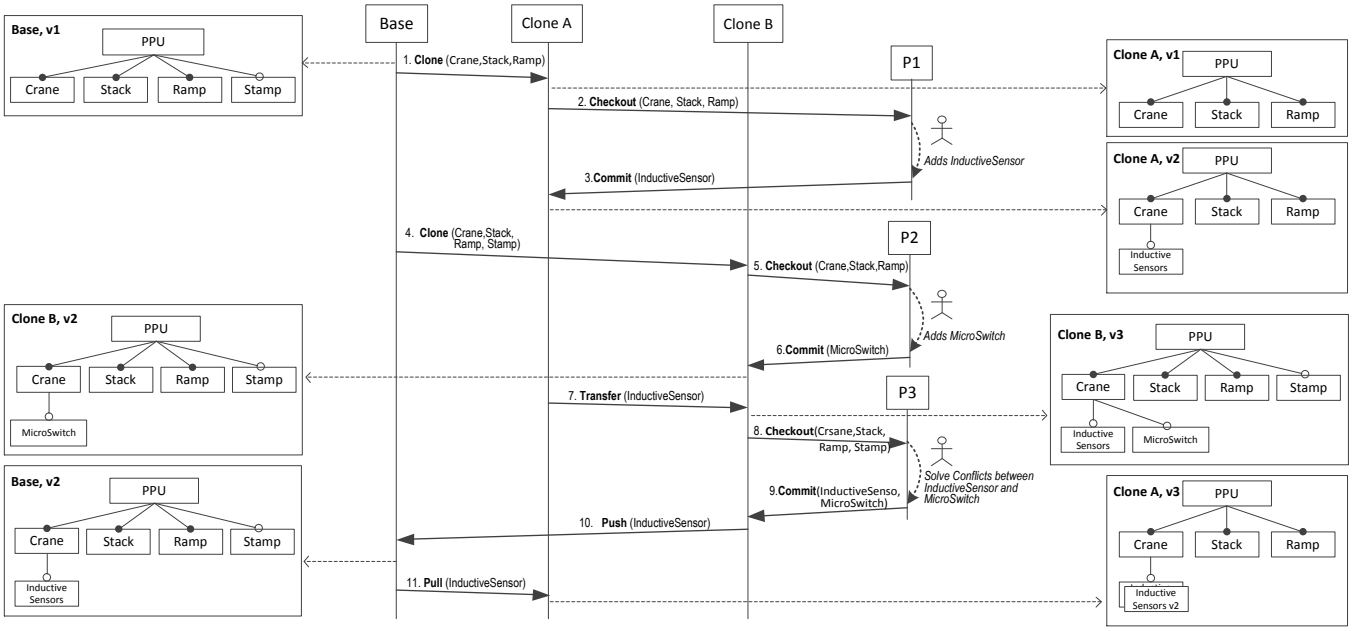

Fig. 5: Evolution timeline illustrating the extension of the crane with an inductive sensor and a micro switch in the Pick-and-Place Unit (PPU) example.

distributed code repositories, and evolution in industrial automation.

Product line engineering: Several techniques proposed in this area address the evolution of product lines, however, with a focus on single product lines. For instance, Pleuss *et al.* present model-driven support for product line evolution on feature level. In this approach evolutions are represented as model fragments and applied to concrete products [32]. Passos *et al.* propose evolution patterns for the co-evolution of variability models and source code artifacts by means of the Linux kernel [33]. The issue of interrelated product lines has been investigated in the context of multi product lines, i.e., "sets of several self-contained but still interdependent product lines that together represent a large-scale system" [34]. For instance, several approaches have been proposed to structure variability models and to check the consistency among them. The model fragments approach [35] defines guidelines for structuring the modeling space while the compositional variability management (CVM) framework [36] allows defining a product hierarchy in a multi product line. Consistency checking in multi product lines can be achieved by checking variability models across product line boundaries during modeling and during distributed product derivation. An example is the product line configuration (PLiC) framework [37], which automatically ensures consistency during product configuration by invoking builders on diverse configuration changes.

Clone-and-own reuse: Several approaches provide support for creating and managing clones in product line engineering. For instance, Rubin *et al.* [6] present an operator framework covering atomic operations used to perform common cloning activities. Rabiser *et al.* present a modeling approach based on prototypes, i.e., prefabricated objects from which clones are created [9]. Similar to the operations presented in this paper, the approach considers clones at different levels (products, components, features). However, the approach focuses on the variability modeling aspects and does not address aspects of distributed version control. Fischer *et al.* [8] present the ECCO (Extraction and Composition for Clone-and-Own) approach to refactor cloned variants into software product lines, i.e., comparing different product variants in retrospect to extract feature-to-code traces, interactions between features, and dependencies between traces.

Distributed code repositories: Pull-based development processes play an important role in distributed development scenarios. However, empirical studies confirm the difficulty of this approach: for instance, Gousios et al. [21] investigate reasons for not merging pull requests and show that almost one third of unmerged pull requests are closed as no longer relevant, e.g., if a feature is already implemented in another branch or if the pull request would duplicate already existing functionality. Similarly, in [24] the authors show that pull requests are often rejected due to problems with code quality, but also if they become outdated due to newer pull requests. Other approaches have thus proposed role-based collaboration models for SECOs to raise awareness among engineers about their currently ongoing development activities [38].

Variation control systems: In industrial practice the branching and forking mechanisms of version control systems are used to manage features and variabilities. Montalvillo et al. [39] introduced a branching model and operations for GitHub, trying to provide better support of a version control systems to be used in SPL development. However,

in a recent survey Linsbauer *et al.* [30] show that a number of variation control systems provide specific capabilities to support the handling of variants of products and product features. Specifically, a key component of our FORCE^2 platform is the version control system ECCO [26]–[29], which also provides variation control support at the level of features. Is similar approach is the SuperMod system [40] providing feature-oriented support in the area of model-driven software product line engineering. SuperMod also supports a collaborative development environment and support for merging and solving conflicts [41]. However, SuperMod is currently limited to EMF models and may be difficult to apply in industrial SECOs with their diverse artifact base.

Evolution in industrial automation: Vogel-Heuser *et al.* [2] comprehensively discuss software evolution challenges in automated production systems. They outline the state-of-the-art, challenges and future research directions along the stages of the development process for automated production systems. Moreover, they describe variability management as a key technique in automated production system engineering, as automated production system are usually highly diverse. As key challenges they see variability modeling being cross-cutting and evolution of variability models. We argue, that our approach addresses those challenges: the repository allows maintaining artifacts crossing different disciplines. Moreover, feature models support feature versions and the repository is built as a feature-aware version control system.

Another interesting approach related to our work is presented in [42]. The paper proposes an approach for exploiting the evolution experience as well as performance observations in a fleet of similar machines for supporting evolution. Given a specific evolution task, experience from similar evolution scenarios already observed within the machine fleet represents the basis to carry the evolution step. Similar to our approach, the goal is to transfer evolution knowledge and their implementation steps between machine solutions.

VII. Conclusions and Future Work

In this paper we outlined our research activities towards a tool-supported approach for distributed and feature-oriented development and evolution in SECOs. We expect that the capabilities and operations for distributed feature-oriented development will be essential in future SECO development scenarios. We illustrated our approach and our FORCE^2 implementation using application scenarios from the PPU example.

As already indicated, supporting developers in resolving conflicts which can occur when pushing, pulling, or transferring features will be a major research direction. In fact, in previous work [10], [43], [44] we have developed a configuration-aware dependency and change impact analysis technique. The approach builds on a system dependence graph which encodes all the control and data dependencies in a program and also is capable of considering the configuration of systems. Currently, we are working on a technique for lifting the dependencies found at source code level to the level of features, which will allow developers to see if and how features are interacting. We expect that such information on feature interactions together with a comparison of the changed features, will provide important insights to analyse and resolve possible conflicts.

To evaluate our approach and its FORCE^2 implementation we plan to follow a similar process as for the PPU system. The goal is to replay evolution scenarios from real-world systems, e.g., from our industry partner KEBA, or based on mining forks and pull-requests from open source repositories.

Acknowledgment

This work has been conducted in cooperation with KEBA AG, Austria, and was supported by the Christian Doppler Forschungsgesellschaft, Austria, and the Austrian Science Fund (FWF) project P25289-N15.

References

[1] B. Vogel-Heuser, J. Fuchs, S. Feldmann, and C. Legat, "Interdisciplinary Product Line Approach to Increase Reuse (Interdisziplinärer Produktlinienansatz zur Steigerung der Wiederverwendung)," *Automatisierungstechnik*, vol. 63, no. 2, pp. 99–110, 2015.

[2] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[3] J. Bosch, "From software product lines to software ecosystems," in *Proceedings 13th International Software Product Line Conference*. Carnegie Mellon University, 2009, pp. 111–119.

[4] D. Lettner, F. Angerer, P. Grünbacher, and H. Prähofer, "Software evolution in an industrial automation ecosystem: An exploratory study," in *Proceedings International Euromicro Conference on Software Engineering and Advanced Applications*, 2014.

[5] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *Software Product Lines*. Springer, 2004, pp. 266–283.

[6] J. Rubin, K. Czarnecki, and M. Chechik, "Managing cloned variants: a framework and experience," in *Proceedings of the 17th International Software Product Line Conference*, 2013, pp. 101–110.

[7] D. Lettner, F. Angerer, H. Prähofer, and P. Grünbacher, "A case study on software ecosystem characteristics in industrial automation software," in *Proceedings International Conference on Software and Systems Process*, 2014, pp. 40–49.

[8] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Enhancing clone-and-own with systematic reuse for developing software variants," in *Proceedings 30th International Conference on Software Maintenance and Evolution*, 2014, pp. 391–400.

[9] D. Rabiser, P. Grünbacher, H. Prähofer, and F. Angerer, "A prototype-based approach for managing clones in clone-and-own product lines," in *Proceedings 20th International Software Product Line Conference*, Beijing, China, 2016, pp. 35–44.

[10] F. Angerer, A. Grimmer, H. Prähofer, and P. Grünbacher, "Configuration-aware change impact analysis," in *Proceedings 30th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'15, 2015, pp. 385–395.

[11] M. Antkiewicz, W. Ji, T. Berger, K. Czarnecki, T. Schmorleiz, R. Lämmel, S. Stanciulescu, A. Wasowski, and I. Schaefer, "Flexible product line engineering with a virtual platform," in *Proceedings 36th International Conference on Software Engineering, ICSE'14, Hyderabad, India*, 2014, pp. 532–535.

[12] A. N. Duc, A. Mockus, R. L. Hackbarth, and J. D. Palframan, "Forking and coordination in multi-platform development: a case study," in *Proceedings ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM'14, Torino, Italy*, 2014, pp. 59:1–59:10.

[13] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, and K. Czarnecki, "What is a feature? a qualitative study of features in industrial software product lines," in *Proceedings 19th International Software Product Line Conference (SPLC'15)*. ACM, 2015, pp. 16–25.

[14] D. Rabiser, H. Prähofer, P. Grünbacher, M. Petruzelka, K. Eder, F. Angerer, M. Kromoser, and A. Grimmer, "Multi-purpose, multi-level feature modeling of large-scale industrial software systems," *Software and Systems Modeling*, vol. 17, p. 913–938, 2018.

[15] M. Zou and B. Vogel-Heuser, "Feature-based systematic approach development for inconsistency resolution in automated production system design," in *13th Conference on Automation Science and Engineering (CASE 2017)*, Xi'an, China, 2017.

[16] M.-O. Reiser and M. Weber, "Multi-level feature trees," *Requirements Engineering*, vol. 12, no. 2, pp. 57–75, 2007.

[17] T. Simon, J. Fischer, and B. Vogel-Heuser, "Variability management for automated production systems using product lines and feature models," in *Proceedings of the 14th IEEE International Conference on Industrial Informatics (INDIN)*, Poitiers, France, 2016.

[18] H. Prähofer, D. Rabiser, F. Angerer, P. Grünbacher, and P. Feichtinger, "Feature-oriented development in industrial automation software ecosystems: Development scenarios and tool support," in *Proceedings 14th IEEE International Conference on Industrial Informatics (INDIN 2016)*, Poitiers, France, 2016, pp. 1218–1223.

[19] 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," Technische Universität München, Tech. Rep., 2014.

[20] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke, "Automated software transplantation," in *Proceedings International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2015, pp. 257–269.

[21] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model," in *36th International Conference on Software Engineering (ICSE)*, 2014, pp. 345–355.

[22] D. Lettner and P. Grünbacher, "Using feature feeds to improve developer awareness in software ecosystem evolution," in *Proceedings 9th International Workshop on Variability Modelling of Software-intensive Systems*, 2015, pp. 11–18.

[23] T. Pfofe, T. Thüm, S. Schulze, W. Fenske, and I. Schaefer, "Synchronizing software variants with variantsync," in *Proceedings 20th International Systems and Software Product Line Conference*, 2016, pp. 329–332.

[24] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *Proceedings 37th International Conference on Software Engineering*, 2015, pp. 358–368.

[25] S. Schulze, M. Schulze, U. Ryssel, and C. Seidl, "Aligning coevolving artifacts between software product lines and products," in *Proceedings 10th International Workshop on Variability Modelling of Software-intensive Systems*, 2016, pp. 9–16.

[26] L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Recovering traceability between features and code in product variants," in *Proceedings 17th International Software Product Line Conference, Tokyo, Japan*, 2013, pp. 131–140.

[27] L. Linsbauer, F. Angerer, P. Grünbacher, D. Lettner, H. Prähofer, R. Lopez-Herrejon, and A. Egyed, "Recovering feature-to-code mappings in mixed-variability software systems," in *Proceedings 30th International Conference on Software Maintenance and Evolution*, 2014, pp. 426–430.

[28] L. Linsbauer, A. Egyed, and R. E. Lopez-Herrejon, "A variability-aware configuration management and revision control platform," in *Proceedings 38th International Conference on Software Engineering*, 2016, pp. 803–806.

[29] L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Variability extraction and modeling for product variants," *Software & Systems Modeling*, pp. 1–21, 2016.

[30] L. Linsbauer, T. Berger, and P. Grünbacher, "A classification of variation control systems," in *Proceedings of the 16th International Conference on Generative Programming: Concepts & Experience (GPCE'17)*. New York, NY, USA: ACM, 2017, pp. 49–62.

[31] L. Linsbauer, S. Fischer, R. E. Lopez-Herrejon, and A. Egyed, "Using traceability for incremental construction and evolution of software product portfolios," in *Proceedings 8th International Symposium on Software and Systems Traceability*, 2015, pp. 57–60.

[32] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, "Model-driven support for product line evolution on feature level," *JSS*, vol. 85, no. 10, pp. 2261–2274, 2012.

[33] L. Passos, L. Teixeira, N. Dintzner, S. Apel, A. Wasowski, K. Czarnecki, P. Borba, and J. Guo, "Coevolution of variability models and related software artifacts," *Empirical Software Engineering*, 2015.

[34] G. Holl, P. Grünbacher, and R. Rabiser, "A systematic review and an expert survey on capabilities supporting multi product lines," *Information & Software Technology*, vol. 54, no. 8, pp. 828–852, 2012.

[35] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer, "Structuring the modeling space and supporting evolution in software product line engineering," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1108–1122, 2010.

[36] A. Abele, Y. Papadopoulos, D. Servat, M. Törngren, and M. Weber, "The CVM framework – A prototype tool for compositional variability management," in *Proceedings 4th International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria*, 2010, pp. 101–105.

[37] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat, "Consistent product line configuration across file type and product line boundaries," in *Proceedings 14th International Conference on Software Product Lines (SPLC'10), Jeju Island, South Korea*, 2010, pp. 181–195.

[38] Ştefan Stănciulescu, D. Rabiser, and C. Seidl, "A technology-neutral role-based collaboration model for software ecosystems," in *Proceedings 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2016); Track on Variability modelling for scalable software evolution*, Corfu, Greece, 2016.

[39] L. Montalvillo and O. Díaz, "Tuning GitHub for SPL development: Branching models & repository operations for product engineers," in *Proceedings of the 19th International Conference on Software Product Line*, ser. SPLC '15. New York, NY, USA: ACM, 2015, pp. 111–120.

[40] F. Schwägerl, T. Buchmann, and B. Westfechtel, "Supermod – A model-driven tool that combines version control and software product line engineering," in *Proceedings of the 10th International Conference on Software Paradigm Trends*, 2015, pp. 5–18.

[41] F. Schwägerl, T. Buchmann, S. Uhrig, and B. Westfechtel, "Towards the integration of model-driven engineering, software product line engineering, and software configuration management," in *3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2015, pp. 1–14.

[42] C. Haubeck, A. Chakraborty, J. Ladiges, A. Fay, A. Pokahr, and W. Lamersdorf, "Evolution of cyber-physical production systems supported by community-enabled experiences," in *Proceedings 15th IEEE International Conference on Industrial Informatics (INDIN 2017)*, Emden, Germany, 2017, pp. 867–874.

[43] F. Angerer, H. Prähofer, D. Lettner, A. Grimmer, and P. Grünbacher, "Identifying inactive code in product lines with configuration-aware system dependence graphs," in *Proceedings 18th International Software Product Line Conference*, 2014.

[44] A. Grimmer, F. Angerer, H. Prähofer, and P. Grünbacher, "Supporting program analysis for non-mainstream languages: Experiences and lessons learned," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 460–469.