Combined Variability Management of Business Processes and Software Architectures

Andreas Daniel Sinnhofer[1], Andrea Höller[1], Peter Pühringer, Klaus Potzmader[2], Clemens Orthacker[2], Christian Steger[1] and Christian Kreiner[1]

[1]*Institute of Technical Informatics, Graz University of Technology, Austria*

[2]*NXP Semiconductors, Gratkorn, Austria*

{*a.sinnhofer,andrea.hoeller,christian.kreiner,steger*}*@tugraz.at, p.puehringer@inode.at,*

{*klaus.potzmader, clemens.orthacker*}*@nxp.com*

Keywords: Software Product Lines, Feature Oriented Modeling, Business Process Variability Management, Software Configuration

Abstract: Nowadays, organizations are faced with the challenge of surviving in a highly flexible and competitive market. Especially the domain of Internet of Things is affected by short product cycles and high pricing pressure. Business Process oriented organizations have proven to perform better regarding highly flexible markets and fast production cycles. However, especially industries focused on low cost IoT systems are facing big problems if the according business processes are not aligned with the business processes. Consequently, a lot of development effort is spent for features which are never addressed by any business goal. With this work, we propose to use a combined variability management in order to efficiently address product variability on the organizational level as well as on the technical level.

1 INTRODUCTION

We are living in an ever changing and interconnected world. The dawn of the Internet of Things (IoT) further increases the trend for organizations to deliver feature rich systems in high quantities and at low costs. Due to this pricing pressure, methods have to be investigated which allow modular and highly configurable systems such that the products can be adapted to the current requirements of the market.

Business Process (BP) oriented organizations are known to perform better regarding highly flexible demands of the market and fast production cycles (McCormack and Johnson (2000); Hammer and Champy (1993); Valena et al. (2013); Willaert et al. (2007)). This is achieved by introducing a management process during which business processes are modeled, analyzed and optimized in iterative improvement processes. During recent years, business process management is further coupled with a workflow management in order to monitor the correct execution of the process and to integrate responsibilities to the process models. In order to react to changing requirements, context aware business process modeling techniques were introduced by Saidani and Nurcan (2007): Flexibility is gained through the analysis of the context states of the environment which are mapped to the according business processes and their related software systems. The problem with such approaches is that the used software systems are often developed independently from each other, although they share a similar software architecture.

Software Product Lines (SPL) have proven to be essential for the development of flexible product architectures which can be adapted to the current requirements (Pohl et al. (2005)). Through the use of a common architecture and reusable product features, SPL promises to deliver high quality products while simultaneously maintaining low development costs. The most critical phase during the design and the implementation of a product line is the identification of the variable parts and the common parts of the product family (Pohl et al. (2005)). Consequently, a lot of effort is invested to identify the domain requirements of the final product portfolio. Equally important is the selection of the according features during the application engineering: It has to be guaranteed, that the customer requirements are fully met; further, all unnecessary features need to be excluded in order to ensure low productions costs of the final product. Since the identification of the domain requirements is usually carried out from developers, an integrated view of the

organizational goals is often missing. Thus, the efficiency of the product line is reduced since additional effort needs to be invested to configure the product according to the current requirements.

This work focuses on the development of a framework which aims to enforce a link between the variability of the business processes and the variability of the product platform. As such, we propose a combined variability modeling in which the requirements for the organization as well as for the development of the product platform are identified together. After identifying the requirements, order processes are designed which reflect the possible product configurations that can be ordered by a customer. These variable order processes are further used to automatically trigger the product customization process in order to reduce the production costs of the final product. This work is based on our previous works in which we already defined systems for the modeling variability of business process models (Sinnhofer et al. (2015)) as well as a framework for generating software configurations based on order processes (Sinnhofer et al. (2016, 2017)).

This work is structured in the following way: Section 2 summarizes basic concepts about business process modeling as well as software product line engineering. Section 3 summarizes out approach to link variable order process models to variable software architectures in an automatic way. In Section 4 we describe how we applied the introduced concepts in an industrial use case and present a simplified example for illustration purposes. Since the identification of business drivers is essential for an organization to survive in a competitive market, we show in Section 5 how we were able to identify improvement opportunities, by analyzing the results of our framework. We conclude this work by presenting related work in Section 6 and a summary in Section 7.

2 BACKGROUND

The current section summarizes the basic concepts of Software Product Line Engineering and Business Process Modeling, which are applied in this work. Further, our previous publications – which are forming the foundation of this work – are briefly summarized.

2.1 Software Product Line Engineering

Software Product Line Engineering (SPLE) applies the concept of product lines to software products. As a consequence, SPLE promises to create diverse and high quality software products of a product family in short time and at low costs Pohl et al. (2005). Instead of writing software for every individual system, software products are automatically generated by combining the required domain artifacts. The principal concept can be split into two main phases: the Domain Engineering and the Application Engineering (Pohl et al. (2005); Weiss and Lai (1999)).

The Domain Engineering is the phase in which the variabilities and the commonalities of the according domain are identified and the reusable domain artifacts are implemented. Domain artifacts are development artifacts like the software architecture or the software components. One essential phase during the domain engineering is the requirements engineering process, in which a domain analysis has to be performed in order to identify the requirements of the final product. Based on the identified requirements, the domain is usually modeled by using a Feature Oriented Design Modeling (Kang et al. (1990)) approach. During this process, Feature Models are used to explicitly describe all features of a product, their relationships, dependencies and additional restrictions.

The Application Engineering is the phase during which the final products are created by combining the domain artifacts in a meaningful manner. This is enforced by the use of domain constraints which were modeled during the Domain Engineering phase. In difference to the Domain Engineering, the Application Engineering is mainly focused on reusing artifacts rather than the implementation of new artifacts. In the ideal case, this phase makes use of software generators to automatically derive product variants without the need of implementing any new logic. The amount of reused domain artifacts heavily depends on the application requirements and gives an estimate on the efficiency of the product line. Hence a major concern of the application engineering is the detection of deltas between the application requirements and the available capabilities of the product line.

2.2 Business Process Modeling

Business Processes (BP) are a specific sequence of activities or (sub-) processes which are executed in a certain order to create an amount of value to the customer Hammer and Champy (1993). In this work, we use the concept defined by Österle (1995) to model BPs: BPs are modeled in different layers, where the top level (macroscopic level) is a highly abstract description of the overall process and the lower-levels (microscopic levels) are more detailed descriptions of the sub-processes. A reasonable level of detail is reached, if the process description on the lowest levels can be used as work-instructions for the responsi-

ble employees. This leads to the fact that the higher levels of the process description are usually independent of the production facility and the supply chains; while the lower levels are highly dependent on the production facility and its capabilities. As a consequence, the macroscopic level is more stable with respect to changes and can be reused in different contexts and production environments. The microscopic levels need to be updated in order to reuse them in different contexts. Variability of such process structures can be modeled through a variable process structure (i.e. by adding/removing activities in a process) or by replacing process refinements with different subprocesses. In general, three main types of business processes can be distinguished (see Association of Business Process Management Professionals (2009)):

- **Primary Processes**: Each of the process activities adds a specific amount of value to the value chain. Consequently, such processes are also often referred to as Core Processes since the customer value is directly reflected in these processes.
- **Support Processes**: Are processes which are designed to support the Primary Processes like managing resources or infrastructure. Such processes do not directly add value to the customer but are essential to ensure the proper execution of the Primary Processes.
- **Management Processes**: Are designed to monitor and schedule business activities like the execution of Primary Processes or Support Processes. While Management Processes do not directly add value to the customer, they are designed to increase the efficiency of the business activities.

Domain specific modeling languages are usually used to model all the activities, resources and responsibilities within a Business Process. In the scope of this work, the Business Process Model and Notation (BPMN, Object Management Group (2011)) is used to model processes, but the general concept of this work is not limited to this notation. The key concepts which are used in this work, are summarized below (Object Management Group (2011); Sinnhofer et al. (2017)):

Events: Occurs during the execution of a process and may affect the flow of the process. For example, the start or the completion of an Activity are typical events that occur in every process. According to the BPMN specification Object Management Group (2011), events are used only for those types, which affect the sequence or timing of activities of a process.

Activities: An Activity is a specific amount of work that has to be performed by the company – or another organization – during the execution of a process. Two different types of activities can be distinguished: Atomic activities (i.e. a task) and non-atomic activities (e.g. sub-processes).

Gateways: Are used to control how the process flows through different sequences of activities. Each gateway can have multiple input and/or output paths. One example is a decision, where out of many possibilities, only one path is selected. The selection of the paths can be coupled to conditions or events which are triggered during the execution of the process.

Data: Data objects represent the information flow through the process. Two types of Data objects can be distinguished: Input Data that is required to start a specific activity and Output Data which is produced after the completion of an Activity.

Pool and **Lanes**: Are used to model responsibilities for specific activities in a process. Responsibilities can be usually assigned to an organization, to specific roles or even dedicated employees.

It is common practice for organizations to maintain multiple variants of business processes which are based on a common template (Rosa et al. (2017)). This leads to the situation that similar process variants are created through a copy and clone strategy. As a consequence, maintaining these process variants is a time consuming tasks since every single process variant has to be manually updated by the according process designer. Besides the additional maintenance effort, using copy and clone strategies also have a negative influence on the process documentation. To solve these issues, we proposed a Software Product Line approach for the derivation of process variants from business process models (see Sinnhofer et al. (2015, 2016, 2017)). The concept can be split into four different phases:

Process modeling: During the process modeling, process designers are responsible to design process templates. The process templates are designed using the BPMN notation and additional artifacts are integrated like documentation templates, responsible roles, resource allocations, etc. The process templates are designed in an appropriate BPM Tool to fully support the process designers during the design process. The process of designing the process templates and the process of creating the according domain model goes hand in hand to ensure that the created templates can be reused in many different contexts.

Domain modeling: During this process, the created templates are imported into a Software Product Line tool and translated into a so called feature model (see Sub-Section 2.1). During the creation of the feature model, it has to be decided which parts of the process are designed to be variable and which parts are

static. For illustration purposes, the following example is given: A company creates car parts for two major car manufacturers. While the overall process for creating the car parts is identical for both customers, different production planning strategies are used to optimize the material usage (e.g. stock size, etc.). As a consequence, the production planning strategy has to be designed variable such that the overall process model can be reused for both customers. The definition of variable parts and static parts happens in close cooperation with the according process designers and may even lead to a re-iteration of the first phase if some process templates need to be adapted. Not every combination of variants may create meaningful process variants. As a result, a comprehensive list of restrictions and rules has to be designed as well to guarantee that only valid and meaningful process variants can be created by the product line. The list of rules and restrictions has to be defined flexible as well, since not every restriction may be identified when the process model is created. Consequently, re-iterations of the restriction model are common after collecting evaluation data from the execution of the process.

Feature selection: Based on the current requirement of the organization, process variants are created using the created feature model. This is done be selecting the required features from the model and by translating this feature selection to a valid business process structure. To ensure an automatic transformation, generators have to be developed which are able to translate between the business process model and the feature model. The defined rules and restrictions are enforced during this process to guide the domain expert in selecting a meaningful set of options. To continue the example from above, two process variants may be created for the two customers. The only difference between the processes is the production planning strategy.

Maintenance and Evolution: One of the most important phases is the maintenance and evolution phase: To be highly flexible and adaptive to the current requirements of the market, processes and their according models have to be continuously improved and adapted. As such, the derived processes are monitored by production experts during the time in use and evaluated against the requirements. Based on the collected data, feature selections can be improved or process improvement processes can be scheduled. During a process improvement process, process templates are updated or created from the process designers and integrated into the existing feature model. Through the capabilities of the Software Product Line tool, it is possible to automatically propagate the changes of the process templates to every instance. As a consequence, no time consuming and error prone manual maintenance process is necessary to adapt all the existing process variants. Since it may happen that some of the process variants shall not be updated in case of changes, version control can be used to explicitly state which version of a template shall be used.

Our today's business environment is focused on creating sustainable value by increasing the revenue of business drivers. The identification of such business drivers, or the identification of the drivers which are able to destroy value is an essential step for an organization Strnadl (2006). Otherwise, staying competitive or even survive on a flexible market is not possible. The combination of business variability and software variability is a promising way to improve the identification of such drivers. Further, having a combined view of the requirements helps to increase the overall efficiency of the product line.

3 COMBINED VARIABILITY MODELING

The goal of this work is an automatic generation of software products based on the product order. In order to achieve this goal, an integrated view is necessary in which the variability of the software product is reflected in a variable order process. Since only a few configuration options are usually exposed to the end-customer, also all internal processes need to be covered in the variability management process. The overall concept of the resulting process is highlighted in Figure 1. As illustrated, the Process Variability Framework – which was described in Sub-Section 2.2 – is used as a foundation to model variable order process models. Based on this order process models, order entry forms are automatically generated which need to be filled by internal or external customers. Based on the provided data, a product line is triggered which maps the provided order data to the customization options of the software product. Based on the generated feature mapping, the final product can be automatically derived without any manual step beside verification steps which may be required by certification requirements. In order to achieve a binding between the order process models and the generated order entry forms, the following type model was introduced (Sinnhofer et al. (2017)):

- **None**: No special data needs to be submitted. Thus, a process node marked with none will not appear as a setting in the order entry form.
- **Inputs**: This is the abstract concept for different input types which are described below. Each

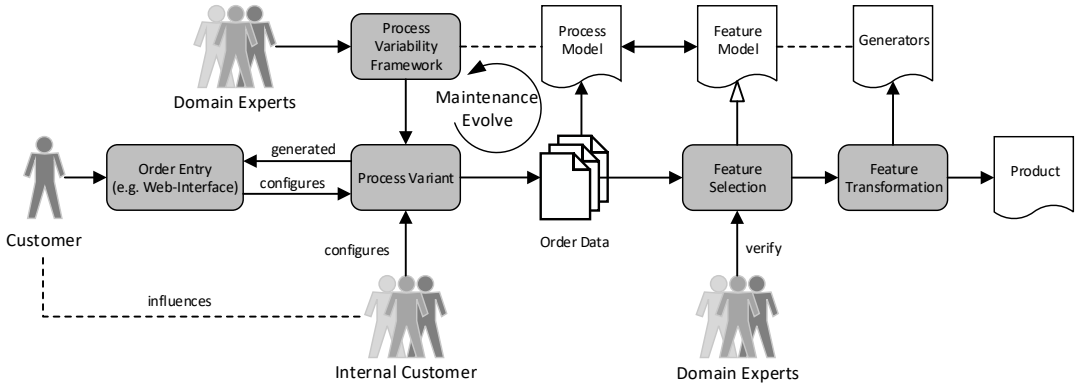

Figure 1: Overview of the concept for combining order process variability and software variability to automatically derive software products.

Input is mapped to a specific input type, defining the format of the input. For example, input data could be delivered in form of a file, or configuration settings could be delivered in form of strings or integer values. Depending on the applied domain, also non functional properties may need to be modeled in the Input type. For example, if a security critical product is developed, a customer may be asked to provide a cryptographic key which is used to authenticate the customer to the device. Besides providing this key, also some kind of specification is required in which format this data was provided (e.g. pgp encrypted, etc).

- **Customer Input**: Specific data that has to be added from an external customer. A process activity marked with this type will generate an entry in the order entry form of a specific type. For example, drop down list will appear if a customer can select between different options.
- **Internal Input**: Specific data needs to be added from an internal stakeholder. A process activity marked with this type will not generate an entry in the external customer order interface, but will create a separate order entry for the according internal stakeholder.
- **Input Group**: A set of inputs which are logically linked together. As a consequence, all of these inputs will be highlighted as a group in the generated order entry and all of them are required for a single customization feature of the final software product.

The information type has to be added to the process feature model of the SPLE tool. To support the domain experts in creating the according mappings, the following rules are automatically applied by the SPLE tool based on the BPMN types (Sinnhofer et al. (2017)):

- **Activities**: Non-atomic activities are used to group specific sets of input parameter to a single feature. For example, a process designed for customizing an application may require several input parameter (like user name, password, license files, etc.). As a consequence, non-atomic activities will appear as an Input Group for all inputs defined by the according sub-process(es). Any atomic activity will be automatically tagged as input type ”None”. The input type ”None” is also automatically applied if a non-atomic activity does not contain any data. Consequently, ”empty” non-atomic activities will also not appear in the generated order entry form.
- **Gateways**: Are used to define the structure of the generated form. For example, for a decision node, a drop down selection will appear such that the customer can choose between different customization paths. For decisions it is further enforced that the customer can only select and submit the data for one single path.
- **Data**: Data is to be provided by any entity involved in the process(es). With respect to our case study, ”String” turned out to be a meaningful default value.
- **Pool** and **Lanes**: Are used to define the source of the input data. For example, a data node which is part of a company internal lane will automatically be tagged as an ”Internal Input”, while Data in an external lane will be marked as ”External Input”. ”Internal Input” should be used as a default value to circumvent accidental exposure of internal configuration settings to the end-customer if Pool and Lanes are not used.

All default mapping rules can be manually overwritten by the Domain Expert during the creation of the process model. Changes to the process model

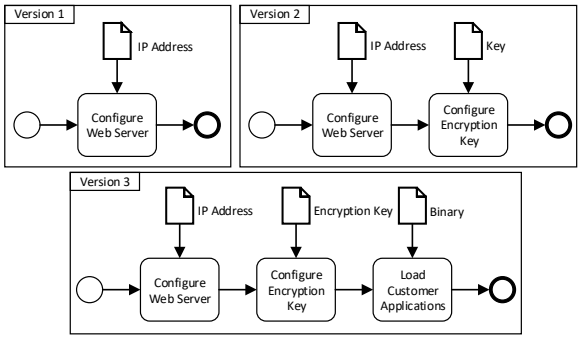

Figure 2: Exemplary order processes for the three different versions of the IoT device, based on (Sinnhofer et al. (2017)).

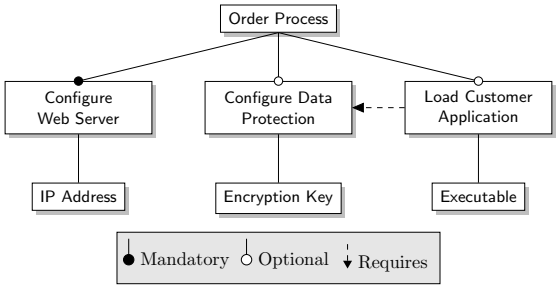

Figure 3: Exemplary feature model for the customization of the IoT device in three different flavors.

(e.g. adding/removing/changing activities) are traced via unique identifiers and illustrated as a diff-model such that changes can be reviewed by the Domain Experts. After the order process model was successfully tagged, the according order entry forms can be generated. With respect to this work, we have chosen web-based forms since they are commonly used in practice.

As illustrated in Figure 1, the provided customer data is used to create the feature selection of the final product. A manual verification step is advisable in order to ensure that no mistake was made during the development of the translation logic. Additionally, for certification purposes it may also need to be proven that a verification was done to ensure that no customer related data is confused with other products. In order to automatically select the features, the grouping information of the order entry is used to select the required features. After the selection was approved by the Domain Experts, it is automatically processed by the product specific code generators of the product line which utilizes the provided order data to actually generate the according product. The result of this process strongly depends on the use case: It could be a binary file that is loaded to the Integrated Circuits during the production, a configuration script which is executed on the final product, or any other approach. We will discuss a script based approach in Section 4 in more detail.

Especially for new types of products it is very likely that new knowledge is gained on how to increase the efficiency of the whole process(es). Only if changes to the generated order entry forms are necessary, a maintenance process for the product customization system is required. The maintenance costs for the product line can be kept as low as possible, since the code generators and model transformation logic only has to be updated once, but can be reused for the whole product line.

4 INDUSTRIAL CASE STUDY

For illustration purposes, we will discuss our industrial case study in more detail, showing how process models are translated and the final product is automatically derived. The implemented business processes of our industrial partner are controlled by an SAP infrastructure and are designed with the BPM-Tool Aeneis. Further, we are using the SPLE tool pure::variants to manage to variability of the business processes as well as the variability of the final product configurations. A more detailed description of the developed tools plugins can be found in our previous publications (see Sinnhofer et al. (2015, 2016)). For illustration purposes, we will consider the following – simplified – example: A company is developing small embedded systems which are used as sensing devices for the Internet of Things (IoT). The devices are sold to distributors (refered to as customer in the following) in high quantities which mean that the customization of the devices cannot be done by the customer in a feasible way. The device is offered in three different variants with the following features (based on Sinnhofer et al. (2017)):

Version 1: Senses the data in a given time interval and sends the recorded signal to a customer operated web-server which is used for post-processing the data. In the first version, the communication channel between the web-server and the device is unprotected. The customer is responsible for providing the connection string of the web-server to the company.

Version 2: Additionally to the basic features of the first version, this version allows encryption of the communication channel between the server and the node using symmetric encryption algorithms. For simplicity of this example, it can be assumed that the encryption key is provided by the customer, which has to be loaded to every single device. For simplicity, we assume that the key is sent in plain by the customer.

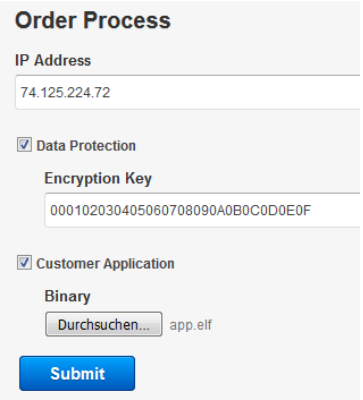

Figure 4: Exemplary order entry form that is automatically generated from the Feature Model highlighted in Figure 3.

Version 3: Additionally to the basic features of the first version, this version allows customer applications to be run on the system. This requires that the customer submits a binary file which is loaded to the device during the production.

Traditionally, this would result in three different order processes which are formed via a copy and clone strategy (see Figure 2): The order process of the first version is copied and extended for the second version, while the third version is an extended copy of the second version. This means that changes to the basic version would result in the maintenance of two other processes as well. Using our developed framework leads to the situation that all three process variants are derived from one common process model. As such, the same result is achieved like using a manual preparation, but the maintenance costs can be reduced essentially since all variants are automatically updated. The according Feature Model is illustrated in Figure 3. For illustration purposes, a ”requires” relationship between the web-server configuration and the data protection configuration is not highlighted since the ”Configure Web Server” feature is a mandatory feature. Consequently, the configuration is always part of the final product and does not need to be explicitly modeled.

To automatically generate the order entry form based on the feature model, the input data ”IP Address”, ”Encryption Key”, and ”Binary” has to be set to the according type. As such, rules can be defined to ensure that the given IP Address is formatted as a valid IPv4 or IPv6 IP Address, or that the encryption key has a meaningful length, etc. If no additional checks are implemented, the Domain Expert would only need to specify the input type of the ”Binary” to be a binary file. Doing so, the order entry form illustrated in Figure 4 can be generated, assuming that all input parameter are provided by the customer. After

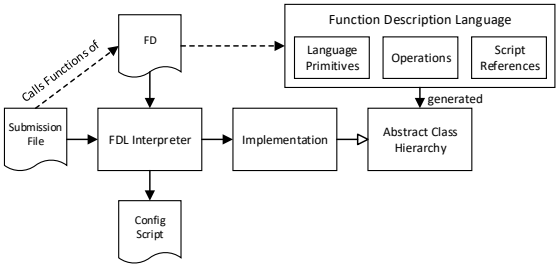

Figure 5: Framework for a flexible, runtime-configurable script generation system.

clicking the submit button on the order entry form, the provided data is converted into an XML file and zipped together with all the provided files to a zip archive. The XML file is necessary to ensure that the product configuration product line is able to automatically interpret the given zip file. Further, having an XML file has also the positive side effect that it is human readable which is essential for manual verification steps. Additional data can be included into the archive like identifiers and time-stamps to have a traceable link from placing an order to the actual manufacturing of the product.

Based on the provided data, the final product can be generated using dedicated code generators. To allow a flexible system without the need of re-releasing the product line if new products are supported, we decided to define run-time configurable generators. For this purpose we defined a Domain Specific Language (DSL) which is designed to be used by product experts. This DSL is called Function Description Language (FDL) and is used to create customization scripts for the final product. This means that during the production process, a script is loaded to each device which is triggered automatically to customize the according devices. For every supported product family, a Function Description (FD) is written which basically lists all the possible features of the platform (i.e. the customization options of the order process model) and how the provided order data is translated into the final script. The overall concept is illustrated in Figure 5. A script library is used which contains common scripts that can be referenced by the function description. For example, a 'loadApplication' script may be developed for the product line in order to install the customer provided application to the devices.

5 EVALUATION

First results were already compiled in our previous works (Sinnhofer et al. (2016, 2017)) in which we compared the development efforts using ”traditional

software development" techniques and compared it with the overhead of the developed framework. We use the term "traditional software development" techniques for a software development with ad-hoc (or near to ad-hoc) software architecture which means that multiple different systems are designed almost independent, but make use of copy and adapt strategies. Consequently, the maintenance efforts for such systems are rather high. We investigated, that the economical breakeven point of the developed framework is at around 3 to 4 systems. Further, the robustness of the customization process was increased since automatic methods were used for the feature selection and thus, configuration errors could be reduced significantly. Through the use of automatic methods, it was also possible to generate log-files for certification purposes which are used to ensure that the provided customer data was loaded and not confused or manipulated.

In this work, we will investigate other aspects of the developed framework, namely the identification of business drivers: We analyzed the development efforts of individual product features and contrasted them with the revenue that was earned by selling the according product configuration. The development efforts were extracted from the time-recordings of the responsible workers and should give a reasonable estimate about the real development efforts. In total, 106 different product orders were analyzed which provided 30 individual configuration settings. The results are illustrated in Figure 6. The first 10 product configuration options are internal system specific configuration settings which are mandatory for every product. A business decision was taken to reduce the costs for the base system to a minimum level to ensure a low cost base product. As a consequence, the revenue earned by the basic product configuration is rather low compared to other customization options. An interesting finding was that a lot of development effort was invested in complex features which were never used for any customer or are only rarely used. Due to this finding, it was decided that some of the features will be removed from the product in a future release, in order to reduce the overall costs. As illustrated in the Figure, feature 12 required a lot of development effort, but is also frequently ordered by customers. Consequently, an improvement process was triggered in order to reduce the costs of this feature.

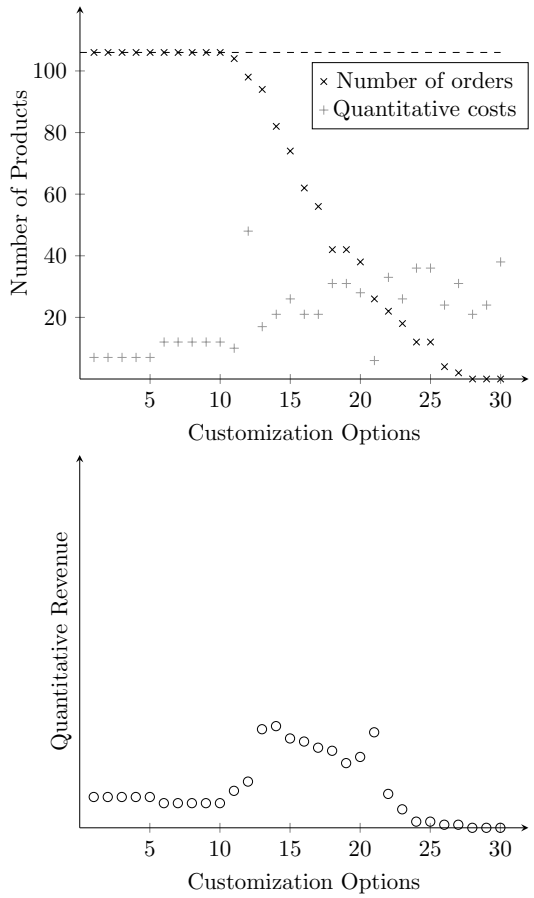

Figure 6: Analysis of the development efforts and revenue to identify business drivers. The revenue and the costs are illustrated in a quantitative manner.

After having discussed the positive aspects of the developed framework, we also want to address some limitations of the current implementation: While we were able to fully generate the required customization scripts for simple product configurations, we were able to only partially generate the scripts for complex product configurations due to the high number of inter-feature constraints of the product features. This is not a technical problem of the approach, but having a complete coverage of all inter-feature constraints is a time-consuming and iterative process. Further, modeling all the constraints in advance is usually not possible for complex systems. As a result, we decided to model only basic constraints in advance and to update the constraint model with every product order. Based on this semi-automatic generation, we managed to reduce the time to release a complex product by 50 percent.

6 RELATED WORK

Traditionally, business process modeling languages do not explicitely support the representation of families of process variants (Rosa et al. (2017)). As a consequence, a lot of work can be found which tries

to extend traditional process modeling languages with notations to build adaptable process models. As such, adaptable process models can be customized according to domain requirements by adding or removing fragments to the model and by explicitly transforming this model to dedicated process variants which can be executed in the field. This promises to increase the flexibility of business process oriented organizations with respect to highly flexible requirements of the market. Having such a variability modeling for business process models builds the foundation of this work. Thus, related work which is utilizing similar modeling concepts are presented in the following:

Derguech (2010) presents a framework for the systematic reuse of process models. In contrast to this work, it captures the variability of the process model at the business goal level and describes how to integrate new goals/sub-goals into the existing data structure. The variability of the process is not addressed in his work.

Gimenes et al. (2008) presents a feature based approach to support e-contract negotiation based on web-services (WS). A meta-model for WS-contract representation is given and a way is shown how to integrate the variability of these contracts into the business processes to enable process automation. It does not address the variability of the process itself but enables the ability to reuse business processes for different e-contract negotiations.

While our used framework to model process variability reduces the overall process complexity by splitting up the process into layers with increasing details, the PROVOP project (Hallerbach et al. (2009a,b) and Reichert et al. (2014)) focuses on the concept, that variants are derived from a basic process definition through well-defined change operations (ranging from the deletion, addition, moving of model elements or the adaptation of an element attribute). In fact, the basic process expresses all possible variants at once, leading to a big process model. Their approach could be beneficial considering that cross functional requirements can be located in a single process description, but having one huge process is also contra productive (e.g. the exchange of parts of the process is difficult).

The work of Gottschalk et al. (2007) presents an approach for the automated configuration of workflow models within a workflow modelling language. The term workflow model is used for the specification of a business process which enables the execution in an enterprise and workflow management system. The approach focuses on the activation or deactivation of actions and thus is comparable to the PROVOP project for the workflow model domain.

Rosa et al. (2008) extend the configurable process modelling notation developed from Gottschalk et al. (2007) with notions of roles and objects providing a way to address not only the variability of the control-flow of a workflow model but also of the related resources and responsibilities.

The Common Variability Language (CVL Haugen et al. (2013)) is a language for specifying and resolving variability independent from the domain of the application. It facilitates the specification and resolution of variability over any instance of any language defined using a MOF-based meta-model. A CVL based variability modelling and a BPM model with an appropriate model transformation could lead to similar results as presented in this paper.

The work of Zhao and Zou (2011) shows a framework for the generation of software modules based on business processes. They use clustering algorithms to analyse dependencies among data and tasks, captured in business processes. Further, they group the strongly dependent tasks and data into a software component.

7 CONCLUSION

The reuse of business process models is an important step for process driven organizations to survive in a competitive market. Through an integrated view of the according IT, it is possible to raise the efficiency of the overall business. With this and our previous works, we proposed a way to use software product line engineering techniques for the modeling of business process models. Further, we developed a framework which enables to combine the variability models of order processes with the variability models of software customization product lines. This enables an automatic customization process which is triggered by the according order processes. As a result, the development costs and the required time to react to changes of the market can be reduced significantly. Moreover, using the proposed techniques supports Domain Experts to identify business drivers and thus, raise the overall efficiency of the organization.

In the current state, the presented framework is focused on covering the variability of order processes for similar type of products. Consequently, future work will address the extension of the developed framework to other processes. Further, we are currently investigating methods on how to bind non-functional requirements like security requirements to the variability models in order to enforce specific properties throughout the whole process in an automatic and systematic way.

ACKNOWLEDGEMENTS

The project is funded by the Austrian Research Promotion Agency (FFG). We want to gratefully thank Danilo Beuche from pure::systems for his support.

REFERENCES

Association of Business Process Management Professionals (2009). *Guide to the Business Process Management Common Body of Knowledge: ABPMP BPM CBOK®*. Association of Business Process Management Professionals.

Derguech, W. (2010). Towards a Framework for Business Process Models Reuse. *In The CAiSE Doctoral Consortium*.

Gimenes, I., Fantinato, M., and Toledo, M. (2008). A Product Line for Business Process Management. *Software Product Line Conference, International*, pages 265–274.

Gottschalk, F., van der Aalst, W. M. P., Jansen-Vullers, M. H., and Rosa, M. L. (2007). Configurable Workflow Models. *International Journal of Cooperative Information Systems*.

Hallerbach, A., Bauer, T., and Reichert, M. (2009a). Guaranteeing Soundness of Configurable Process Variants in Provop. In *Commerce and Enterprise Computing, 2009. CEC '09. IEEE Conference on*, pages 98–105. IEEE.

Hallerbach, A., Bauer, T., and Reichert, M. (2009b). Issues in modeling process variants with Provop. In Ardagna, D., Mecella, M., and Yang, J., editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 56–67. Springer Berlin Heidelberg.

Hammer, M. and Champy, J. (1993). *Reengineering the Corporation - A Manifesto For Business Revolution*. Harper Business.

Haugen, O., Wasowski, A., and Czarnecki, K. (2013). Cvl: Common variability language. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute.

McCormack, K. P. and Johnson, W. C. (2000). *Business Process Orientation: Gaining the E-Business Competitive Advantage*. Saint Lucie Press.

Object Management Group, O. (2011). Business process model and notation (bpmn). version 2.0. pages 1–538. available at http://www.omg.org/spec/BPMN/2.0/.

Österle, H. (1995). *Business Engineering - Prozess- und Systementwicklung*. Springer-Verlag.

Pohl, K., Böckle, G., and Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Reichert, M., Hallerbach, A., and Bauer, T. (2014). Lifecycle Support for Business Process Variants. In Jan vom Brocke and Michael Rosemann, editor, *Handbook on Business Process Management 1*. Springer.

Rosa, M. L., Aalst, W. M. P. V. D., Dumas, M., and Milani, F. P. (2017). Business process variability modeling: A survey. *ACM Comput. Surv.*, 50(1):2:1–2:45.

Rosa, M. L., Dumas, M., ter Hofstede, A. H. M., Mendling, J., and Gottschalk, F. (2008). Beyond control-flow: Extending business process configuration to roles and objects. In Li, Q., Spaccapietra, S., and Yu, E., editors, *27th International Conference on Conceptual Modeling (ER 2008)*, pages 199–215, Barcelona, Spain. Springer.

Saidani, O. and Nurcan, S. (2007). Towards context aware business process modelling. In *8th Workshop on Business Process Modeling, Development, and Support (BPMDS07), CAiSE*, volume 7, page 1.

Sinnhofer, A. D., Pühringer, P., and Kreiner, C. (2015). varbpm - a product line for creating business process model variants. In *Proceedings of the Fifth International Symposium on Business Modeling and Software Design - Volume 1: BMSD,*, pages 184–191.

Sinnhofer, A. D., Pühringer, P., Potzmader, K., Orthacker, C., Steger, C., and Kreiner, C. (2016). A framework for process driven software configuration. In *Proceedings of the Sixth International Symposium on Business Modeling and Software Design - Volume 1: BMSD,*, pages 196–203.

Sinnhofer, A. D., Pühringer, P., Potzmader, K., Orthacker, C., Steger, C., and Kreiner, C. (2017). *Software Configuration Based on Order Processes*, pages 200–220. Springer International Publishing, Cham.

Strnadl, C. F. (2006). Aligning business and it: The process-driven architecture model. *Information Systems Management*, 23(4):67–77.

Valena, G., Alves, C., Alves, V., and Niu, N. (2013). A Systematic Mapping Study on Business Process Variability. *International Journal of Computer Science & Information Technology (IJCSIT)*.

Weiss, D. M. and Lai, C. T. R. (1999). *Software Productline Engineering: A Family-based Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Willaert, P., Van Den Bergh, J., Willems, J., and Deschoolmeester, D. (2007). *The Process-Oriented Organisation: A Holistic View - Developing a Framework for Business Process Orientation Maturity*. Springer.

Zhao, X. and Zou, Y. (2011). A business process-driven approach for generating software modules. *Software: Practice and Experience*, 41(10):1049–1071.