Feature Modeling of Two Large-Scale Industrial Software Systems: Experiences and Lessons Learned

Daniela Lettner* Klaus Eder† Paul Grünbacher* Herbert Prähofer‡
*Christian Doppler Laboratory MEVSS ‡Institute System Software
Johannes Kepler University Linz, Austria
†KEBA AG, Linz, Austria

***Abstract*—Feature models are frequently used to capture the knowledge about configurable software systems and product lines. However, feature modeling of large-scale systems is challenging as many models are needed for diverse purposes. For instance, feature models can be used to reflect the perspectives of product management, technical solution architecture, or product configuration. Furthermore, models are required at different levels of granularity. Although numerous approaches and tools are available, it remains hard to define the purpose, scope, and granularity of feature models. In this paper we thus present experiences of developing feature models for two large-scale industrial automation software systems. Specifically, we extended an existing feature modeling tool to support models for different purposes and at multiple levels. We report results on the characteristics and modularity of the feature models, including metrics about model dependencies. We further discuss lessons learned during the modeling process.**

***Index Terms*—feature modeling, industrial software systems, experience report.**

I. Introduction and Motivation

Feature modeling was originally proposed as part of the FODA method to elicit and represent commonalities and variability of systems' capabilities in a specific domain [1]. Feature models define features—the end-users' (and customers') understanding of the general capabilities of systems in the domain—and their relationships. Feature models, and variability models in more general, are nowadays widely used to capture the knowledge of domain experts regarding customer-facing features, system capabilities and properties, as well as configuration options [2], [3]. The term feature is commonly used by customers, product managers, and engineers to communicate about product characteristics [4]. However, although numerous approaches and tools are available [5], defining the purpose, scope, and granularity of feature models remains hard, specifically when modeling large-scale industrial software systems.

Regarding the *purpose* of feature models, researchers have distinguished different modeling spaces [6], [7]: *problem space* features generally refer to systems' specifications established during domain analysis and requirements engineering; *solution space* features refer to the concrete systems created during development; whereas *configuration space* features exist to ease the derivation of products. Regarding the *scope* of feature models in large-scale systems there is a consensus that single monolithic feature models are inadequate to deal with the complexity of industrial systems [8]–[10]. This has led, e.g., to the development of multi product line approaches that support modularizing feature models in various ways, as pointed out in a systematic review [11]. Similarly, it has been shown that feature models vary with respect to their *granularity*, e.g., to distinguish high-level system features from lower-level capabilities. Moreover, *dependencies* between different feature models need to be managed. For instance, it is often unclear how problem space features describing customer-facing capabilities and their variability are related to solution space features implementing this functionality; or how configuration space features are related to configuration options used by service engineers for customizing and fine-tuning a system.

Only few reports are available on how these different feature models should be structured and organized, and what kind of dependencies need to be considered. In particular, there is a lack of reports on feature modeling in large-scale systems. Organizations moving towards a product line approach or feature-oriented development paradigm can benefit from examples and lessons learned when planning their own modeling approach.

In this paper, we thus present experiences and lessons learned of developing feature models for two large-scale software systems in the domain of industrial automation. The modeling process, performed with our industry partner Keba as part of an ongoing research cooperation, allowed us to study the purpose, scope, and granularity of feature models. Furthermore, we adapted and extended a state-of-the-art feature modeling approach to better support the characteristics and needs of large-scale industrial software systems. In particular, we extended the FeatureIDE [12], an Eclipse-based feature modeling tool, to improve support for multi-purpose, multi-level feature models. Our report can be useful for practitioners facing challenges of modularizing feature models and managing dependencies between features in the problem space, solution space and configuration space.

The paper is organized as follows: we proceed by summarizing the industrial context and modeling requirements of Keba in Section II. Section III presents our modeling approach. Section IV describes the two investigated industrial systems, our modeling process, and the extensions to the FeatureIDE modeling tool. Section V presents results on model characteristics and Section VI discusses lessons learned. Section VII relates our work with existing research on variability modeling of large-scale systems. Section VIII rounds out the paper with a conclusion and an outlook on future work.

II. Industrial Modeling Requirements

Keba (http://www.keba.com) develops and produces hardware and software solutions for industrial automation. Their products are globally marketed and exist in numerous variants to address the requirements of customers in different market segments such as injection molding machines, robotics, or heating control systems.

As part of a research cooperation we recently studied the development practices of KePlast, a 3.8 million LoC system for the automation of injection molding machines [4], [13], [14]. The data collected in workshops and interviews with Keba's senior developers, software architects, and product managers allowed us to better understand the industrial context for feature modeling in the large. Specifically, we found that the term feature is widely used in the company. Not surprisingly, the meaning of the term depends on role-specific perspectives and needs in Keba's current development process. Product managers, for instance, use features to define the scope of products from a market and customer perspective. They document problem space features in product maps, i.e., matrices that allow comparing related products over numerous features. These spreadsheets comprise high-level system features, feature associations, available hardware options, and references to order numbers used by sales people. Architects use UML class diagrams for modeling and documenting solution space features. Software engineers further use a wide range of mechanisms to implement variability, e.g., interfaces to hook in new functionality; capabilities for adding, exchanging or reloading modules; or support for defining specific I/O ports. Keba uses a custom-developed configurator that defines configuration space features guiding the derivation of KePlast applications [15]. This tool allows deriving initial products, which are then customized by developers, who adapt existing features and add new features to meet the customers requirements. Similarly, in the robotics domain Keba offers a configuration tool as part of their IDE to facilitate product customization.

A. An Example of a Feature

We use the KePlast feature MoldCavityPressureSensor to illustrate that features exist in different forms and for different purpose. In injection molding machines, the polymer raw material is injected into a mold to shape it into the desired form [16]. Molds can be of a single cavity or multiple cavities. In multiple cavity molds, cavities can be identical and form equal parts, however, cavities can also be unique and form multiple different parts [17]. Cavity sensing is used to provide a quality index of the injection-molded part. A pressure signal is used for determining whether the cavity pressure curve is repeatable between shots. The measured cavity pressures indicate the quality of the produced parts. In case of anomalies, it is likely that the quality of the produced parts degrades.

A representation of the feature MoldCavityPressureSensor can be found in each modeling space and traces exist in spreadsheets, KePlast platform code, and the source code of the configuration tool. The feature is documented in the problem space as an option in three of four variants of the KePlast product map. In the solution space, the feature is represented by the variable `hw_CavityPressure`. The sensor is also shown on several user interface masks. Finally, regarding the configuration space, the custom-developed configurator allows selecting up to four sensors for measuring injection-molded parts.

The example shows that the different stakeholder groups manage the feature well within their scope of responsibility. However, there are only few explicit links between system features documented in spreadsheets, configuration options documented in configurators, and software features defined in UML diagrams, specification documents, or user manuals.

B. Requirements for Modeling

A feature modeling approach is needed for managing multiple modeling spaces and inter-space dependencies between features. The approach also needs to better support modularization to facilitate a divide-and-conquer modeling strategy needed to deal with the complexity of large-scale industrial systems. Specifically, the approach needs to meet three requirements:

Requirement 1 – Feature models for different purpose. The modeling spaces proposed in the literature (e.g., [6], [7]) are useful to distinguish different types of feature models in complex systems. Feature models need a clear purpose allowing a modeler to distinguish between customer-facing features, software capabilities, and configuration decisions.

Requirement 2 – Feature models at different levels. Feature models are needed at multiple levels of abstraction in the different modeling spaces. Product managers may need to describe groups of product features at different levels of granularity. Also, the multi-layered architecture of many large-scale systems suggests the use of hierarchically organized variability models, as e.g., proposed in hierarchical product lines. Furthermore, the need for complex product configuration in multiple stages [18] calls for multiple levels of models in the configuration space.

Requirement 3 – Dependencies between different feature models. Modelers need to explicitly define dependencies between feature models of different purpose that exist at different levels. Many approaches have been proposed in this regard, e.g., to model dependencies between different modeling spaces [19], [20], between models of one space [21], [22] or between different levels of abstraction [9], [23].

III. Modeling Approach

We extend feature modeling to support different modeling spaces, modeling variability at different abstraction levels in

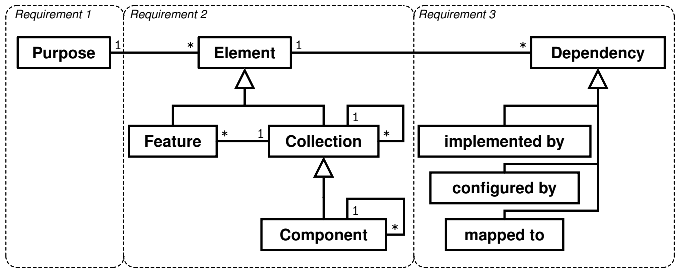


Fig. 1: Modeling approach supporting feature models for different purpose, at different levels, and dependencies between models.

each space, and dependencies between spaces. Figure 1 shows a conceptual overview of our modeling extensions.

Feature models for different purpose. We assume multiple feature models, each serving a different *purpose*. We follow the idea of separating spaces proposed in [6] and distinguish problem space, solution space, and configuration space features as a foundation for defining role-specific views for product management, software architecture, and product configuration.

Feature models at different levels. Our modeling approach is influenced by the idea of configurable units, which was proposed in the Common Variability Language (CVL) [24]. We use *collections* to support variability modeling at different levels in each modeling space. Collections logically group features and can be organized hierarchically to structure large models. A *component* is a special kind of a collection used to represent physical elements of a system (e.g., software modules).

Dependencies between different feature models. Cross-tree constraints (i.e., `requires` and `excludes`) have been originally proposed in [1]. More complex relationships in the form of generic propositional formulas have been proposed later in the literature [25]. Beyond that, our extensions allow defining different types of inter-space dependencies when relating features or collections from different modeling spaces. For instance, the problem space feature MoldCavityPressureSensor is implemented_by the solution space user interface mask Injection-Mask1. The configuration space option CavityPressureSensor of KePlast's configurator is mapped_to the solution space variable `hw_CavityPressure`. The problem space feature MoldCavityPressureSensor documented in a product map is configured_by the configuration space option CavityPressureSensor.

IV. Modeling KeMotion and KePlast

Keba is currently exploring the benefits of the feature-oriented software development paradigm, which is seen as promising to ease software maintenance, to create awareness for feature reuse, to automate product derivation, and to improve documentation. The term feature is widely used in the company to communicate during development and maintenance, independent of the specific methods and technologies used [4]. For instance, sales people identify the needs of potential customers in terms of new system features that are required. Product managers drive the development of different KeMotion and KePlast product variants by defining product line features addressing market needs. Domain engineers develop and maintain product features and provide the required variability. Application engineers build concrete applications by selecting, adapting and extending features meeting the specific customer requirements. Commissioning engineers fine-tune systems by calibrating the properties of features.

We have been developing feature models of KeMotion and KePlast, two industrial systems comprising hardware and software components. The aim of partially modeling software aspects of the two systems was to better understand the purpose, scope, granularity, and dependencies of features and feature models in this organization. The goal of our modeling activities was not to completely model the investigated systems. However, the experience still allowed us to learn how multi-purpose, multi-level variability models can be organized. In particular, we report model metrics and insights related to modeling spaces, levels and dependencies. Furthermore, the modeling process allowed us to test the modeling tool and to gain experiences by applying it to two large-scale industrial systems. Specifically, we investigated two research questions:

RQ1. *How useful are multi-purpose, multi-level feature models for large-scale industrial systems?* We explore whether organizing feature models in terms of distinct modeling spaces and multiple modeling levels makes sense. This research question addresses specifically the breadth of the resulting models and the coverage of different spaces and levels.

RQ2. *What are characteristics of specific modeling spaces?* We modeled specific areas of the systems in detail to gain in-depth results of selected feature models. In particular, we describe detailed metrics measuring KePlast's problem space and configuration space models.

A. Industrial Systems

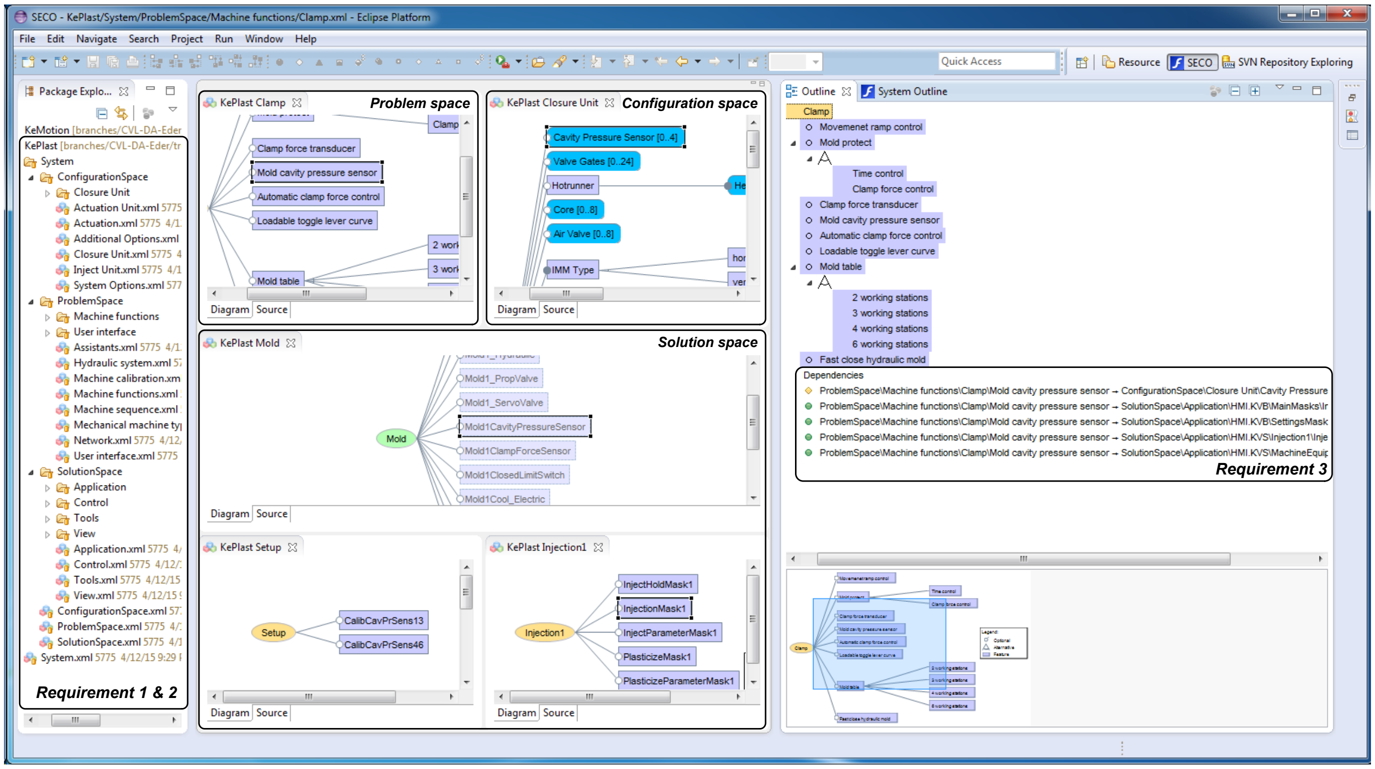
KeMotion (2.7 million LoC) is a control system for robotics, comprising a software platform as well as hardware control units and mobile display units. The system offers all types of interpolation, unlimited in the 6D space (position and orientation). KeMotion covers the entire motion spectrum of the robot, from track-consistent, shortest possible point-to-point movements or driving of individual robot axes. Besides its motion capabilities, the system offers guided programming and execution of robot sequences.

KePlast (3.8 million LoC) is a comprehensive platform for the automation of injection molding machines, comprising a configurable control software framework, a visualization system, programming tools, and a configuration tool to customize solutions based on existing components and variants. The platform exists in various variants, e.g., there is one specific variant for the Chinese market.

B. Preparatory Steps

Before we started the modeling phase, several preparatory steps were conducted:

Analyzing representations of selected features in different modeling spaces. To better understand how people use features in product management, product configuration, and for



Dependencies
- ProblemSpace\Machine functions\Clamp\Mold cavity pressure sensor → ConfigurationSpace\Closure Unit\Cavity Pressure
- ProblemSpace\Machine functions\Clamp\Mold cavity pressure sensor → SolutionSpace\Application\HMI.KVB\MainMasks\Ir
- ProblemSpace\Machine functions\Clamp\Mold cavity pressure sensor → SolutionSpace\Application\HMI.KVB\SettingsMask
- ProblemSpace\Machine functions\Clamp\Mold cavity pressure sensor → SolutionSpace\Application\HMI.KVS\Injection1\Inje
- ProblemSpace\Machine functions\Clamp\Mold cavity pressure sensor → SolutionSpace\Application\HMI.KVS\MachineEquip

Fig. 2: Tool prototype showing KePlast feature models for different purposes (Requirement 1), at different levels of abstraction (Requirement 2), and with different types of dependencies (Requirement 3). The Eclipse Package Explorer shows the entry points for the three modeling spaces. The solution space elements Setup and Injection1 are collections whereas the solution space element Mold is a component. The FeatureIDE tool lists all dependencies available for a selected feature, it highlights related features, and allows navigating to related features.

development (e.g., regarding different variability mechanisms), we started with investigating exemplary features of KeMotion and KePlast. For instance, the KePlast feature ImpulseCounter needed for direct clamping in injection molding machines can be found in all modeling spaces: in product management the ImpulseCounter is represented as a standard function automatic mold height adjust for direct clamping machines. The feature can further be found in the product configuration tool, i.e., the common setup of the closure unit of an injection molding machine. At code level, the feature is reflected by a variable representing an endpoint to optional machine equipment like the mold-height-adjust sensor [26].

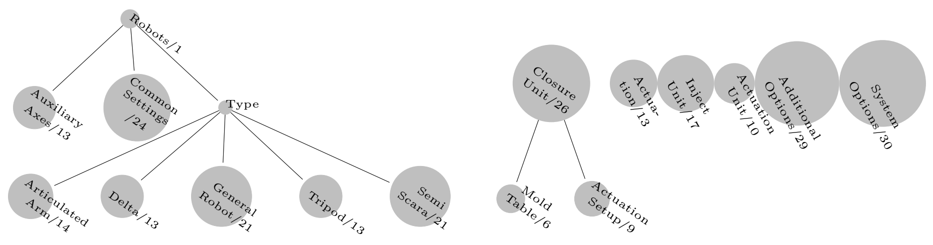
Prototypical modeling of selected subsystems. We then created initial feature models for KeMotion using SINTEF's CVL 2 tool prototype [27]. Specifically, we created five models comprising KeMotion's configuration space features and six models reflecting solution space features. The tool was selected as it adheres to the proposed CVL standard [24] and supports multiple interrelated feature models. Although the CVL 2 tool prototype was not mature enough for our purpose, the experiences helped us to better plan our extensions to the FeatureIDE tool suite.

Development of tool extensions. We implemented the modeling approach described in Section III using FeatureIDE [12]. We selected this state-of-the-art feature modeling tool because it is available as an open-source system and easily extensible. However, our modeling approach could be implemented in other feature modeling tools if needed. The workspace of the FeatureIDE tool suite can only manage an individual feature model. We developed extensions to manage multiple feature models at different levels as well as dependencies between inter-space features and collections. The extension points provided by FeatureIDE's core components did not support adding new types of modeling elements. Thus, our extensions were mainly done by exploiting inheritance, i.e., the existing model representation of the feature tree was extended to handle the new element types. Furthermore, the existing diagram editor was replaced to display and handle the diagram representations of new element types. Figure 2 shows a snapshot of the tool prototype and the KePlast model.

C. Modeling Process

We created the feature models in two steps:

Modeling strategy and data sources. Based on the CVL prototype models we started modeling KeMotion and KePlast, following a top-down modeling strategy for both systems. For KeMotion, the problem space models were created as a first step, despite no detailed product map was available at that time (this was only started recently by the company). We then focused on modeling the configuration space and analyzed the KeStudio configurator including the MotionWizard. The author in charge of modeling the solution space has detailed



(a) KeMotion (b) KePlast

Fig. 3: Configuration space feature models. During product configuration, configuration decisions are first taken for features in higher-level models, while lower-level models address more detailed configuration.

knowledge of KeMotion's code base. The resulting models thus provide a good coverage of the code base, however, they have not been completed for all subsystems (cf. RQ1). For KePlast, we started with creating the problem space models, based on an existing product map maintained by product managers. For creating the configuration space models we investigated KePlast's custom-developed configuration tool. Finally, we created solution space models, again emphasizing breadth over depth. We did not consider feature attributes that would be important for product derivation and feature selection. For instance, we did not address the binding time of the features.

Model validation and analysis. The author who created the multi-level feature models is involved with the KeMotion application for more than eight years and recently moved as a developer to the KePlast team. A second author cross-checked the created models and resulting metrics. The feature models were iteratively refined and validated in multiple discussions. Further, the created models were presented in a workshop with KeMotion and KePlast architects to get feedback and to clarify open issues. When cross-checking the feature models, we used a dictionary standardizing domain terminology [28]. For instance, the domain dictionary for injection molding describes the problem space features related to HotRunner as follows: *A hotrunner is used to maintain a molten flow of plastic from the injection molding machine nozzle to the gate in a plastic injection mold.* Such definitions are helpful to understand the meaning of the features.

V. Results and Experiences

We present results and experiences on the usefulness of feature modeling with multiple modeling spaces and levels. We report system-wide model characteristics (cf. RQ1) as well as detailed model characteristics for KePlast's problem and configuration space (cf. RQ2).

A. RQ1 – System-wide Model Characteristics

We report metrics on feature model properties as proposed by Berger et al. [29]. More specifically, we measure the created variability models with structural metrics concerning the size and shape of the models. Table I summarizes the number of features per type (mandatory, optional, alternative, and modeling space), collections and components, as well as inter-space and intra-space dependencies.

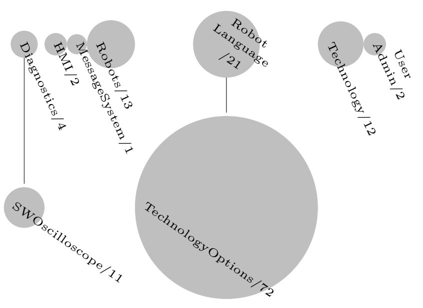
We further provide bubble tree diagrams visualizing the size of feature models for the different modeling spaces and modeling levels for KeMotion and KePlast. For instance, Figure 3a representing KeMotion's configuration space comprises the high-level model Robots, the second-level models AuxiliaryAxes and CommonSettings, and third-level models covering configuration options for different robot types.

Modeling Spaces. The results show that for both KeMotion and KePlast features were modeled in all three modeling spaces. The configuration space models (cf. Figure 3) define configuration decisions in the KeMotion and KePlast configurators, reflecting Keba's staged configuration process [18]. For instance, KeMotion's configuration space models for different robot types (e.g., Tripod, or SemiScara) eliminate configuration choices provided by GeneralRobot. KePlast's configuration space contains 9% mandatory features while KeMotion's configuration space contains 33% mandatory features. The higher number of mandatory features is caused by a number of core features reflecting characteristics of diverse robot types.

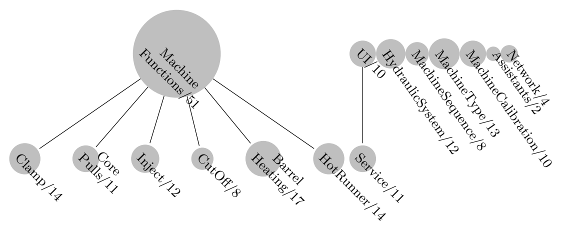
Figure 4 shows that some of the problem space models are quite large, reflecting the rich capabilities and operations of KeMotion's domain-specific language for programming robots (cf. TechnologyOptions and RobotLanguage) and KePlast's MachineFunctions. KePlast's top-level problem space models—e.g., MachineFunctions, MachineType, or HydraulicSystem—have been defined based on the KePlast product map. KePlast's problem space contains 22% mandatory features representing standard functionality. Optional features typically require an extra license. KeMotion's problem space mainly covers commands of KeMotion 's domain-specific language for programming robots.

TABLE I: KeMotion and KePlast model characteristics.

Characteristic	KeMotion	KePlast
features	395	454
mandatory	181	77
optional	154	212
alternative	60	165
configuration space	120	140
problem space	138	199
solution space	137	115
collections	48	52
components	5	5
inter-space dependencies	29	40
intra-space dependencies	5	38



(a) KeMotion



(b) KePlast

Fig. 4: Problem space feature models. Top-level models define a high-level system capabilities while lower-level models address detailed system characteristics.

It contains 38% mandatory features reflecting standardized commands.

The solution space models of both KeMotion and KePlast include several smaller feature models with less than 30 features, thus reflecting the modular design of the applications (cf. Figure 5). KeMotion's solution space contains 36% optional features defining capabilities of the robot programming language. Specific robot commands used in end-user programs are activated only during load-time, thus the instruction set was modeled as optional features whose variability is bound at load time. KePlast's solution space on the other hand contains 82% optional features, most of them in the HMI.KVB and HMI.KVS visualization systems. The inclusion of optional visualization system features often depends on configuration space features. For instance, the feature CalibCavPrSens13 reflecting the user interface for visualizing up to three cavity pressure sensors is included depending on the configuration space feature MoldCavityPressureSensor.

Modeling Levels. Both models comprise around 50 collections and components, which establish a hierarchy of feature models. Collections have been used frequently for structuring the models with a nesting level ranging between two and five. However, the solution space models are an initial attempt to create feature-based abstractions of the source code, and further refactoring of the larger models will likely increase their depth. For instance, larger collections like TechnologyOptions, RobotLanguage, or MachineFunctions will possibly be re-modularized by extracting feature collections in separate feature models.

Modeling Dependencies. Although revealing inter-space dependencies was not our primary goal when creating the models, our experiences show a lack of explicit knowledge about feature dependencies. The author creating the models added commonly known constraints. For instance, cross-tree constraints were defined in KePlast's configuration space after analyzing KePlast's custom-developed configurator. The KeMotion and KePlast models comprise 69 inter-space dependencies of different types—7 mapped_to, 33 implemented_by, and 29 configured_by, a first attempt for documenting relations between features in different spaces.

B. *RQ2 – Specific Model Space Characteristics*

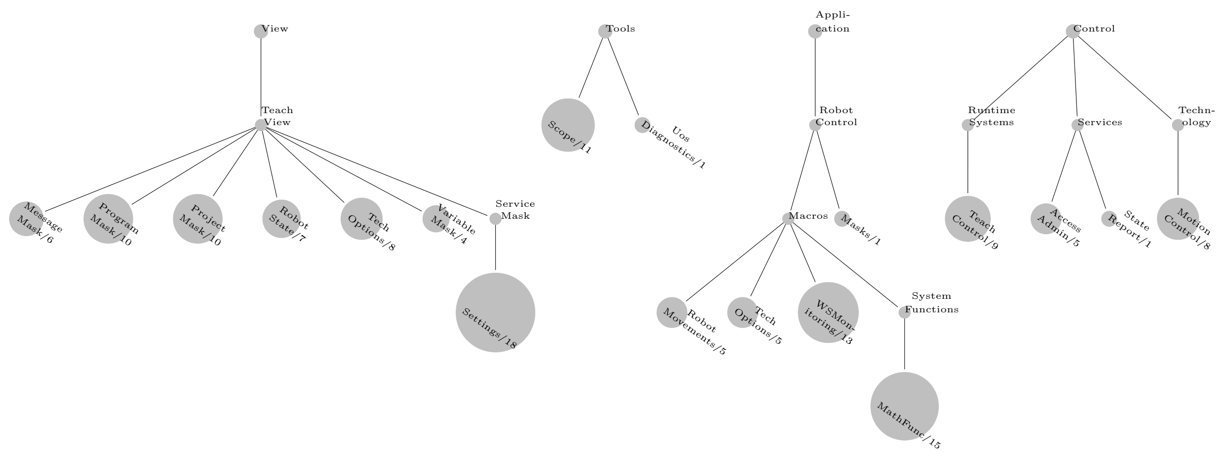
We report detailed characteristics about KePlast's problem and configuration space feature models, which are based on product maps and the custom-developed configurator, i.e., artifacts of high maturity.

TABLE II: Model characteristics for KePlast problem and configuration space models.

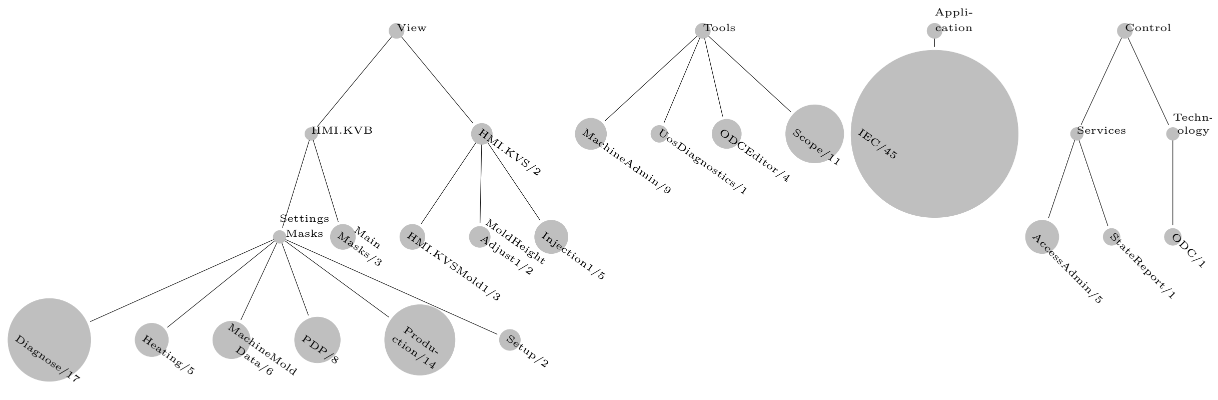
Characteristic	KePlast CS	KePlast PS
features	140	199
mandatory	12	44
optional	58	60
alternative	70	95
avg features per collection	17.5	13.3
maximum depth of leaf features	6	5
inter-space dependencies	6	34
intra-space dependencies	36	1
features with cardinality	13	-

Table II summarizes the results. The maximum depth of leaf features considers both the depth of a feature model and the level of the modeling space, i.e., the depth is increased with the number of hierarchically nested collections above a specific feature model. The maximum depth is 6 for the configuration space and 5 for the problem space. Examples for configuration space feature models with a maximum depth are Actuation, ClosureUnit, ActuationSetup, and SystemOptions. Configuring KePlast requires high domain expertise. For instance, features modeled in the SystemOptions feature model (cf. Figure 3b) are often related to the specific hardware equipment of an injection molding machine.

The six exemplary inter-space dependencies modeled for KePlast's configuration space are of type mapped_to. These dependencies link configuration space features (e.g., FastCloseValve) with solution space features (e.g., Mold1FastClose). The intra-space dependencies (i.e., cross-tree constraints) are also available in the custom-developed configurator, however, could only be revealed by inspecting the tool's source code. Most of these constraints are related to an injection molding machine's actuation type (e.g., electrical or hydraulic). The problem space models comprise a more complete set of inter-space dependencies of types configured_by and implemented_by:



(a) KeMotion



(b) KePlast

Fig. 5: Solution space feature models. Top-level elements are higher-level system functions and collections for organizing the model. Low-level elements are fine-grained features and configuration settings.

18 of these dependencies are related to the feature model MachineFunctions, 4 are related to MachineSequence, 8 are related to MachineType, and 4 are related to UI.

Features with cardinality are especially relevant in KePlast's configuration space models. An example for a feature with cardinality is the MoldCavityPressureSensor, allowing for up to 4 sensors measuring cavity pressures.

C. Threats to Validity

As with any empirical research, our results may not generalize beyond the cases we considered. There is a potential bias caused by the selection of KeMotion and KePlast, as they are both from the industrial automation domain. However, the systems are from two different areas (i.e., injection molding and robotics). We also try to avoid generalizations and present a detailed analysis of the models we created. We can only present descriptive model metrics and cannot claim statistical significance. However, given that companies typically do not provide access to data about their systems we believe that our results are valuable to other researchers and practitioners.

As the second author was in charge of creating the variability models, it can be argued that the results are solely due to our manipulations. However, the modeler adhered to product maps, specification documents, custom-developed configurators, and the code base, mature artifacts created and maintained by diverse domain experts without any influence from our side. We further attempted to mitigate this threat by performing an iterative modeling process with feedback and validation based on prototypes of the models.

VI. Lessons Learned

We report observations and lessons learned we made when modeling and validating the feature models:

Be specific about the purpose and level of features to facilitate the modeling process. Our modeling approach assumes classifying features by their purpose to better understand their role in the system. The approach further allows defining features at different levels of granularity. The discussed feature models comprise more than 100 structural elements in all modeling spaces, thus indicating the need and usefulness of modularizing feature models in such a divide-and-conquer manner. The modularization eases involving modelers with different background, as they can focus on their area of expertise, i.e., product management, architecture, or product configuration aspects. Our results further show that detailed domain expertise is required for defining the feature models, regardless of the modeling space. Typically, models could only be created after detailed inspections of other artifacts such as product maps, models, or source code, as we did regarding RQ2. The results also show that the feature models are of manageable size and a modeling tool with specific support for creating views was not needed.

Focus on the dependencies between feature models to develop a system-wide perspective. The feature models cannot be defined in isolation and understanding their dependencies is fundamental in a feature-oriented development process. However, revealing and understanding the dependencies between features from different models turned out to be extremely challenging as can be seen by the rather low numbers of dependencies. Our observation is supported by Berger et al. [30], who found that modelers in industry focused on building the parent-child relationship between features, while trying to avoid cross-tree constraints. It has been pointed out that modeling dependencies would be very helpful, e.g., to reveal the implementation of high-level features in the code base, or check consistency during product derivation [31]. However, while providing modeling support for dependencies is easy, revealing actual dependencies between modeling spaces, and people working with features in these spaces, is much harder. Again, involving different roles is useful: software engineers in charge of a component can define its solution space features, while system architects can define dependencies between different feature models.

Provide code-level views on the features. Features in a feature diagram are just a label and engineers want to know how the features manifest themselves in the underlying architecture and code base. Views based on feature-to-code mappings are therefore particularly important. For instance, features can be linked to program variables, which represent an initial seed of a feature implementation. Code-level views on features can then be provided by computing slices based on program analysis techniques that follow the control and data dependencies. For instance, Angerer et al. have presented an approach for automatically computing feature slices [26], which can help to validate manually created solution space feature models as shown in [32]. In this way, the feature models also provide a starting point for moving towards feature-oriented software development (FOSD), a programming paradigm for managing the construction, customization and synthesis of software systems based on features as first-class citizens [33].

Feature models help to limit variability. Feature models have originally been proposed to elicit and represent variability *and* commonalities of systems' capabilities [7]. Feature models show explicitly what is *not variable* and which product variants are *not possible*. Keba uses a wide range of variability mechanisms such as interfaces to hook in new functionality; load-time and run-time activation of modules; configuration parameters to influence program behavior; or pre-processing of code. Feature models define a variability interface to components and provide a way to control the otherwise unlimited flexibility, thus improving guidance for developers.

VII. Related Work

Variability modeling is a core activity in software product line engineering (SPLE) [34] and a wide range of variability modeling approaches have been proposed, including feature modeling [1], decision modeling [35], and orthogonal variability management [3]. We discuss existing case studies on variability modeling, research on modularization, multi product lines and megamodels, as well as approaches for modeling dependencies between modeling spaces.

Case studies on feature modeling in practice. Some empirical studies exist on applying feature modeling in practice, however, only few reports exist on variability modeling in large-scale systems. For instance, Berger et al. [30] provide a detailed analysis of features in 128 variability models including detailed metrics about features types, numbers of features, and feature dependencies. Lee et al. [36] report detailed modeling experiences related to an elevator control software product line comprising 490 features – 157 capability, 22 operating environment, 291 domain technology, and 20 implementation technique features. The feature spaces used by Lee et al., originally proposed in [7], are related to the modeling spaces we used in our approach: Capabilities are addressed by configuration space and problem space features. Domain technologies are reflected by solution space features, however, some problem space features also address domain technologies. The operating environment is related to configuration space features representing specific hardware equipment of an injection molding machine or a robot. Developers are concerned about specific implementation techniques, which are covered by solution space features.

A recently conducted case study provides an in-depth analysis of 23 features in real-world settings based on interviews investigating the practical use of features in three large companies [4]. The authors use feature facets for describing and comparing features. Some of the facets are related to the issues investigated in our paper. For instance, the facet *use* relates to the purpose of feature models and the *position in hierarchy* is related to the modeling level. Although the aims of the studies are different, they complement each other: while Berger et al. discuss rationales for classifying individual features as typical, outlier, good, or bad based on the use of features in practice, the aim of our study is to explore multi-purpose, multi-level feature models.

Modularization and Multi product lines. MultiDeltaJ represents a holistic multi product line modeling approach covering problem, solution and configuration space [37]. The approach aims at obtaining multi product lines by fine-grained reuse of delta-oriented product lines. Kästner et al. propose a variability-aware module system, enabling a divide-and-conquer strategy to software development and breaking with the anti-modular tradition of a global variability model in product-line development [10]. Modules are considered as product lines, which can be type checked in isolation, however, variability can crosscut multiple modules. Dhungana et al. present an approach that aims at reducing the maintenance effort of modeling product lines by organizing the modeling space as a set of interrelated model fragments defining the variability of particular parts of the system [8]. Our approach also aims at modularizing feature models, to support the distributed development and modeling of components. Holl et al. support multiple users in performing distributed product derivation of a multi product line by sharing configuration information [21]. The approach emphasizes configuration space features and their dependencies, but can be helpful in establishing a system-wide perspective. The Common Variability Language (CVL) [24] is a domain-independent language proposal for specifying and resolving variability. It facilitates the specification and resolution of variability over any instance of a Meta-Object Facility (MOF)-based language, which is termed a base model. Configurable units are an integral part of CVL and are used for grouping associated variation points.

Mega modeling. Bézivin et al. have recognized the need for global model management using megamodels, i.e., composites of interrelated models and meta models for describing large-scale systems [38], [39]. Megamodels consider models as first-class citizens and relevant dependencies are, for instance, the conformance relation between a model and its meta model. The Atlas Mega Model Management approach (AM3) provides practical support for developing megamodels [40]. Similarly, Salay et al. introduce macromodels for managing multiple models at a high level of abstraction expressed in terms of models and their intended relationships [41]. Seibel et al. present dynamic hierarchical megamodels combining traceability and global management [42]. Another topic of interest in multi-modeling is checking model consistency. Denton et al. present the NAOMI platform for managing multiple models developed in different modeling languages [43]. The approach analyzes dependencies to determine the impact of changes on dependent models and to propagate changes. As our results show the feature collections and components in our approach can be seen as individual models used for defining features of large-scale systems. However, we do not manage dependencies between collections, but between individual features in different collections.

Modeling of dependencies. Many approaches emphasize modeling dependencies between different modeling spaces. For instance, FeatureMapper and VML* support modeling the relationship between problem space features and solution space models describing product line details (e.g., requirements models, architecture and design models) [20]. However, these approaches do not take configuration space features into account, which comprise around 30% of features in both KeMotion and KePlast feature models. The COVAMOF [23] framework models variability in terms of variation points and variability dependencies at different levels abstraction (i.e., features, architecture, and implementation). COVAMOF uses realization relations for providing a hierarchical organization of variation points. In contrast, our approach supports nesting feature models to build hierarchical models. Furthermore, COVAMOF's dependencies focus on guiding and restricting the selection of variation points during product derivation. Dependencies are also important in multi-level feature trees, an add-on to traditional feature models that introduce the notion of reference feature models, which serve as a template and guideline for the referring model [9]. The reference model becomes a means to strategically drive the content of the referring model by allowing or disallowing certain deviations. Locally introduced innovations can be made globally visible in a step-by-step process.

VIII. Conclusion and Future Work

This paper presented experiences of applying a multi-purpose, multi-level feature modeling approach to two large-scale industrial automation systems. We extended the FeatureIDE tool suite to support feature models for different purposes, at multiple levels, as well as dependencies between features from different models. Regarding RQ1 we conclude that our approach was feasible and useful in the context of two industrial systems. Regarding RQ2 we reported detailed characteristics on the size and scope of models. Our lessons learned show that considering the purpose and level of features is useful, that understanding dependencies between feature models is essential for developing a system-wide perspective, that code-level views and domain dictionaries are important to understand the meaning of features, and that feature models help to limit otherwise boundless variability. However, there is still a need for further experience reports of industrial organizations moving towards feature-oriented software development processes and introducing feature modeling in industry.

We are currently extending the tool suite to allow cloning of feature models. Such feature model clones are needed in a distributed clone-and-own development process to manage customer-specific product variants and product line extensions [44]. We also plan on refining our FeatureIDE extensions regarding support for feature-oriented and role-specific views based on the introduced modeling spaces.

Acknowledgment

This work has been conducted in cooperation with KEBA AG, Austria, and was supported by the Christian Doppler Forschungsgesellschaft, Austria.

References

[1] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Tech. Rep., 1990.

[2] F. J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action*, 2007.

[3] K. Pohl, G. Böckle, and F. Van Der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer-Verlag, 2005.

[4] 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 *Proc. SPLC*, 2015.

[5] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski, "Cool features and tough decisions: A comparison of variability modeling approaches," in *Proc. VaMoS*, 2012.

[6] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[7] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Ann. Softw. Eng.*, vol. 5, pp. 143–168, Jan. 1998.

[8] 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.

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

[10] C. Kästner, K. Ostermann, and S. Erdweg, "A variability-aware module system," in *Proc. OOPSLA*, 2012.

[11] 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.

[12] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "FeatureIDE: An extensible framework for feature-oriented software development," *Sci. Comput. Program.*, vol. 79, pp. 70–85, 2014.

[13] D. Lettner, F. Angerer, H. Prähofer, and P. Grünbacher, "A Case Study on Software Ecosystem Characteristics in Industrial Automation Software," in *Proc. ICSSP*, 2014.

[14] D. Lettner, F. Angerer, P. Grünbacher, and H. Prähofer, "Software Evolution in an Industrial Automation Ecosystem: An Exploratory Study," in *Proc. SEAA*, 2014.

[15] D. Lettner, M. Petruzelka, R. Rabiser, F. Angerer, H. Prähofer, and P. Grünbacher, "Custom-developed vs. model-based configuration tools: Experiences from an industrial automation ecosystem," in *Proc. MAPLE/SCALE Workshop at SPLC*, 2013.

[16] R. A. Malloy, "Plastic part design for injection molding," in *Plastic Part Design (2nd Edition)*. Hanser, 2010.

[17] D. Rosato, M. Rosato, and D. Rosato, *Injection Molding Handbook (3rd ed.)*. Kluwer Academic Publishers, 2000.

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

[19] D. Dhungana, P. Grünbacher, and R. Rabiser, "The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study," *Automated Software Engineering*, vol. 18, no. 1, pp. 77–114, 2011.

[20] F. Heidenreich, P. Sanchez, J. Santos, S. Zschaler, M. Alferez, J. Araujo, L. Fuentes, A. Kulesza, Uiraand Moreira, and A. Rashid, "Relating feature models to other models of a software product line," in *Transactions on Aspect-Oriented Software Development VII*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6210.

[21] G. Holl, P. Grünbacher, C. Elsner, and T. Klambauer, "Supporting awareness during collaborative and distributed configuration of multi product lines," in *Proce. APSEC*, 2012.

[22] J. A. Galindo, D. Dhungana, R. Rabiser, D. Benavides, G. Botterweck, and P. Grünbacher, "Supporting distributed product configuration by integrating heterogeneous variability modeling approaches," *Information and Software Technology*, vol. 62, pp. 78–100, 2015.

[23] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "COVAMOF: A framework for modeling variability in software product families," in *Software Product Lines*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 3154.

[24] CVL, "Common variability language," 2012, http://www.omgwiki.org/variability/doku.php?id=start&rev=1351084099, [Online; accessed 27-April-2015].

[25] D. Batory, "Feature models, grammars, and propositional formulas," in *Software Product Lines*, ser. Lecture Notes in Computer Science, H. Obbink and K. Pohl, Eds., 2005, vol. 3714.

[26] F. Angerer, H. Prähofer, D. Lettner, A. Grimmer, and P. Grünbacher, "Identifying inactive code in product lines with configuration-aware system dependence graphs," in *Proc. SPLC*, 2014.

[27] SINTEF (MOD research group), "CVL 2 Tool," 2013, http://modelbased.net/tools/cvl-2-tool/, [Online; accessed 30-April-2015].

[28] K. Lee, K. Kang, and J. Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," in *Proc. ICST*, ser. LNCS, vol. 2319, 2002.

[29] T. Berger and J. Guo, "Towards system analysis with variability model metrics," in *Proc. VaMoS*, 2014.

[30] T. Berger, D. Nair, R. Rublack, J. M. Atlee, K. Czarnecki, and A. Wąsowski, "Three Cases of Feature-Based Variability Modeling in Industry," in *Proc. MODELS*, 2014.

[31] K. Nie, T. Yue, S. Ali, L. Zhang, and Z. Fan, "Constraints: The core of supporting automated product configuration of cyber-physical systems," in *Model-Driven Engineering Languages and Systems*, ser. LNCS. Springer Berlin Heidelberg, 2013, vol. 8107.

[32] 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 *Proc. ICSME*, 2014.

[33] S. Apel and C. Kästner, "An Overview of Feature-Oriented Software Development," *J. Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[34] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[35] K. Schmid, R. Rabiser, and P. Grünbacher, "A Comparison of Decision Modeling Approaches in Product Lines," in *Proc. VaMoS*, 2011.

[36] K. Lee, K. C. Kang, E. Koh, W. Chae, B. Kim, and B. W. Choi, "Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice," in *Proc. of the First International Conference on Software Product Lines: Experience and Research Directions*, 2000.

[37] F. Damiani, I. Schaefer, and T. Winkelmann, "Delta-oriented multi software product lines," in *Proc. SPLC*, 2014.

[38] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," *Proc. OOPSLA/GPCE*, 2004.

[39] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez, "Modeling in the large and modeling in the small," in *Model Driven Architecture*. Springer Berlin Heidelberg, 2005, pp. 33–46.

[40] F. Allilaire, J. Bézivin, H. Brunelière, and F. Jouault, "Global model management in eclipse GMT/AM3," 2006, http://www.emn.fr/z-info/atlanmod/index.php/Global_Model_Management, [Online; accessed July-2015].

[41] R. Salay, J. Mylopoulos, and S. Easterbrook, "Managing models through macromodeling," in *Proc. ASE*, 2008.

[42] A. Seibel, S. Neumann, and H. Giese, "Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance," *Software & Systems Modeling*, vol. 9, no. 4, pp. 493–528, 2010.

[43] T. Denton, E. Jones, S. Srinivasan, K. Owens, and R. Buskens, "NAOMI – an experimental platform for multi–modeling," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Springer Berlin Heidelberg, 2008, vol. 5301, pp. 143–157.

[44] D. Lettner and P. Grünbacher, "Using feature feeds to improve developer awareness in software ecosystem evolution," in *Proc. VaMoS*, 2015.