
Activity Diagram Based Process Family Architectures for Enterprise Application Families^{*}

Arnd Schnieders and Mathias Weske

Hasso-Plattner-Institute for IT Systems Engineering at University Potsdam,
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
{arnd.schnieders, mathias.weske}@hpi.uni-potsdam.de

Keywords: Process Family Engineering, Business Process Variability, Process Configuration, Software Product Lines, Variability Mechanisms

1 Introduction

Today, enterprise software vendors typically sell their enterprise applications in many different variants, which allows for a better market penetration regarding diverging customer needs and financial power of the customers. Since one enterprise application can be composed of several subsystems coming from different software providers interoperability issues between the different subsystems have to be taken into account during the design and implementation of the enterprise application and its variants.

One effective technique for the development of families of similar software system variants is known as product family engineering. Product family engineering concentrates on a concrete family of software variants tackling any interoperability issues explicitly for the set of family members intended to be part of the product family. Advantages of applying product family engineering techniques are reduced development costs, a quicker time to market for the system variants, and typically a higher software quality [1]. Therefore, the application of product family engineering techniques can help improving the

^{*} The work reported in this paper has been supported by the German Ministry of Research and Education by the PESOA project

effectiveness of the enterprise application development as well as the quality of the developed enterprise application variants. Moreover, the speed of enterprise application customization can be increased significantly.

However, up to now product family engineering research has concentrated on software systems, where static diagrams like class diagrams or component diagrams represent the main blueprint for the development of the software system while process oriented software has not been regarded adequately. As a result, existing product family engineering approaches are not suited well for process oriented software like enterprise applications. The intention of our work is to contribute in closing this gap by supporting the investigation of product family engineering of process oriented software, in short process family engineering. Thereby, we concentrate on the representation of process family architectures following a new variability mechanism centric approach [2], [3]. In this paper we provide rules on how the process-oriented architecture for a concrete enterprise application variant can be derived from a process family architecture for a family of enterprise applications described as a variant rich Activity Diagram [4], [5].

This paper is structured as follows: In section 2 we give a brief introduction to Process Family Engineering. In section 3 we introduce to variability mechanism centric process family architecture modeling. In section 4 we describe rules for deriving product variant specific process architectures from process family architectures. Section 5 summarizes the contents of this paper and identifies open issues subject to future research.

2 Preliminaries

According to [1] product family engineering is a paradigm to develop software applications using a set of software subsystems and interfaces that form a common structure based on which derivative products tailored to individual customer needs can be efficiently developed. Another important aspect is that within a software product family reuse isn't restricted to the reuse of implementation artifacts but is expanded to any development artifact (like e.g. requirement or design models).

Product family engineering is characterized by a so called dual lifecycle [6] as indicated in figure 1. In the first section of the product family development process (called product family engineering) generic development artifacts, called the product family infrastructure, are developed based on which product family members are derived efficiently in the corresponding phase within the second section (called application engineering) of the product family engineering process. In order to emphasize that our work focuses on the development of process-oriented software, we use the term process family engineering instead of product family engineering and process family infrastructure instead of product family infrastructure.

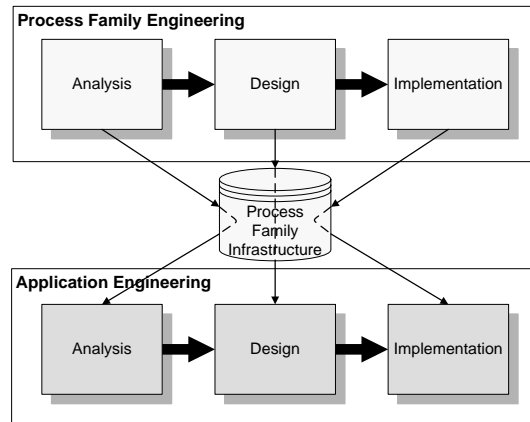


Figure 1. Process Family Engineering Process

3 Modeling Process Family Architectures

During the design of a process family a process family architecture is developed based on the process family requirements. The process family architecture acts as a reference architecture for the members of the process family and describes the basic structure for the applications of the process family. It defines the reusable system parts and their interfaces and has to cover both, the functional as well as the non-functional requirements on the process family. Moreover, the process family architecture describes which variability mechanisms shall be applied for realizing the variability and on which variation points they shall be applied. The selection of appropriate variability mechanisms is crucial for the design of the process family since they can have a substantial impact on the functional and non-functional properties of the system. Using the wrong variability mechanism it may for example be impossible to integrate subsystem variants provided by external enterprise vendors. E.g. the invocation of varying web service implementations would be an unsuitable variability mechanism for the integration of enterprise applications only accessible by CORBA function calls. Additionally, the proper selection of a variability mechanism guarantees for an easy generation of process family members based on the process family infrastructure.

Thus, for supporting process family engineering, concepts and a notation for PFA (process family architecture) variability mechanisms are required, which allow for modeling architecturally relevant decisions concerning the realization of the system's variability. Figure 2 describes the correlation between the process family requirements, the process family architecture, PFA variability mechanisms and implementing variability mechanisms. The model is structured according to the three phases of process family engineering into three packages: 'Analysis', 'Design' and 'Implementation'. The requirements on the process family members are realized by a corresponding process fam-

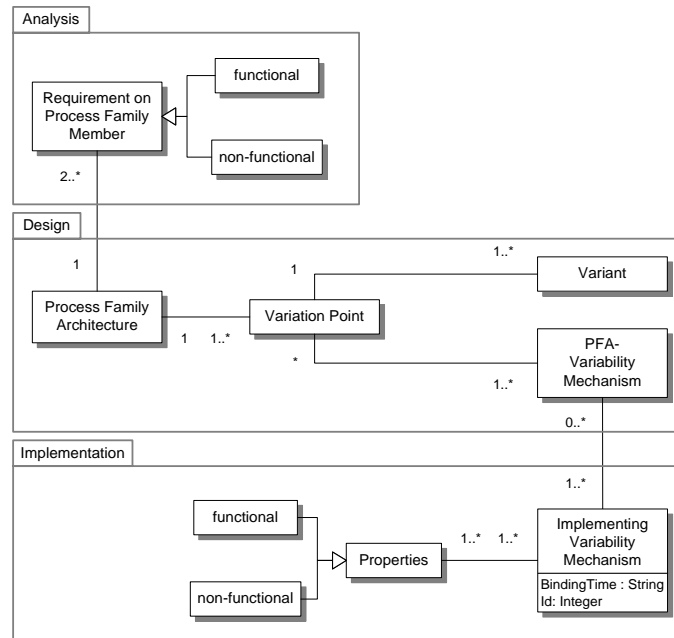


Figure 2. Role of Variability Mechanisms in Process Family Engineering

ily architecture. Even though a process family architecture will typically also comprise static diagrams, we focus on the behavioral aspects, which have been neglected in research so far. The variability in the process family is modeled by means of variation points to which variants can be bound by means of PFA variability mechanisms. The variability mechanisms represented in the process family architecture are realized by implementing variability mechanisms. During the implementation different variability mechanisms can come into question, which can show different binding times. Which variability mechanisms are available highly depends on the application domain and the enterprise application to be implemented. The idea is that a set of implementing variability mechanisms with the same functional properties shall be represented by the same PFA variability mechanisms. Thereby, the variability modeling determines the functional and the resulting non-functional properties of the variability implementation. Moreover, a binding time can be specified for the variability, which is also application domain dependent.

3.1 Notation for Process Family Architectures in UML

As indicated in the example for the variability mechanism parameterization shown in figure 3 in order to highlight variation points in UML Activity Diagrams the stereotype `<<VarPoint>>` is assigned to the affected Activity Diagram element. Variants, which are identified by the stereotype `<<Variant>>` are assigned to their variation point using UML Dependencies. The variability

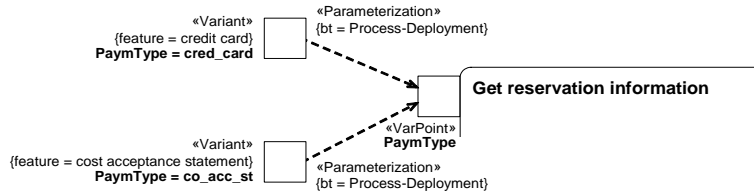


Figure 3. Notation for Variability in Process Family Architectures

mechanism is modeled by assigning a stereotype with a name of the variability mechanism to the Dependency relation. A list of stereotypes for identifying variability mechanisms is provided in [3]. Moreover, we suggest the introduction of the stereotype *«Variable»* as a generic identifier to express that a process element is somehow affected by variability. Concerning parameterization, it can be used to highlight the parameter dependent process parts. The *«Variable»* stereotype can also be used to indicate that a CallBehaviorAction invokes an Activity that disposes of variation points. The binding time can be displayed by means of a tagged value (tagged value key *bt*) of the variability mechanism stereotype and, if necessary, the implementing variability mechanism may be uniquely identified by adding an identifier (tagged value key *id*). The functional system requirements a variation point implements are represented by means of a tagged value (tagged value key *feature*) of the variation point stereotype, which can hold a list of functional system requirements.

4 Deriving Process Architectures from Process Family Architectures

In this section we describe a set of PFA variability mechanisms, briefly describe their functionality, illustrate their representation in UML Activity Diagrams, and provide rules for their resolution during process architecture derivation. Thereby we provide a means to introduce variability into Activity Diagrams.

4.1 Encapsulation of Varying Subprocesses

Functionality: Application-specific subprocess implementations are inserted into an invariant subprocess interface. **References:** Encapsulation is a variability mechanism also pointed out by [7], [8] to be relevant on a design model level.

As shown in figure 4 encapsulation of varying subprocesses in Activity Diagrams is realized by means of CallBehaviorActions invoking varying Activities, which serve as the varying subprocess implementations. For deriving an application specific process architecture the application specific subprocess

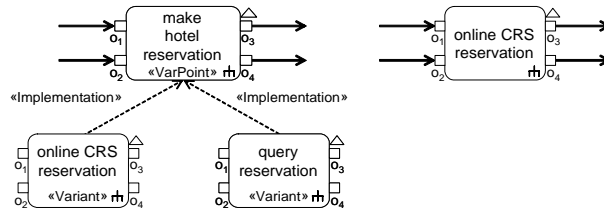


Figure 4. Variability Mechanism Encapsulation of Varying Subprocesses

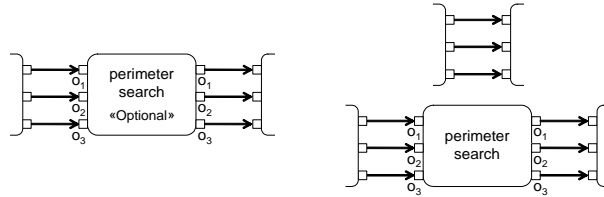


Figure 5. Variability Mechanism Addition/Omission of Encapsulated Subprocesses

implementation variant is assigned as the invoked behavior to the CallBehaviorAction representing the subprocess interface as explicitly shown on the right hand side of figure 4. Additionally, any variability information (i.e. the stereotypes and the variants which are not required for the application) is deleted from the diagram.

4.2 Addition, Replacement, Omission of Encapsulated Subprocesses

Functionality: Variability in software systems can be reached by the addition, replacement, and omission of encapsulated subprocesses. **References:** The addition, replacement, and omission of subsystems are referred to as a variability mechanism (for static systems) by [9].

If a subprocess is omitted the incoming and outgoing arcs of the omitted subprocesses are joined pairwise, i.e. the target-nodes of the outgoing ActivityEdges are assigned as the new target-nodes to the respective incoming ActivityEdges. As described in [3] in order to avoid syntax violations the deleted CallBehaviorAction must have the same number and types of ingoing and outgoing ActivityEdges. The left hand side of figure 5 gives an example for an optional Activity, which is once present and once omitted as shown on the right hand side of the figure.

Figure 6 demonstrates the derivation of process architectures from process family architectures using the variability mechanism replacement of encapsulated subprocesses. In contrast to encapsulation, replacement allows for the substitution of entire subprocesses (i.e. the subprocess implementation as well as the subprocess interface). In the upper case (in the right half of figure 6)

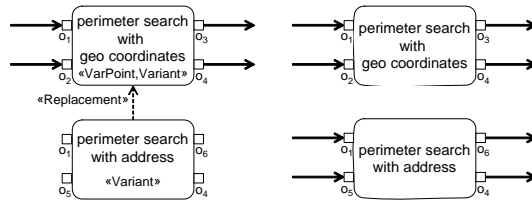


Figure 6. Variability Mechanism Replacement of Encapsulated Subprocesses

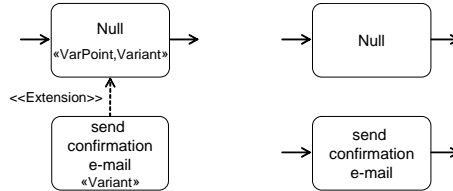


Figure 7. Variability Mechanism Extensions/Extension Points

a process architecture variant is derived where the subprocess hasn't been replaced, while it has been replaced in the lower process architecture variant.

4.3 Extensions/Extension Points

Functionality: Extensions and Extension Points are used to extend a subprocess at predefined points, the extension points, by additional optional behavior selected from a set of possible variants. **References:** Extensions/Extension Points are a very common variability mechanism referred to in many publications [10], [9], [11], [12].

As shown in the right half of figure 7 either a subprocess architecture variant without extension can be derived (upper case), where a null-Activity without visible behavior acts as a placeholder, while in the lower case a subprocess architecture has been derived replacing the null-Activity invoked by the CallBehaviorAction by an extending Activity.

4.4 Parameterization

Functionality: Using parameterization variants of encapsulated subprocesses are generated by configuring a generic encapsulated subprocess with a set of parameter values. The prerequisite for this is that all possible variants are provided in the subprocess's code. **References:** Parameterization is referenced as variability mechanism by [13], [7], [12].

An example for the parameterization of an Activity is shown on the left hand side of figure 8, where the Activity „Process reservation“ is parame-

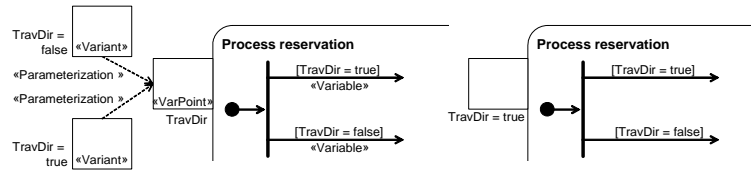


Figure 8. Variability Mechanism Parameterization

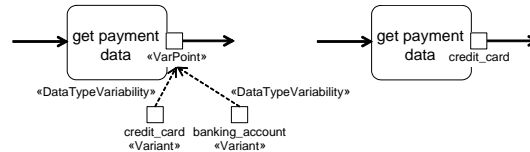


Figure 9. Variability Mechanism Data Type Variability

terized. The initial value of a value pin is initialized according to the desired variant. Depending on the parameter value guard conditions then activate/deactivate flows within the Activity. On the right hand side of the figure a configuration is shown where the initial value of the value pin is set to the value „true“.

4.5 Data Type Variability

Functionality: Data type variability allows for varying the type of data being processed by a computer system. **References:** referred to as templates by [10], [11], [12] as variability mechanism.

The left half of figure 9 gives an example for the representation of data type variability. As the exemplary process architecture on the right half indicates, data type variability is resolved by assigning a variant specific data type to an ObjectNode, in this case a Pin.

4.6 Design Patterns

Here we will concentrate on the 'Strategy Pattern' as one of the Design Patterns referenced most frequently in the context of product family engineering. **Functionality:** The idea of the Strategy Pattern is to make different subprocess variants, which are hidden behind a common interface, interchangeable. The algorithm variants are derived from a basic subprocess version using inheritance. **References:** Some 'Gang of Four' Design Patterns [14] like the 'Adapter', 'Strategy', 'Template Method', 'Factory', 'Abstract Factory', 'Builder', and 'Decorator' Pattern are frequently referred to as variability mechanisms [10], [12], [7].

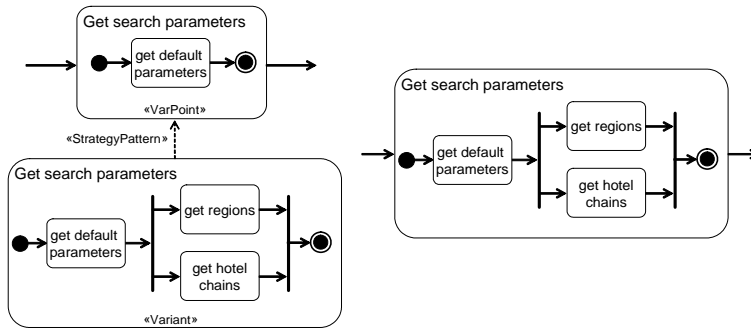


Figure 10. Variability Mechanism Strategy Pattern

In Activity Diagram based process family architectures the Strategy Pattern allows for deriving varying subprocess implementations from a basic, abstract subprocess implementation by means of Activity Diagram Inheritance according to the rules described in [15]. The variant specific subprocess implementation is then invoked in the respective process architecture variant instead of the basic subprocess implementation as illustrated in the right half of figure 10.

5 Conclusions

In this paper we have shown how the behavioral aspects of a process family architecture for a family of enterprise applications can be modeled as a variant-rich Activity Diagram and have described rules for deriving process architectures from process family architectures for members of the family. The presented variability mechanism centric technique for modeling process family architectures allows for tackling variability implementation issues during the development of a suitable process family architecture for a family of process oriented systems. Variability implementation issues include among others the integration of varying system parts, which can be provided by different parties and the impact of the variability implementation on the non-functional properties of the developed system family. Moreover, variability mechanism centric modeling of process family architectures supports the reuse of parts of the process design, since process architectures for system variants can be derived simply following the rules described in section 4 of this paper. While an extensive case study has been performed for modeling variability mechanism centric process family architectures for a family of e-business applications (in excerpts published in [16]) open issues for future work comprise a case study for the implementation of process family architecture variability mechanisms for a family of enterprise applications.

6 References

- [1]. Pohl, K., Böckle, G., van der Linden, F.: *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer (2005)
- [2]. Puhlmann, F., Richter, E., Schnieders, A., Weiland: *Variability Mechanisms for Process Models*. PESOA-Report No. 17/2005. Technical report, Daimler-Chrysler Research and Technology, Hasso-Plattner-Institute for IT Systems Engineering (June 2005)
- [3]. Schnieders, A.: *Variability Mechanism Centric Process Family Architectures*. In: *Proceedings of the 13th Annual IEEE International Conference on the Engineering of Computer Based Systems ECBS 2006* (to appear), IEEE Computer Society Press (2006)
- [4]. OMG: *UML 2.0 Superstructure Specification*. (2003)
- [5]. Pender, T.: *UML Bible*. Wiley Publishing Inc., Indianapolis, Indiana (2003)
- [6]. Weiss, D.M., Lai, C.T.R.: *Software Product-line Engineering: a Family-based Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
- [7]. Gomaa, H.: *Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures*. Addison-Wesley Professional (2005)
- [8]. Gomaa, H., Webber, D.: *Modeling Adaptive and Evolvable Software Product Lines Using the Variation Point Model*. In: *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, IEEE Computer Society Press (2004)
- [9]. Clements, P., Northrop, L.: *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley (2001)
- [10]. Bosch, J.: *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, Harlow, England et al. (2000)
- [11]. Jacobson, I., Griss, M., Jonsson, P.: *Software Reuse: Architecture, Process and Organization for Business Success*. Addison Wesley Longman, Harlow, England et al. (1997)
- [12]. Svahnberg, M., Bosch, J.: *Issues Concerning Variability in Software Product Lines*. Volume June of 146. *Lecture Notes in Computer Science* (2003)
- [13]. Bachmann, F., Bass, L.: *Managing Variability in Software Architectures*. In: *Proceedings of the 2001 Symposium on Software Reusability*, ACM Press (2001)
- [14]. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley (1995)
- [15]. Schnieders, A., Puhlmann, F.: *Activity Diagram Inheritance*. In: *Proceedings of the 8th International Conference on Business Information Systems BIS*. (2005)
- [16]. Plötner, D., Kose, M., Hering, T., Werner, A.: *Prozesse im E-Business am Beispiel ausgewählter Geschäftsprozesse des Partners ehotel AG*. PESOA-Report No. 20/2005. Technical report, ehotel AG, Universität Leipzig (June 2005)