PESOA

Process Family Engineering in Service-Oriented Applications

BMBF-Project

Process Family Engineering

Modeling variant-rich processes

Authors:
Joachim Bayer
Winfried Buhl
Cord Giese
Theresa Lehner
Alexis Ocampo
Frank Puhlmann
Ernst Richter
Arnd Schnieders
Jens Weiland
Mathias Weske

PESOA-Report No. 18/2005
Sep 01, 2005

PESOA is a cooperative project supported by the federal ministry of education and research (BMBF). Its aim is the design and prototypical implementation of a process family engineering platform and its application in the areas of e-business and telematics.
The project partners are:

- DaimlerChrysler Inc.
- Delta Software Technology GmbH.
- eHotel AG
- Fraunhofer IESE
- Hasso-Plattner-Institute
- University of Leipzig

PESOA is coordinated by
Prof. Dr. Mathias Weske
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam

www.pesoa.org

Abstract

In this report, we present the methodological foundation for process family engineering. The methodological foundation consists of a *conceptual model* for variant-rich processes and a process engineering process called the *PESOA process*. The conceptual model describes the conceptual requirements for defining variant-rich processes in both the e-business and the automotive domains, whereas the PESOA process embeds the variant-rich processes in an approach for developing, using, and maintaining families of processes for such domains.

The methodological foundation constitutes a fundamental achievement with respect to the main goal of the PESOA project, which is to design and implement a platform for process families of related applications. Such platform shall support the management of the variant-rich processes contained in a process family and enable the automatic process-based instantiation of applications.

Keywords: PESOA, Process, Variant-Rich Processes, Process Variability

Table of Contents

1 Introduction

1.1 Project Context

PESOA is a cooperative project financed by the German federal ministry of education and research (BMBF). The goal of the PESOA project is to design and implement a platform for families of related applications. The envisioned platform is used to manage process variants for families of applications and to enable the process-based instantiation of such application families. This goal is addressed by enhancing the approved technologies from the area of domain engineering, product line engineering, and software generation with new methods from the area of workflow management.

1.2 Goals of the Report

In this report, we present the methodological foundation for process family engineering for applications. The motivation for process family engineering is to transfer product line engineering approaches to software engineering domains, where processes are the driving software engineering artefact. For example, the e-business domain, where workflows determine the developed applications or automotive domain, where embedded control processes play the same role in the development of software for embedded control units.

The goal of product line engineering is to handle a number of similar software systems together, enabling large scale reuse during the development and maintenance of the different systems covered by the product line. The transfer of this principal approach to software engineering domains that use processes as driving software engineering artifact leads to process family engineering.

In this report, we present the techniques that have been developed in the PESOA project to realize process family engineering. These techniques support the definition of processes in the PESOA domain and embed the resulting variant-rich processes in a software engineering approach for process families.

1.3 Outline

This report is structured as follows. Chapter 2 motivates process family engineering and describes the techniques required for process family engineering. Chapter 3 presents a conceptual model that reflects the required concepts for modeling processes in the PESOA domains. In Chapter 4, variability is introduced to the conceptual model in order to define the conceptual basis for modeling variant-rich processes in the PESOA domain. Chapter 5 contains the general PESOA process that embeds the variant-rich processes in an approach for developing, using, and maintaining families of processes in the PESOA domains. Both, the conceptual model and the PESOA process, are defined in a general, domain-independent way. They need to be adapted to the domain in which they are used. The realization of the concepts for modeling variant-rich processes in the two PESOA domains, e-business and automotive software engineering, is described in chapter 6.

2 Motivation

2.1 Goal

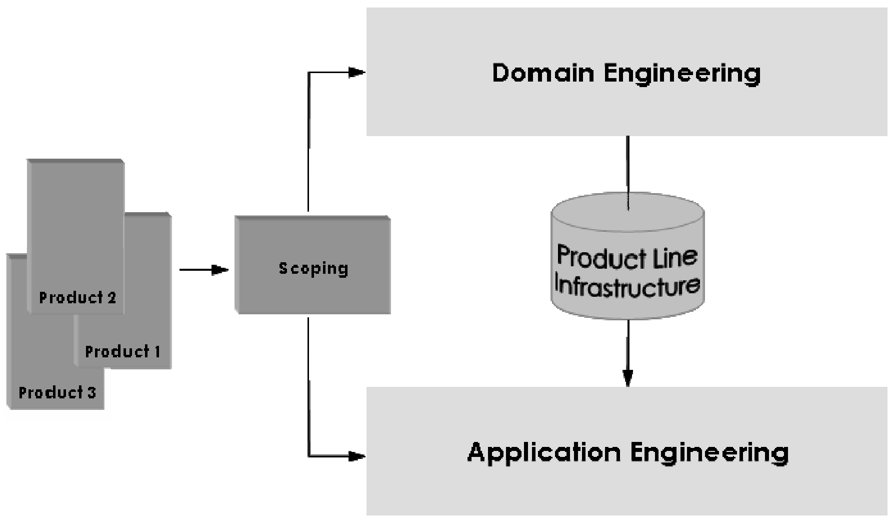
Business and embedded control processes are often used as a central software engineering artifact. Then, processes determine the different characteristics of the developed systems. This process-based software development approach has not been taken into great consideration in software product line engineering. In this report, we take a first step towards an approach for process-based product line engineering. Using processes as drivers for product line engineering leads to an approach in which processes are used to determine the scope of a product line, to differentiate the product line members, and to specify them.

Product line engineering distinguishes two development phases – domain and application engineering – as presented in Figure 1. Each phase is preceded by an activity defining the scope of the product line that identifies the systems which are members of a product line and the systems outside the product line. Scoping is done by investigating a set of concrete products that already exist, are planned, or are envisioned. The result of scoping is a set of products that make up the product line along with the features of the different product line members. The relationship between product line members and features is often documented in so-called product maps [1].

Based on a product map, domain engineering identifies the common features (commonalities) and the variable features (variabilities) of the identified products. Commonalities define the skeleton of the systems in the product line; variabilities bind the space of required and anticipated variations of the common skeleton. Each artifact produced during domain engineering contains the commonalities and specially labelled variabilities. These so-called variant-rich artifacts are stored in the product line infrastructure.

During application engineering the product line infrastructure is instantiated to create a concrete product; the commonalities are reused and the variabilities are resolved for the specific product.

Figure 1: Product line engineering



In the following, we sketch the envisioned process-based product line engineering approach by presenting the differences in scoping, domain engineering, product line infrastructure, and application engineering when processes are used as driving software engineering artifact.

Scoping generally elicits the product line members and documents the relationship between products and their features in a product map. A product map in process-based product line engineering associates the products, which provide processes, with the main process functionality and further process characteristics. The different characteristics of processes are given in chapter 3.

Domain engineering identifies commonalities and variabilities and documents them in variant-rich artifacts. In process-based product line engineering, domain engineering produces a number of variant-rich processes. The variant-rich processes cover a number of similar processes. Variability in the processes is explicitly modeled. In [2] we showed that technical processes and business processes are similar enough to use the same mechanisms for modeling variability.

Process-based application engineering develops a specific product by instantiating the variant-rich process. The instantiation or rather the resolution of variabilities is also supported by what we call configuration models that combine features and process characteristics with variabilities. The resulting specific process can then be used to develop applications/products as done in traditional process-based software engineering.

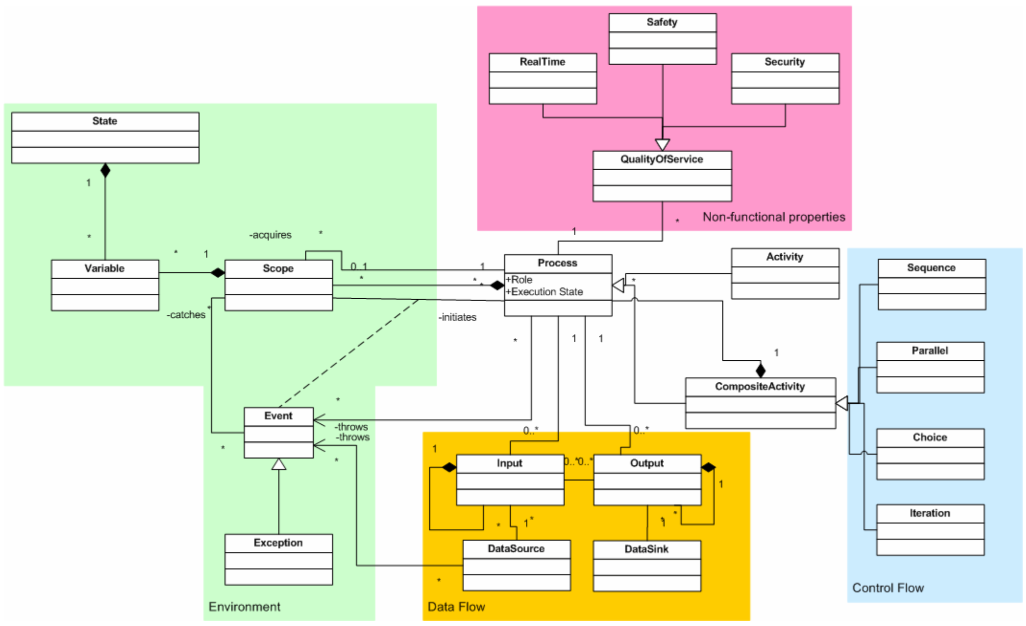
2.2 Approach

The concepts that will be described to introduce variability and configuration modeling to the processes in the PESOA domains are based on a light-weight approach for facilitating the transition towards product lines [7]. This is a systematic approach to extend any given asset to be generic, that is, to enable the explicit modeling of variability in that asset. Once variability is modeled, a variability management approach enables the extension of arbitrary assets to become product line suitable. The approach also enables an explicit modeling of variability and the derivation of single product line members.

3 Conceptual Process Model

In this section the necessary concepts for modeling processes from the e-business, as well as from the automotive domain are presented. Examples will be provided with the purpose of illustrating the concepts defined in the conceptual model.

Figure 2: **Conceptual model (UML static structure diagram)**



The conceptual model presented in Figure 2 as a UML Static Structure is intended to describe both the requirements on processes in the e-business and the automotive domain extracted from the analysis performed in [1].

Figure 2 is split into areas i.e., process, control flow, data flow, non-functional properties, and environment. Each area is characterized by a set of concepts defined within it. The areas have been used for structuring the description of the conceptual model, in the following sections.

3.1 Processes

Processes are used for specifying e-business systems and for specifying control systems in the automotive domain. In e-business, a process is typically a workflow, while in automotive, an embedded control process describes possible sequences executing related control functions of a controller. Therefore, the process concept constitutes the core of the conceptual model [3].

The process can be hierarchically refined into smaller activities until an atomic level is reached that should or cannot be parted anymore. This is the functional aspect [3] and it consists of composite and atomic activities.

These relationships between process, composite activity, and activity have been realized in the conceptual model by means of the composite pattern. The composite pattern is often used to represent recursive data structures [4]. It allows a client object to treat both single components and collections of components identically. Looking at Figure 2 this means that if the scope is considered a client object, then it could be said that a process, a composite activity, as well as an activity each have an associated scope. The same can be assumed for other concepts that have an association with the process concept.

A composite activity groups atomic activities within a process. Thus, in e-business, a composite activity will group several workflow activities. An example can be seen in the “Register Auction Item Process” (see Figure 3), where the composite activity “Describe item” encompasses (atomic) activities such as for example: “Search for existing categories”, “Check item characteristics”, “Save item description”. In automotive, a composite activity groups several sub-processes. One example is the process for controlling the engine operation. This process comprises (sub-) processes such as “Controlling Operating Time”, “Controlling Engine Temperature”, and “Controlling Catalytic Converter” (see Figure 4).

An activity is an atomic step in a process. Each process contains at least one activity. In e-business, an activity belongs to the functional workflow aspect [3]. Examples can be seen in Figure 3 : “Reject item”, “Select Auction Type”, Review fees”. An activity in the context of automotive embedded control processes constitutes a data transformation. An example of an activity is reading out a characteristic or calculating function values.

Figure 3: Register auction item process: process, composite activity, and activity (BPMN notation)

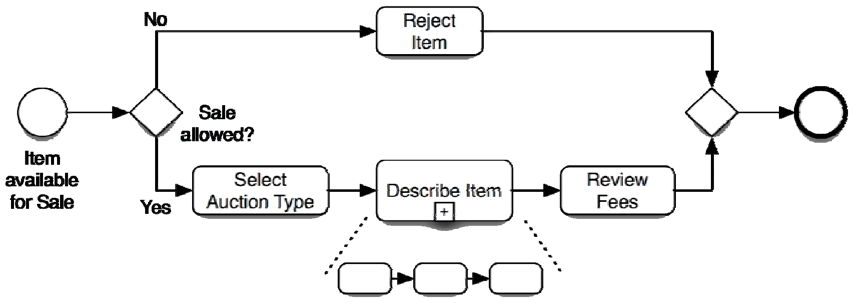


Figure 3 presents a BMPN example [5] where the realization of the earlier presented concepts can be observed. The composite activity “Describe item”, which is a process as well, is marked specially with a “+” symbol inside a square. The remaining are single activities.

Figure 4: Operating engine process: realization of process, composite activity, and activity [14] (UML activity diagram)

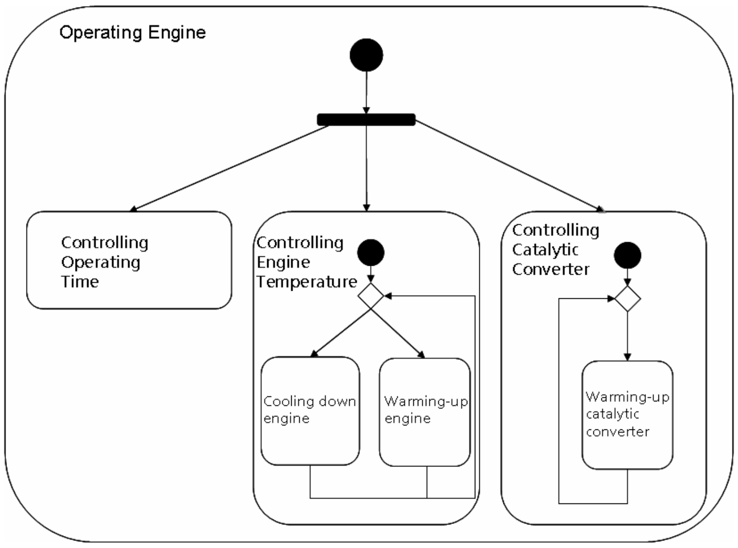

Operating Engine
Controlling Operating Time
Controlling Engine Temperature
Cooling down engine
Warming-up engine
Controlling Catalytic Converter
Warming-up catalytic converter

Figure 4 presents an UML Activity Diagram Example [14] with an example of the automotive domain. Composite activities are: “Controlling Engine Temperature” and “Controlling Catalytic Converter”. The remaining are single activities.

Roles can be assigned directly either to a process, a composite activity, or a single activity, since they have been modeled as an attribute of process.

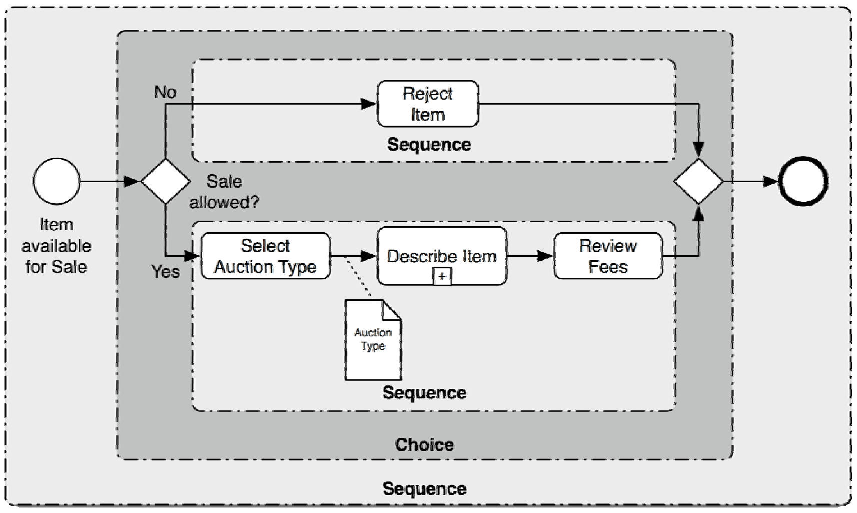
A process execution state shows the defined value of the executed process at a given point in time. In e-business, each workflow activity keeps a state like waiting, executed, or completed, as well as states related to variable assignment. In automotive, the execution of control functions depends on the states of the system. These states depend on variable assignments, as well as on the strategy of scheduling the process execution.

3.2 Control Flow

The control flow describes the topology of a process. The control flow can be refined to concrete control flow constructs like sequence, parallel, etc. as required by the specific domains. For example, in e-business, the control flow represents the behavioural aspect of workflows [3], for which a set of workflow patterns have been published in [6] and is widely used. Workflow patterns include for instance sequence, parallel split and join, discriminator, n-out-of-m join, multiple instances, milestone or cancel. In the automotive domain, control flow defines the (open/closed loop) process intrinsically. A control flow comprises for example sub-processes, which are executed in sequence or in parallel. An example of a control flow is coordinating concurrent torque requests, like traction control, gear-shift, or accelerator position. Another example can be seen in Figure 4, where a choice must be taken in order to control the engine’s optimal temperature.

Please note that the concepts shown in the conceptual model in the control flow rectangle constitute a subset of the mentioned workflow patterns. The control flow concepts are modeled as a specialization of the composite activity concept, allowing recurrent structures. One example of what the conceptual model allows through this specialization can be seen in Figure 5. The example uses the BPMN as a concrete modeling notation.

Figure 5: Control flow realization in BPMN



The control flow starts with a “Sequence” between the start end event with a “Choice” between. The control flow “Choice” contains the control flow “Sequence”, which in turn contains single activities and the composite activity “Describe Item”.

3.3 Data Flow

The concepts necessary for modeling data flow are: input, output, data sink, data source, process, composite activity, and activity (see Figure 2).

The data flow captures the passing of data within a process. This has been realized in the conceptual model through the relationship between the input and output concepts. This in turn implies that the data flow allows arbitrary connections between activities and composite activities.

In e-business, the data flow belongs to the informational workflow aspect [3]. An example is the forwarding of an “Auction Type” between “Select Auction Type” and “Describe Item” (see Figure 5). In automotive, data flow comprises transferring data from one activity or sub-process to another activity or sub-process, for example, transferring and computing of the target torque.

3.3.1 Input

Input describes data consumed by a process. In the conceptual model, each process, activity, or composite activity can have inputs assigned. A hierarchical structure of inputs (applies for outputs as well) is allowed in the conceptual model. This gives the possibility of grouping inputs/outputs when needed.

In e-business, input belongs to the informational workflow aspect [3]. Thereby each workflow activity might require incoming data to get started. An example is the “Item Description” that is required by the workflow activity “Review Fees” to get started (see Figure 5). In automotive, input data drive (control-) processes. They are provided as measurements of sensors or as output data from other processes. Examples of input data are signals from the clutch control or the accelerator sensor for adjusting the required torque.

3.3.2 Output

In the conceptual model an output describes data generated by the process, activities, or composite activities. In e-business, output also belongs to the informational workflow aspect [3]. Thereby each workflow activity might produce outgoing data as a result. An example is the selected “Auction Type” during the workflow activity “Select Auction Type” (see Figure 5). In automotive, output data constitute results of (control-) processes. They serve for driving actuators or as input for subsequent processes. Examples for output data are the value of the angular ignition spacing or the values for starting injection and the injection period.

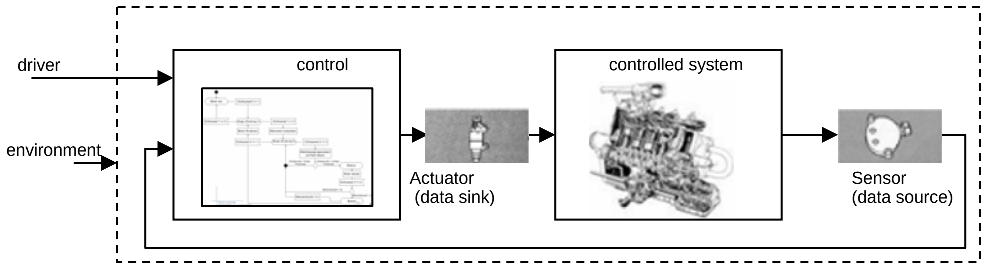
3.3.3 Data source

Data sources produce data used as input. An example is the official quotation of a stock index which might be available as a web service or a table in a database. In automotive, sensors are data sources and actuators are data sinks. In addition to regular data flow, where data are exchanged between (sub-) processes, data sources and data sinks serve for the asynchronous exchange of data (e.g., as memory).

3.3.4 Data sink

Data sinks consume data produced as outputs. In e-business and automotive, data sink is the complement to a data source. Examples of data sinks are databases that keep the orders. In automotive, data sinks are the actuators (see Figure 6).

Figure 6: **Data source and data sink in an embedded closed loop control system**



The conceptual model allows visualizing processes either as data sources or data sinks, since they produce/consume inputs/outputs as well. It is important to underline that even though the concepts are similar they need to be modeled separately. Especially, because data sinks and data sources are well-established concepts related to hardware (sensors, actuators) in the automotive domain. The main purpose of having such special concepts separated is to simplify the model's interpretation.

3.4 Environment

This section presents the set of concepts and relationships that were modeled in order to define the execution environment of a process.

3.4.1 Scope

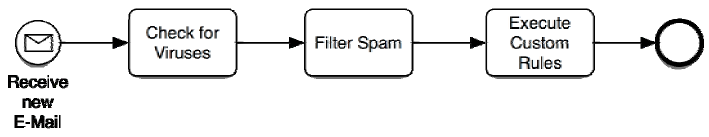
In the conceptual model, a process is related to a scope, where a scope represents an execution environment for a process/composite activity/activity. Due to the composite pattern, a process can have more than one scope associated. Events are caught within a scope and influence the set of associated variables by affecting their values and determining the process execution state at a point in time.

In e-business, an example is the scope of activities, where the user must be authenticated. The authentication data is kept inside the scope and events can be triggered for several reasons, for example, to specify a time-out. In automotive, engine and gear control are two control systems within the power train. Both control systems have their own scope. Communication between these two control systems is carried out by exchanging signals.

3.4.2 Event

An event can be thrown by a data source or a process. In the case of a data source, an event is thrown, when a special value or threshold defined for the data source has been reached. Something similar can be assumed for a process that reaches a certain state and/or exit criteria. Note that events can not be thrown, unless they are associated to a scope.

Figure 7: Message event example in BPMN [5]



An example from the e-business domain can be seen in Figure 7. Here, an e-mail processing workflow is triggered once a new e-mail is received. The e-mail is then checked for viruses, spam und further prepared regarding to custom rules.. In the automotive domain, events are usually classified as control events (e.g., for switching-on/-off the ignition) and data events (e.g., in case the value of a sensor under-/over-runs a threshold). Additionally, events can be classified as normal events (e.g., switch on/off) and temporal events (e.g. time synchronous throttle control). These can be modeled as well, as a specialization of the event concept.

3.4.3 Exception

One special kind of event is the exception. An exception can be used to define a possible error in the execution of a process.

In e-business, an exception signals an error in the default workflow execution. It can be handled by special exception handlers which are invoked by a Workflow Management System (WfMS) [5]. An example is an exception that is raised if information is invalid to allow further processing. In automotive, an exception is a condition, often an error, which causes a program or a microprocessor to branch in a different routine. An example could be the failure of a sensor, which results into an alternative control strategy, or a window-lift that stops closing the window once an object is blocked.

3.4.4 Variable

Variables provide the means for holding data. A variable is associated to a scope and a set of variables defines a state. Variables hold data of process attributes (i.e., inputs, outputs, data sources, data sinks, roles, and non-functional properties).

In e-business, variables belong to the informational aspect [3] of workflow instances. An example is an order identification number. In automotive, control processes require variables e.g., the engine's temperature.

3.4.5 State

A state is a situation during the execution of a process, which is characterized by the current variable assignments of its contained processes. In the conceptual model, this is reflected by the aggregation from the state concept to the variable concept, and through the relationship between the scope and the process concepts. For example, in e-business the state of a shopping transaction at a given time could be described by: the process -> “buy products”, the product ->“Book”, and the payment method -> “bank transfer”. In automotive the state of an intrusion detection process could be described by variables such as: lock -> “activated”, engine -> “idle”.

3.5 Non-functional Properties

Examples for non-functional properties are real time aspects, security, safety, or costs. In e-business, for instance, a non-functional property named costs could group several activities and decide at runtime which to choose depending on the current prices. In automotive embedded systems real-time aspects are essential. Besides the correctly computed result, it is crucial that the result is available in a timely manner. Numerous automotive control processes have real-time requirements, where time- and crankshaft-synchronous processes are differentiated. An example for a cyclic execution of process steps is determining the angular ignition spacing.

The conceptual model reflects these non-functional properties, by the concept named “Quality of Service”. This means that each process, activity, and composite activity can have properties attached that can be used as constraints for execution.

4 Integrating Variability in the Conceptual Process Model

This section presents the approach used for including variability in the conceptual model and thus represents the conceptual basis for modeling variant-rich processes in PESOA. First, the method selected as basis for building up the process variability approach will be explained. Additionally, an example of a process model illustrating the approach is presented.

The main ideas and thoughts used for elaborating this approach have been taken from [7]. It is geared towards the decision model based domain engineering. Another alternative, not considered here, is the feature based domain engineering approach [12], [26].

4.1 Selected Technique

Variability modeling is not a new concept in software engineering. It has been investigated and put into practice in the product line engineering area.

Figure 8: **Product line engineering two life cycle approach [1]. Refinement of Figure 1**

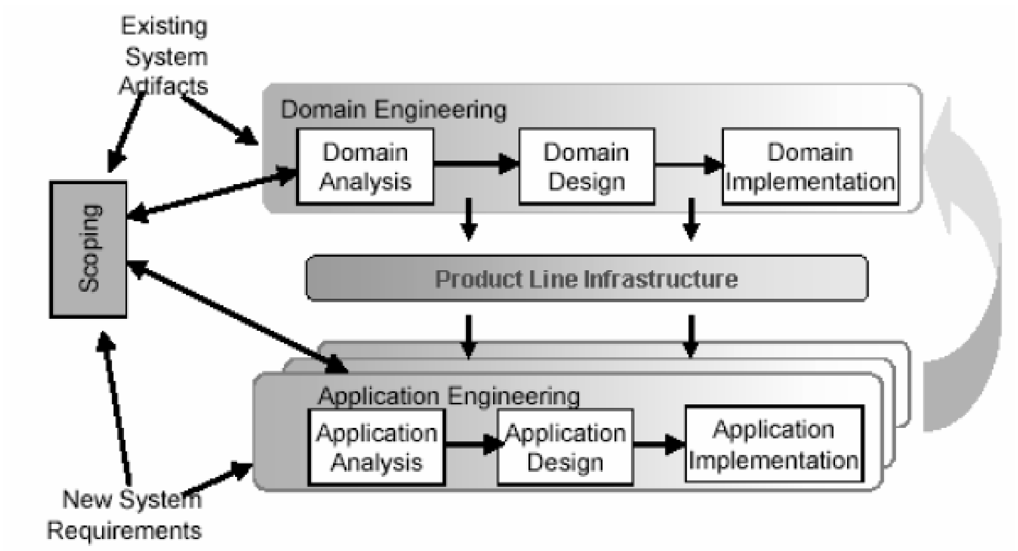


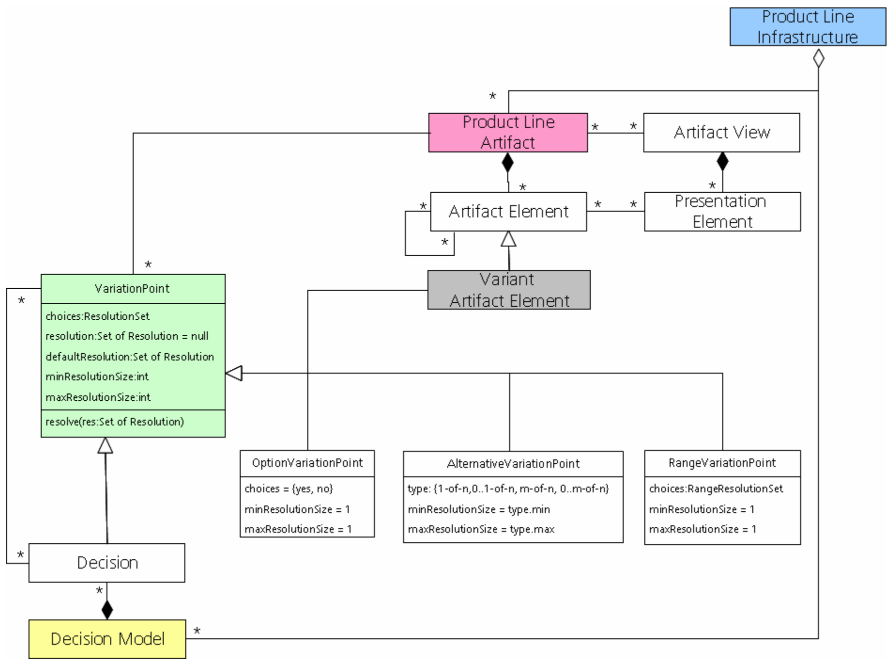
Figure 8 illustrates the product-line engineering approach. The first step is to define an appropriate scope for the domain. *Scoping* is done by using the features of a finite number of concrete (existing, planned, or future) products developed by a given organization.

Product line engineering focuses on comparing the characteristics of work products, systematically throughout the execution of a project, analyzing commonalities and variabilities. Commonalities identify the characteristics common to a set of products. They provide an enterprise with an understanding of the type of applications it produces. Variabilities describe the characteristics that vary from application to application. In other words, variabilities identify those characteristics that are uncommon to a set of products.

According to [7] the commonality and variability concepts can be captured in the product line artifact concept. Examples of product line artifacts are: Requirements document, design document, and source code. Product line artifacts can be generic or non-generic. Usually, non-generic artifacts represent commonalities between systems, while generic artifacts represent variabilities.

Figure 9 shows the used meta-model [7] (UML Static Structure), where it can be visualized how variability is related to the product line artifact concept.

Figure 9: **Product line infrastructure (UML static structure diagram)**



It can be observed that the variability of a product line artifact is represented through variation points. A variation point can be associated with choices. This means that selecting at least one of the choices will resolve the variability. The minimum and maximum numbers of choices that resolve a variation point as well as the default resolution choice are modeled as attributes. Finally, the resolve method is in charge of resolving the variation point by selecting a subset of the choices. This information is enough for modeling variability in product line artifacts. Some of the existing defined variation types are: option variation point, alternative variation point, and range variation point. The option variation point references information that is either relevant for a product line artifact or not. This means that there are only two choices and exactly one must be resolved. Alternative variation points represent the different combination of choices that can resolve the variation point. Finally, a range variation point represents a finite range of values that can resolve the variation point. Other sets of variation types can for example be found in [9] [10] [10].

Another important concept is the variant artifact element. It has been modeled as a concept that inherits from artifact element, and variation point. Modeling this concept through this mechanism provides a means for marking explicitly those artifact elements that are considered variable (i.e., those artifacts where variation points have been located)

A decision is a variation point that constrains the resolution of other variation points. They are used to document and structure variation points and relationships between them. The set of decisions is captured in a decision model. A decision can be of simple or non-simple nature. Simple decisions are those that do not constrain other decisions. This means that the variation can be resolved without expecting resolution from dependent variation points. The relationship between the decision and constrained variation points is called a resolution constraint. FODA's feature model can be considered a special view of a decision model that provides additional information on the feature (e.g., references to other sources of information) [7]. In the feature model, the composition captures the relationships among features, which corresponds to the constraints mentioned previously. This concept together with a more detailed description of decision models can be found in [12].

According to [7] decisions are defined through questions that are related to concepts from the domain. Such questions must be defined in such a way that they resolve the variation point. It is not the scope of this report to present details on how the variation points are resolved, but to provide an insight on the relationships between the concepts shown in Figure 9.

In the following, the application of the concepts presented above to the conceptual model presented is provided.

4.2 Conceptual Model Extension

In order to understand how to introduce the variability concept into the PESOA processes on a conceptual level, it is important to make an analogy between product line engineering and process family engineering.

Figure 10: **PESOA process – high level (UML activity diagram)**

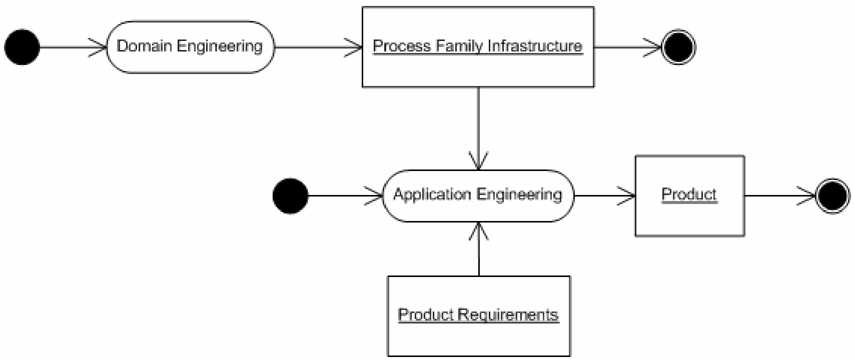


Figure 10 shows the PESOA process at a high level of abstraction by means of an UML activity diagram. The approach is based on the product line approach shown and explained previously (see Figure 1 and Figure 8). Domain engineering analyzes information on individual processes, integrates it by consideration of commonalities and variabilities, and stores the integrated information as part of the process family. Application engineering uses and specializes the integrated information according to the needs of a particular set of product requirements. In the following chapter a more detailed description of the domain and application engineering processes will be provided. Continuing with the analogy, it can be said that process family engineering focuses on finding the commonalities and variabilities of a set of processes in a given domain, and integrating them in a process family infrastructure. Workflow and embedded control processes are the main driving software engineering artifacts in PESOA. A process family infrastructure contains variant-rich processes and configuration models.

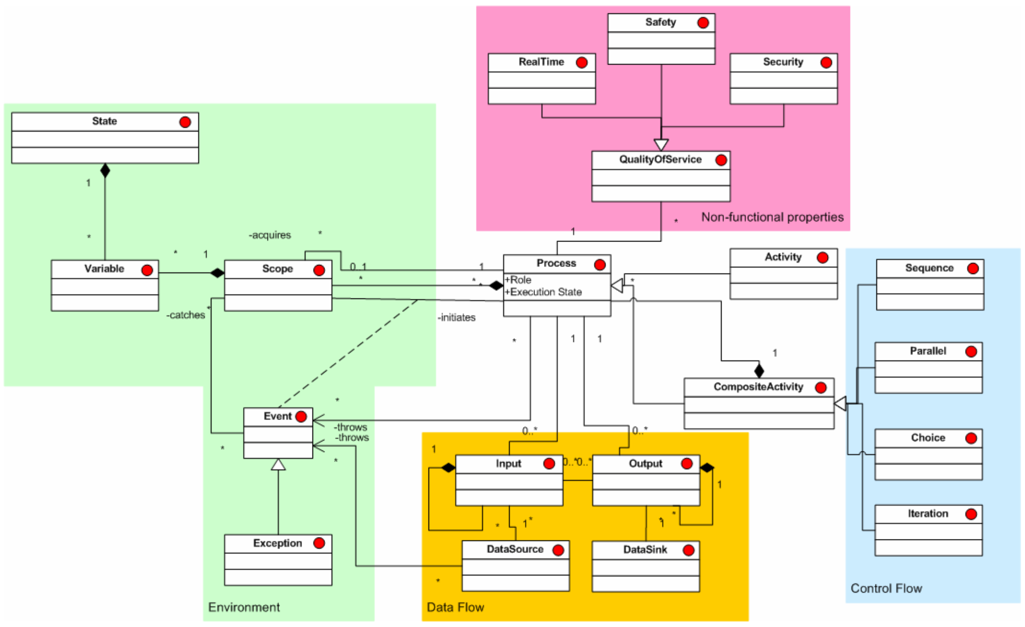
Table 1: **Mapping from product line engineering to process family engineering**

Product Line Engineering		Process Family Engineering	
Product Line Infrastructure		Process Family Infrastructure	
Product Line Artifact		Variant-Rich Process	
Artifact Element	Requirements, Design, Code.	Variant-Rich Process Element	Process and its descendants, Input, Output, Data Sink, Data source, Event and its descendants, Quality of service and its descendants, State, Variable, Scope, Event and its descendants

Product Line Engineering	Process Family Engineering
Decision models	Configuration models

Finally, a variant-rich process can contain variation points as well. Such variation points will represent the variability of variant-rich processes. Table 1 and Figure 11 shows how all the concepts of the conceptual model are considered variant-rich process elements (red-filled circles).

Figure 11: **Extended conceptual model (UML static structure diagram)**



Process: One example of process variability can be seen in online shops that support different mechanisms for paying an order. A process "Pay Order" contains a variation point that offers three choices to resolve the variability. Such choices can be: "Pay order with credit card", "Pay order with bank transfer", "Pay order per telephone". Depending on the choice selected by the user the "Pay Order" will be resolved. Please note that the concepts that inherit from process can contain variation points as well. They are: composite activity, activity, sequence, parallel, choice, and interaction.

Input/Output: Inputs and outputs can be variable as well. Continuing with the example, assuming a client has selected to pay with bank transfer, the input form might vary depending on the country where the bank belongs to. For example, American bank codes might differ from the European ones. In the case of outputs, the invoices to be generated might have several different representations that depend on the country where the order is to be delivered.

Data sink/Data source: In automotive, one example of a variable data sink can be observed in the one used by a sensor that detects an intrusion in the vehicle. Such data sink can record a different digital signal depending on the type of intrusion. On the other hand depending on the signal type, an actuator (a data source) will activate either an auto-dialler that calls a remote security centre, or it will turn on an alarm, or both. The variability will be resolved depending on the type of intrusion.

Quality of service: Regarding the quality of service defined in the conceptual model, they are susceptible of variability as well. In the case of real time as used in automotive, the processing of incoming events or internal interrupts may lead to varying execution times and processing sequences for the software functions. The external events and interrupts are watched by safety mechanisms that could for example close the gas valve in case of fire. In the e-business domain, different user security levels, that is, security variability will influence the privileges of a user in a secure transaction on an e-shop.

Scope: An example in e-Business is the scope of activities depending on the authenticated user. In a workflow administrators have a different scope than normal users. In automotive, the car navigation system can be considered optional and therefore turned-off. This means leaving out of the scope the processes that are relevant for accomplishing this functionality.

Variable: A variable can be a container of variation points. This can be observed in the e-Business domain in the case of services targeted to several languages. The values of variables must change depending on the language. The same applies for services targeted to different countries. In automotive safety requirements mandate the safe behaviour of the vehicle in the event of a failure or malfunction of a component, for example, by including a fail-safe mode in the system. They guarantee the operability of the vehicle to a certain degree: if, for example, a sensor should fail, default values may be assigned to its respective variables for the necessary computations (an example is the so-called lambda probe) [3].

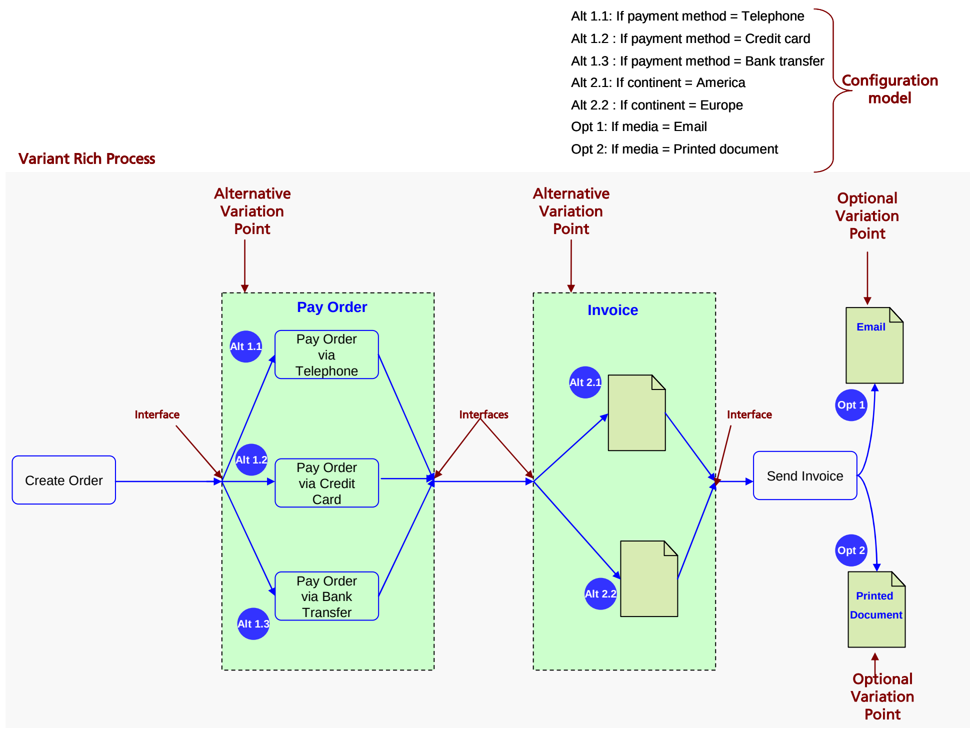
State: Based on the assumption that the states depend on the variable assignments it can be said that states can also contain variation points. For example in e-Business the state of a transaction might depend on the country legal requirements. In automotive, the state of the fail-safe mode process varies according to the sensor information that is being collected while the auto is on operation. For example a sensor could detect a non-normal situation and affect the fail-safe mode process execution.

Event: One example of variability in e-Business can be seen when the client exits a transaction. He can generate the exit event by pushing a button, or the event can be generated automatically by a timer. In the case of automotive, there are events that can be generated either mechanically or automatically. The car’s lights can be turned on automatically once a sensor detects the lack of light, or the driver can turn them on manually.

4.3 Example

Figure 12 provides an example that illustrates the concepts that introduce variability in the processes. It shows the flow between the “Create Order”, “Pay Order”, and “Send Invoice” processes of an online shop. The “Pay Order” process contains three alternatives (telephone, credit card, and bank transfer). The process has one interface that interacts with the create order process. At this point three alternatives split, and one of them must be chosen in order to resolve this variation. The resolution of the variation determines the path taken by the flow. The three alternatives converge in another interface that joins them. This interface is communicated with the product of the pay order process. The same can be observed in the case of the “Invoice” output that contains two alternatives (America, or Europe), and two interfaces. This means that depending on the continent of destination, the invoice to be sent to the customer will have different fields of information (e.g., currency, address). One variation point can be assigned respectively to the “Email” and “Printed document” outputs. Each have only two alternatives i.e., yes or no. Therefore, once the variation points are resolved, the client has the possibility of receiving the invoice via Email, as a printed document, or both. Please note that a variation point as modeled in Figure 9 has a default value.

Figure 12: **Variant-rich process example (BPMN extended notation)**



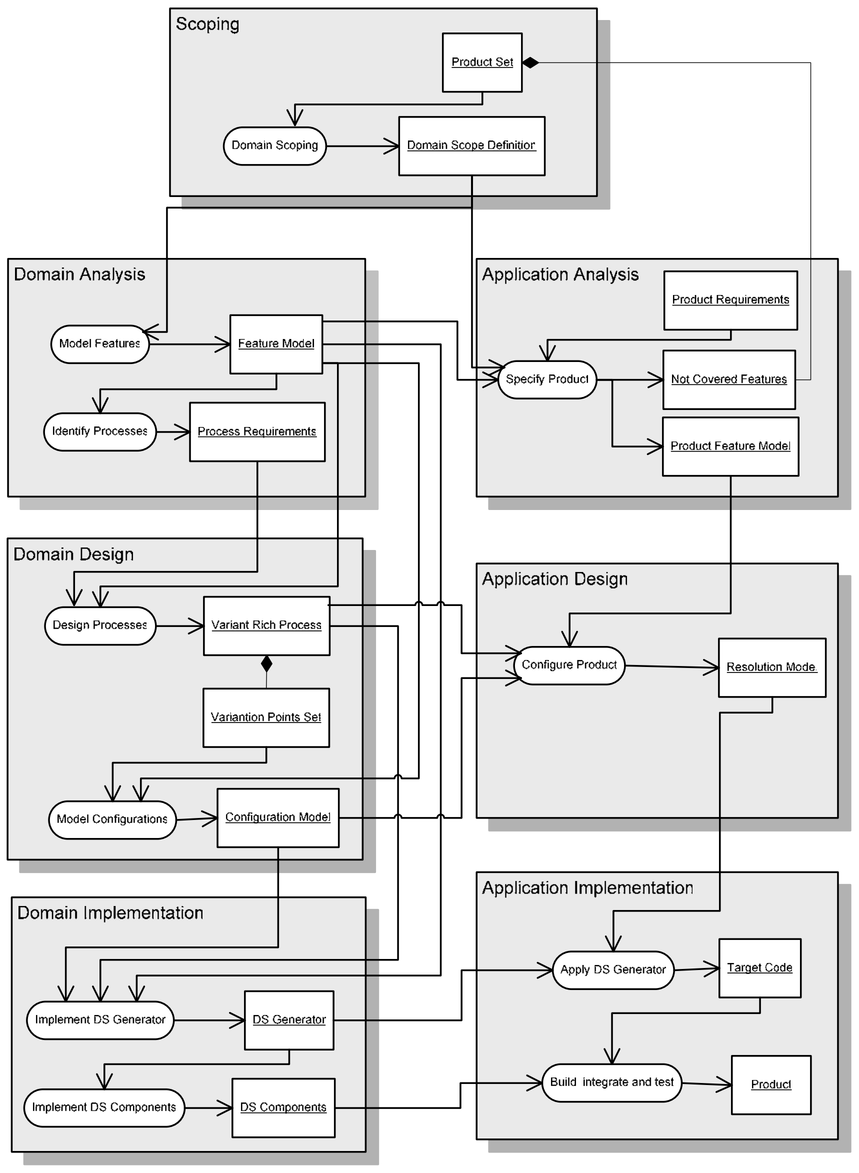
Alt 1.1: If payment method = Telephone
Alt 1.2 : If payment method = Credit card
Alt 1.3 : If payment method = Bank transfer
Alt 2.1: If continent = America
Alt 2.2 : If continent = Europe
Opt 1: If media = Email
Opt 2: If media = Printed document
Configuration model
Variant Rich Process
Alternative Variation Point
Alternative Variation Point
Optional Variation Point
Pay Order
Invoice
Email
Alt 1.1
Pay Order via Telephone
Alt 2.1
Interface
Interfaces
Interface
Opt 1
Alt 1.2
Create Order
Pay Order via Credit Card
Send Invoice
Opt 2
Pay Order via Bank Transfer
Alt 2.2
Printed Document
Alt 1.3
Optional Variation Point

Examples that show the realization of these concepts in the e-business and the automotive domain will be provided in chapter 6.

5 PESOA Process

This section presents in more detail the PESOA process. Figure 13 presents a product flow view of the PESOA process.

Figure 13: **PESOA process (UML activity diagram)**



In the following section, the processes and products are described as well as the product flow among them.

Each process is described as follows:

- Purpose: What has to be done?
- Description: How shall it be done?
- Input criteria: List of products needed for the process
- Output criteria: List of products produced by the process
- Product flow: Which are the products consumed and/or produced by the process

Each product is described as follows:

- Purpose: What is the product intended for?
- Description: What are the contents of the product?
- Possible notation: How can the product be described?

5.1 Domain Scoping

Purpose

The purpose of this process is to determine the appropriate bounds of the process family infrastructure [7].

Description

Scoping shall be based on the premise that one shall obtain as much return on investment as possible from effort of establishing a process family infrastructure. Using as input an existing or a planned set of process family infrastructure products a subset of such products is selected. Afterwards, the selected products are mapped to the features that they should offer. This information is recorded in a domain scope definition. In product line engineering such scope definition is often documented as product map [7].

Input Criteria

An existing or planned set of products

Output Criteria

A completed domain scope definition

5.2 Domain Analysis

5.2.1 Model Features

Purpose

The purpose of this process is to model the features to be part of the process family infrastructure.

Description

The domain scope definition is used as input for identifying *consists-of* relationships among features. Afterwards, such relationships must be described in a model e.g., a hierarchical structure [25], or a tabular representation [2].

Input Criteria

A completed domain scope definition

Output Criteria

A completed feature model

5.2.2 Identify Processes

Purpose

The purpose of this process is to define the set requirements for those processes that will constitute the process family infrastructure.

Description

The feature model can be used as basis for identifying and documenting the requirements for those processes that will be part of the process family infrastructure. Such processes shall be conceived as unique building blocks in order to reuse them with no major problems.

Input Criteria

A completed feature model

Output Criteria

A completed description of processes requirements

5.3 Domain Design

5.3.1 Design Processes

Purpose

The purpose of this process is to model the existent commonalities and variabilities among the previously identified processes.

Description

Using as input the list of identified processes, a commonality analysis among processes shall be performed in order to identify variant-rich process elements. At the moment there are not many techniques or approaches on how to perform such a comparison. One idea can be taken from [8], where a systematic comparison of a set of software process models is illustrated. The commonalities and variabilities detected among variant-rich process elements are then integrated into their respective variant-rich process.

Input Criteria

A completed list of processes

Output Criteria

An integrated set of variant-rich processes

5.3.2 Model Configurations

Purpose

The purpose of this process is to establish the dependencies between newly modeled variation points, as well as among new variation points and existing variation points.

Description

Initially, relationships among new variation points are identified and documented in what is called the configuration model. Afterwards, the model is updated with relationships between new variation points and existing ones, which in some cases can produce a new high-level relationship that groups them.

Input Criteria

An integrated set of variant-rich processes

Output Criteria

A configuration model

5.4 Domain Implementation

5.4.1 Implement DS (Domain-specific) Generator

Purpose

The purpose of this process is to implement a mapping from configurations for variant-rich processes to their implementation.

Description

Based on the commonalities and variabilities contained in the variant-rich process, the domain-specific functionalities to be covered by the generator are identified. Code fragments implementing these functionalities are defined. They are connected to the process' variabilities, i.e. each variation point is annotated by one or more code fragments. Dependencies and constraints given by a configuration model and a feature model are considered for this implementation process.

Input Criteria

A completed variant-rich process.

A completed configuration model.

A completed feature model.

Output Criteria

A completed DS (domain-specific) generator.

5.4.2 Implement DS (Domain-specific) Components

Purpose

The purpose of this process is to implement generic components which are either part of the resulting application or used for building it.

Description

There are two kinds of domain-specific generic components:

Based on the implementation of a domain-specific generator, components that are needed to process the generator's output are identified. They are referred to as infrastructure components. Infrastructure components that are specific for the selected domain are being identified and implemented within this process.

Based on the commonalities contained in the variant-rich process and the functionalities already covered by a domain-specific generator, the functionalities are identified which are to be implemented by generic components. Such components are referred to as runtime components. Runtime components that are specific for the selected domain are being identified and implemented within this process.

Input Criteria

A completed variant-rich process.

An existing DS (domain-specific) generator.

Output Criteria

Completed DS (domain-specific) components.

5.5 Application Engineering Processes

5.5.1 Specify Product

Purpose

The purpose of this process is to specify a new product based on the scope definition of the existent reusable process family infrastructure.

Description

New market/customer requests are used as input for specifying a new product. Such requests must be analyzed using as a basis the existing process family scope definition, and the feature model. Those features that are estimated to be realizable are mapped to the actual products from the process family infrastructure. Such mapping must be documented in a product feature model. Those features that are not yet planned in the process family shall be documented in a list of not covered features, which will be later integrated in the scope of the process family.

Input Criteria

A completed domain scope definition

A completed set of product requirements

A completed feature model

Output Criteria

A completed product feature model

A completed list of not covered features

5.5.2 Configure Product

Purpose

The purpose of this process is to configure the product to be instantiated from the process family infrastructure.

Description

The list of features to be implemented characterizes the new product. In order to reuse the existing process family, such characteristics must be identified. The process family is configured regarding the product characteristics by using the configuration model. The configuration model allows resolving the variation points based on the product features. The resolution of the variation points and their relationships are documented in the resolution model.

Input Criteria

A completed product feature model

A completed configuration model

An integrated set of variant-rich processes

Output Criteria

A resolution model

5.5.3 Build, Integrate and Test

Purpose

The purpose of this process is to perform the final implementation work, consisting of building, integrating and testing the product.

Description

The generated target code is subject to further processing by the use of infrastructure components, including domain-specific ones. The resulting executables have to be integrated with the needed runtime components, which comprise domain-specific ones as well. Together they form the product, whose implementation is completed by testing it.

Input Criteria

Completed target code.

Completed (DS) domain-specific components.

Output Criteria

Completed product

5.5.4 Apply Domain-specific Generator

Purpose

The purpose of this process is to automatically perform the mapping of resolved variation points from variant-rich processes to concrete target code.

Description

The appliance of the domain-specific generator starts with importing data from a process or a resolution model, followed by triggering the generation of target code. If there are additional variabilities that are not part of the variant-rich process, e.g. technical ones specific for the target platform, they can be configured and resolved before triggering the code generation.

Input Criteria

A completed (DS) domain-specific generator.

A completed process and a completed resolution model, respectively.

Output Criteria

Completed target code.

5.6 Products

5.6.1 Product Set

Purpose

The purpose is to contain the set of existing or planned products of the organization’s process family infrastructure.

Description

The product set can be described as a repository of products. Each product is generally described with its purpose, applicable context, and representation (e.g., diagram, model, code).

Possible notation

The notation of the elements of the product set depends on its nature. For example: Requirements might be specified with normal prose or a standard could be used (e.g., IEEE Std 830-1998; IEEE recommended practice for software requirements specification).

5.6.2 Scope Definition

Purpose

The purpose of this product is to hold the relationships between the product set and their features.

Description

The scope is defined by the relationships among a set of existing and planned features and the product set.

Possible notation

According to [7] the scope can be described with a table like the following:

Table 2: Product map

Features	Products			
	Product 1	Product 2	Product …	Product n
Feature 1	x		x	x
Feature 2		x		
Feature …	x		x	
Feature n	x	x		x

Other notations for describing the scope can be found in [15].

5.6.3 Feature Model

Purpose

The purpose of the feature model is to represent a set of features and their inter-relationships.

Description

The feature model captures the functionality expected to be found in the process family infrastructure.

Possible notation

Feature models can be represented by using hierarchical based diagrams, where the root node represents the basic concept or product and the leaves represent the single features as proposed by the FODA approach [25].

5.6.4 Process Requirements

Purpose

The purpose of this product is to hold the requirements of processes that belong to a process family.

Description

Each process requirements must be described with the following attributes:

- Purpose: What is to be done by the process?
- Description: How should it be done?
- Inputs: What is the list of needed inputs?
- Outputs: What is the list of needed outputs?
- Data sinks: What is the list of produced data sinks?
- Data sources: What is the list of consumed data sources?
- Scope: Which is the environment in which the process will be executed?
- Quality: Which are the quality attribute constraints?

Possible notation

The process requirements can be described using tables that comprise all the previous information.

5.6.5 Variant-Rich Process

Purpose

The purpose of this product is to hold the processes that make up the process family. These contain the commonality and variability relationships among variant-rich process elements.

Description

Variant-rich processes are processes that conform to the conceptual model presented in chapter 3. A variant-rich process can be defined as a process family infrastructure element and the variation points it contains.

Possible notation

Variant-rich processes in the PESOA domains are represented as described in chapter 6.

5.6.6 Variation Points Set

Purpose

The purpose of this product is to hold the variation points of each variant-rich process element.

Description

A variation point contains a set of choices that resolve the variability of a variant-rich process element. A set of variation points are the union of all the variation points contained in a variant-rich process element.

Possible notation

The variation point concept is represented in the PESOA domains as described in chapter 6. Please note that in Chapter 6, the variation point has been separated from its choices (choices are called variants in Chapter 6).

This separation is needed for representing graphically, either in UML Activity Diagrams, UML State Machines, or BPMN, the variability mechanism (e.g., extension, parameterization, and inheritance) that realizes such variabilities (i.e., alternatives, options, or ranges).

5.6.7 Configuration Model

Purpose

The purpose of this product is to hold the relationships among variation points in a process family infrastructure.

Description

A configuration model can be defined as a set of dependencies. Each one is a variation point that constrains other variation points and that can be explicitly related to a domain concept.

Possible notation

A configuration model can be captured in a table. Using as an example the variant-rich process shown in Figure 12, the following is an example:

Table 3: Configuration model of the Pay order variant-rich process

Variant-rich process	ID	Name or question	Type	Variant-rich process element	Constrains
Pay order	1.1	Pay order via telephone	Alternative	Process	-
Pay order	1.2	Pay order via credit card	Alternative	Process	-
Pay order	1.3	Pay order via bank transfer	Alternative	Process	-
Pay order	1	Which mechanism is used to pay the order?	Alternative-Decision	-	Resolves variation points 1.1 to 1.3

5.6.8 Domain-specific Generator

Purpose

The purpose of this product is to map resolved variation points from variant-rich processes to concrete target code implementing the configured features.

Description

A domain-specific generator consists of target code fragments related to variation points from a variant-rich process, based on a generation technique providing an interface to apply the generator. All dependencies and constraints related to the target code are implemented in the domain-specific generator.

Possible notation

The domain-specific generator can be implemented using a model-based generator development approach. In this case, code fragments are connected to a generator meta-model. The code fragments are defined using the target code language, enriched by certain constructs defining meta-model connections and code-related constraints. The meta-model can be defined using the OMG MOF (Meta Object Facility) standard.

5.6.9 Domain-specific Components

Purpose

The purpose of this product is either to serve as a generic part of the resulting application or to be used for building it.

Description

Domain-specific components are infrastructure or runtime components. Infrastructure components are tools, frameworks or libraries used for building the application. Runtime components, e.g. runtime libraries, implement some common functionality of the resulting applications in a generic way.

Possible notation

The notation of domain-specific components is specific for the target platform and can be text or binary code.

5.6.10 Product Requirements

Purpose

The purpose of this document is to hold the new requests of the customer or the market.

Description

This is a description of the customer or market features to be fulfilled by a new product.

Possible notation

Requirements could be specified with normal prose or by following any standard such as e.g., IEEE Std 830-1998; IEEE recommended practice for software requirements specification.

5.6.11 Product Feature Model

Purpose

The purpose of this product is to hold the set of features to be implemented in the new product.

Description

For each feature to be implemented the following attributes must be described:

Purpose: What is to be done by the feature?

Quality: Which are the quality attribute constraints? e.g., performance issues

Possible notation

The features can be described using tables.

5.6.12 Not Covered Features

Purpose

The purpose of this product is to hold those features that could not be instantiated or derived from the existing process family.

Description

Features that are not yet covered in the existing process family, but that shall be considered must be described as follows.

Purpose: What is to be done by the feature?

Quality: Which are the quality attribute constrains?

This information will then be used as input for a future process family scoping.

Possible notation

The features can be described using tables.

5.6.13 Resolution Model

Purpose

The resolution model records the configuration information for a specific product.

Description

The resolution model documents the set of choices that led to the set of features and processes encapsulated by a specific product. This is essential for being able to maintain the product.

Possible notation

The resolution model captures the choices taken in the configuration model. The notation is, therefore, dependent on the notation of the respective configuration model.

In the case of tabular configuration models, as in the example above (Table 3), the resolution model contains the answers to the questions in the configuration model.

When generators are used, the resolution model additionally contains the configuration for that generator. Then, XMI is a possible notation.

5.6.14 Target Code

Purpose

The purpose of this product is to implement the configured features, i.e. the resolved variation points of the variant-rich process.

Description

The target code is specific for the target platform as well as for one resolution model and one concrete, i.e. not variant-rich, process.

Possible notation

The notation of the target code may be any conventional programming language or even a platform-specific execution language. In any case it is program text, not binary code.

5.6.15 Product

Purpose

The purpose of this product is to hold the instantiated variant-rich process, and the single products that might be developed from scratch.

Description

This is the final product to be delivered to the customer.

5.6.16 Process Family Infrastructure

Purpose

The process family infrastructure contains all information and tools to develop, use and maintain the product set.

Description

The process family infrastructure consists of the variant-rich processes, the feature model, and the configuration model, and the different resolution models for the different instantiated products.

6 Modeling Variant-Rich Processes in the PESOA Domains

The objective of this section is to describe the integration of the representation of variant-rich processes in the PESOA application domains into the overall PESOA development process.

To this end, the concepts described in 3 are represented by the process descriptions in the PESOA application domains. This means especially that the concepts captured by the conceptual model are mapped to the notations used in the respective domains. The representation of the concepts relevant in the automotive domain with UML Activity Diagrams [17], [18] and UML State Machines [18], [19], [20] has already been sketched in [2] and will be enhanced in this section according to the development of the conceptual model for PESOA processes, which reflects the concepts for the PESOA application domain as well as their interrelations.

For modeling variability in automotive and e-business processes variability mechanisms are applied, which are described extensively in the PESOA technical report “Process Family Engineering – Variability Mechanisms” [16] and summarized in this section.

6.1 Modeling Variant-Rich Software Based Automotive Control Processes

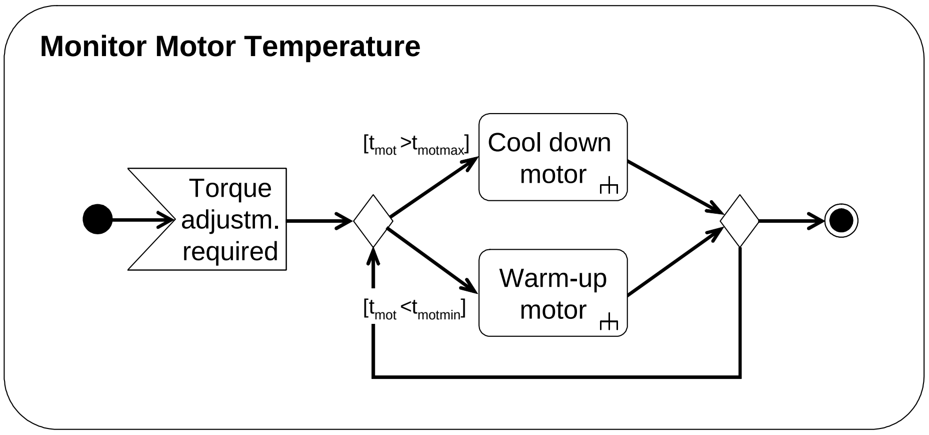
The first subsection (6.1) describes the representation of the concepts described in the PESOA conceptual model by means of UML Activity Diagram (6.1.1) and UML State Machine (6.1.2) constructs. The second subsection (6.2) shows how variant-rich Activity Diagrams and State Machines can be modeled by means of variability mechanisms.

6.1.1 Mapping of Relevant Concepts to UML Activity Diagrams

In the PESOA conceptual model single execution steps as well as sub-processes, which are constructed using the routing constructs for modeling sequences, alternatively, parallel or iteratively executable behaviour are regarded as processes.

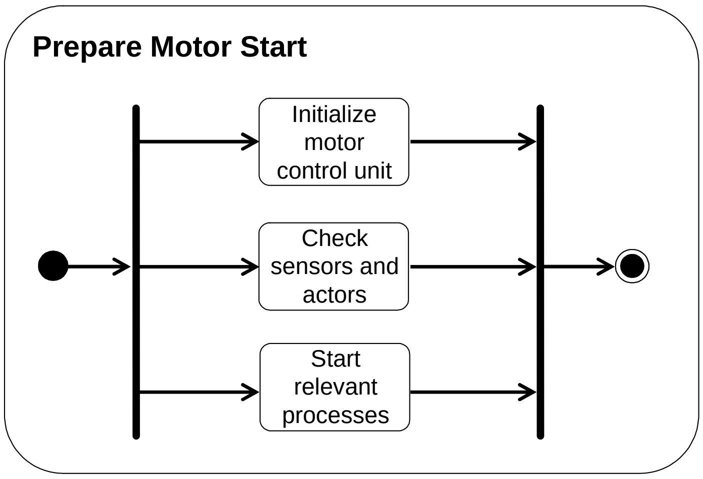
In the conceptual model sub-processes are represented as composite activities. In UML Activity Diagrams sub-processes are modeled using activities. The top level activity in the potential activity hierarchy represents the entire process. Figure 14 shows an example for the Activity Diagram process “Monitor Motor Temperature”. As indicated by a small rake-style symbol the actions “Cool down motor” and “warm-up motor” can be decomposed into a sub-process illustrated in a separate Activity Diagram.

Figure 14: **Example for Activity Diagram processes and some basic routing constructs**



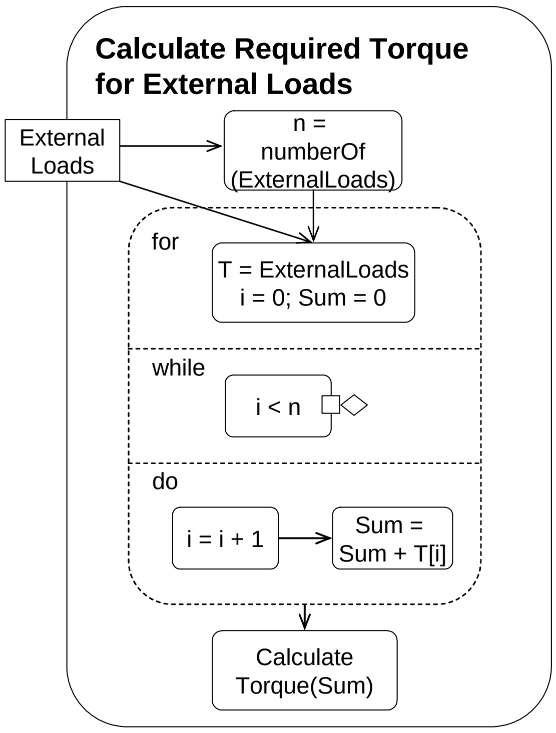
Concerning the control flow Activity Diagrams provide elements for modeling the start and end of a process as well as elements for modeling parallel and alternative flows, which represent the most fundamental routing constructs in processes. Figure 14 gives an example for an initial node and final node, which indicate the start and endpoint of the process as well as a decision and merge node initiating and terminating two alternative flows. Figure 15 illustrates how to model parallel flows (“Initialize motor control unit”, “Check sensors and actors” and “Start relevant processes”) using fork and join nodes.

Figure 15: **Example for modeling parallel flows in Activity Diagrams**



In basic activities iterations can be realized using a combination of re-entering arcs and decision and merge nodes. An example for this is shown in Figure 14. Alternatively with loop nodes and expansion regions structured activities offer elements dedicated solely for modeling iterative behaviour. Figure 16 gives an example for the application of a loop node for calculating an overall external load.

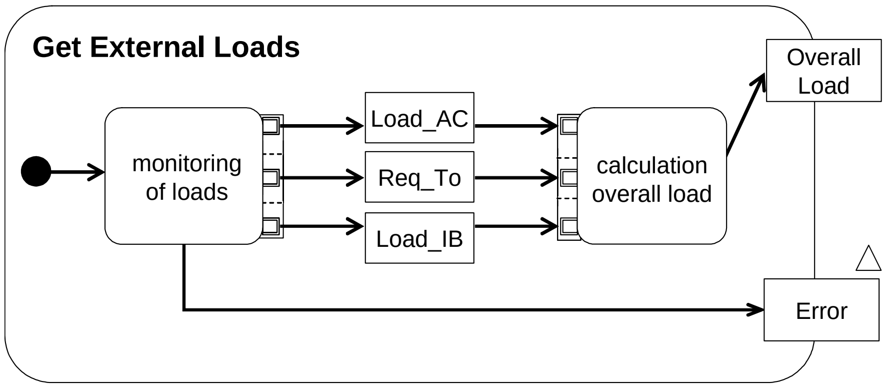
Figure 16: **Example for the application of a loop node**



Calculate Required Torque for External Loads
External Loads
n = numberOf (ExternalLoads)
for
T = ExternalLoads i = 0; Sum = 0
while
i < n
do
i = i + 1
Sum = Sum + T[i]
Calculate Torque(Sum)

Concerning the representation of data flow Activity Diagrams offer pins for modeling the input and output data of actions and activity parameter nodes for the input and output data of activities. Additionally, complete activities provide parameter sets for modeling mutually alternative sets of input or output parameters. In order to indicate the name or type of a parameter the respective information can be written close to the corresponding modeling element representing the input or output parameter. Executable nodes like actions and activities or object nodes like central buffer nodes in intermediate activities and data store nodes in complete activities are possible data sinks and sources in Activity Diagrams. A data store node is a modeling element for indicating the persistent storage of data. Figure 17 exemplifies the representation of data flow in Activity Diagrams. The activity parameter node “Overall Load” provides subsequent activities with the required data for adjusting the torque. The “monitoring of loads” action either forwards the requested torque and the external air condition load (“Req_To” and “Load_AC”) or the requested torque together with the external icebox load (“Req_To” and “Load_IB”) using two parameter sets.

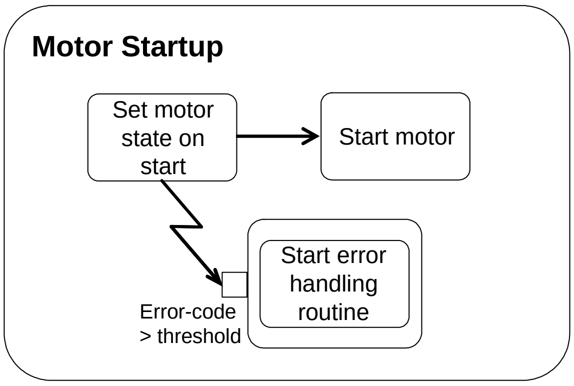
Figure 17: **Example for modeling data flow in Activity Diagrams**



A classifier is what best reflects a scope in UML Activity Diagrams. Additionally, in structured activities an activity may contain structured activity nodes, which have their own scope. Events in Activity Diagrams can be raised using send signal actions. An accept event action on the other hand is a type of action that is activated as soon as an event of the indicated type occurs. An accept event action can be triggered for example by a signal sent by a send signal action or by a time event. Figure 14 gives an example for an accept event action ("Torque adjustm. required").

In Activity Diagrams exceptions can be used to indicate deviations from the normal flow of control. Exceptions can be handled locally using exception handlers (extra structured activities). This is shown in Figure 18, where an exception handler initiates an error handling routine in case an error occurs while setting the motor state on start.

Figure 18: **Example for the application of an exception handler**



Alternatively, exceptions can be forwarded via certain output parameters indicated by the is-exception attribute (complete activities). This is shown in Figure 17, where an exception is thrown and forwarded via the "Error" is-exception-activity parameter node in case some locally unsolvable problems occur during the monitoring of the loads.

Variables in Activity Diagrams are used for example in guards, in decision input behaviours for making routing decisions, in selection behaviours (in complete activities) for reading data from a data store node, in the description of local preconditions or local post conditions of actions (in complete activities), or in the form of parameter names of pins or activity parameter nodes.

Concerning the representation of states, in Activity Diagrams there are no modeling elements explicitly provided for the representation of states. However, during execution the state of an Activity Diagram can be derived from the token distributions within the Activity Diagram and the values of the variables described in the process. Thereby, the execution state of the highest-level activity can be considered as the execution state of the system (concept system execution state) while the execution state of the sub-processes contained in the Activity Diagram correspond to the process execution state.

The quality of service properties safety and security, which have been newly added to the PESOA conceptual model cannot be represented explicitly in an UML State Machine or Activity Diagram. Instead these non-functional properties are typically reflected in the structure of the process. For example, a secure process should of course also be free of deadlocks. Therefore, the process has to be designed correspondingly and deadlock-freedom cannot be provided by a single element. Thus, no dedicated elements can be identified, which provide the non-functional process properties.

6.1.2 Mapping of Relevant Concepts to UML State Machines

In UML State Machines a process is represented by a State Machine. The State Machine can contain composite and submachine states, which encapsulate sub-processes. Sub-processes can be further divided into orthogonal regions containing sub-processes that are carried out in parallel. Some examples for simple states (e.g. “Error”) as well as for a submachine state (“Stop”) are given in Figure 19. The entire State Machine represents the overall process, while the submachine depicted in Figure 20, which is invoked by the submachine state “Error”, is an example for an encapsulated State Machine sub-process. It consists of two parallel executable regions.

Figure 19: Exemplary UML State Machine process

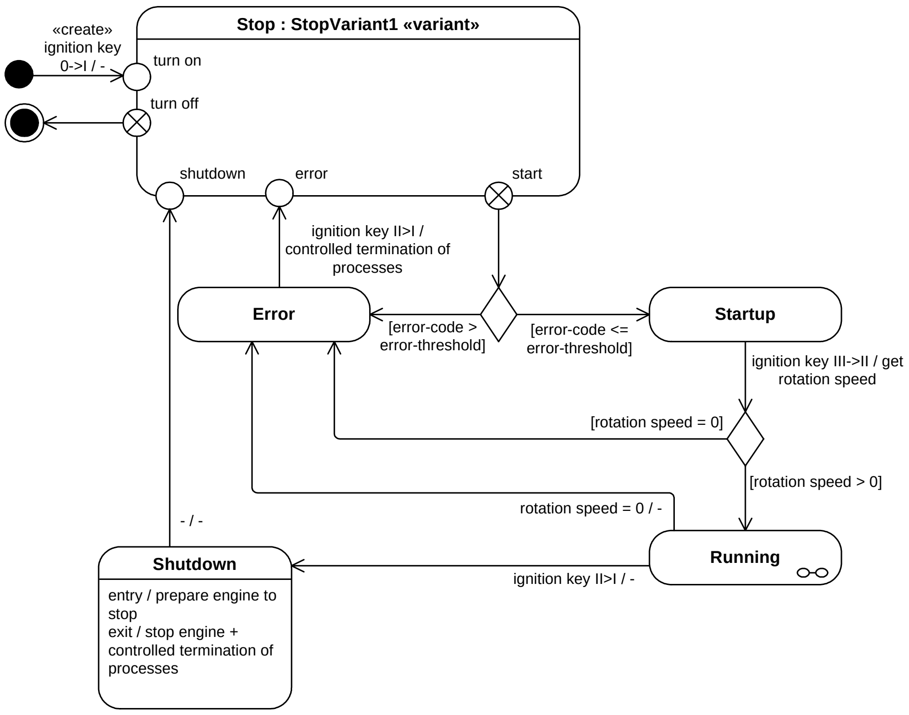
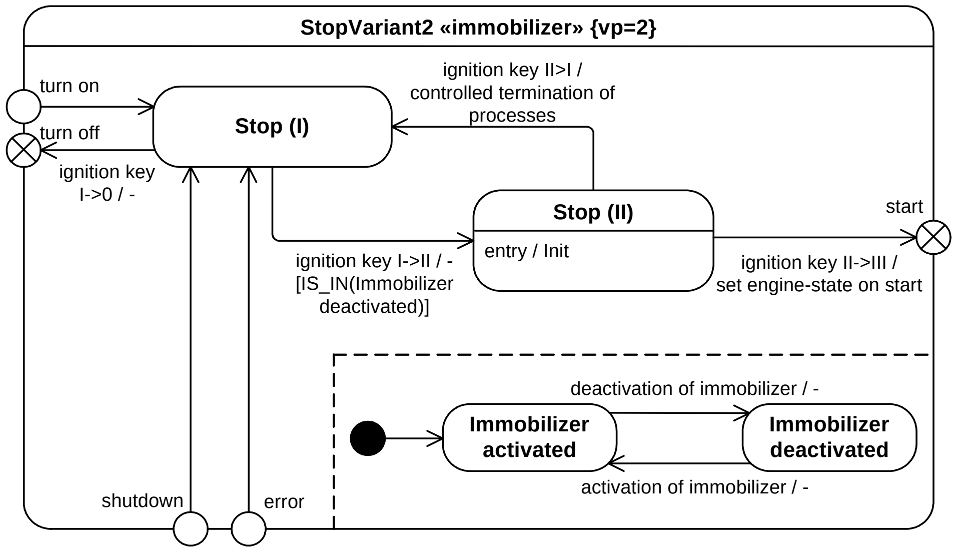

Stop : StopVariant1 «variant»
«create»
ignition key
0->I / -
turn on
turn off
shutdown
error
start
ignition key II>I /
controlled termination of
processes
Error
[error-code >
error-threshold]
[error-code <=
error-threshold]
Startup
ignition key III->II / get
rotation speed
[rotation speed = 0]
[rotation speed > 0]
rotation speed = 0 / -
- / -
Running
ignition key II>I / -
Shutdown
entry / prepare engine to
stop
exit / stop engine +
controlled termination of
processes

Figure 20: **Exemplary submachine**



StopVariant2 «immobilizer» {vp=2}
turn on
turn off
Stop (I)
ignition key II>I / controlled termination of processes
ignition key I->0 / -
Stop (II)
entry / Init
start
ignition key I->II / - [IS_IN(Immobilizer deactivated)]
ignition key II->III / set engine-state on start
deactivation of immobilizer / -
Immobilizer activated
Immobilizer deactivated
activation of immobilizer / -
shutdown
error

Regarding the control flow, which is represented explicitly, State Machines provide so called pseudo states for modeling initial states, final states, and the beginning and end of alternative and parallel processing. The latter is supported by orthogonal state regions, which can be active at the same time. Loops can be modeled using re-entering transitions. An example of alternative flows initiated by a choice pseudo state can be found in Figure 19, where depending on an error-code either the state “Error” or “Startup” is entered.

State Machines are not designed for the explicit modeling of data flows. Nevertheless there are several ways for integrating actions and activities in State Machines. A state in a State Machine may for example have entry and exit activities which are executed when entering or leaving the state. This is shown in Figure 19, where the action “prepare engine to stop” is carried out when state “Shutdown” is entered. Once the state is left actions are carried out that stop the engine and terminate the motor processes. Moreover there are states which are characterized by the execution of an activity. After having finished the execution of the activity the state is normally left via a guard less transition. Additionally activities may be assigned to a transition. These Activities are executed when the transition fires. An example for this can be found in Figure 19 where the action “get rotation speed” is carried out when event “ignition key III->II” occurs in state “Startup”. The activities integrated into a State Machine correspond exactly to the activities as specified in the Activity Diagram meta-model. Hence they can also have input and output parameters of a specific type. In this spirit activities in State Machines are also data sinks and sources even though the data flow isn't modeled explicitly. Therefore, no elements for representing the persistent storage of data are provided either.

A State Machine describes a classifier, which corresponds to the scope of the State Machine. It scopes the triggers and attributes available to the State Machine. Events are one of the core concepts in State Machines. Triggers define the types of events that may occur leading to transitions between states. There are triggers that allow for a transition in reaction to an asynchronous signal, triggers for transitions resulting from the invocation of an operation, time triggers specifying a deadline at which a transition must take place, and triggers recognizing the change in the values of some attributes of the object modeled by the State Machine. The change of attribute values is recognized by evaluating Boolean expressions. Figure 19 shows some examples of triggers. The trigger "ignition key II>I" for example leads to the transition from state "Error" to state "Stop". In contrast to Activity Diagrams State Machines don't support exceptions. In State Machines variables can be used for example in input and output parameters of Activities or in transition guards. Examples for variables in guards are shown in Figure 19. For example, the transition from "Startup" to "Error" is only performed if the variable "rotation speed" has a value equal to zero. As the name suggests in State Machines one major modeling concept are different forms of states. The state of a State Machine (process execution state) during execution results from the combination of the states concurrently active in the State Machine, from the values of the variables defined in the State Machine and from the activities currently active in the State Machine. The system execution state is the execution state of the top level State Machine.

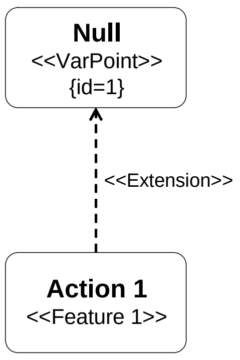
6.1.3 Modeling Variability in Software Based Automotive Control Processes

In order to describe variations within a process model, the process model has to contain the following information:

- the point in the process model where the variability occurs, i.e. the variation points have to be identified
- which variants can be bound to the variation point in dependence of the product feature to be implemented
- how the variants are bound to the variation point, i.e. the variability mechanism to apply for realizing the system variability

Variation points in UML State Machines and Activity Diagrams are identified using the stereotype <<*VarPoint*>> and a corresponding tagged value "vp" that assigns a unique identification number to the variation point. Variants are connected to a variation point using UML dependency relations. This corresponds to the association of choices with a variation point as generically described in 4. Every variant disposes of a stereotype that relates the variant to the product feature that is (partly) implemented by the variant. For UML State Machines and Activity Diagrams a set of variability mechanisms have been defined in [16], which allow for a flexible implementation of variability into the process model. Variability mechanisms are referred to generically as resolve methods in 4. An example for the notation of variability in Activity Diagrams is shown in Figure 21. In this case "Action 1" is the variant which can be bound to the variation point represented by a null-activity. For binding "Action 1" to the variation point the variability mechanism "Extensions" is used as indicated by the respective stereotype assigned to the dependency relation connecting the variation point with the variant. "Action 1" is used to implement "Feature 1" as suggested by its stereotype.

Figure 21: **Notation of variability in Activity Diagrams considering an extension as example**



In order to indicate the correlation to the developed concepts for modeling variant-rich processes in PESOA product line development process, the variability mechanisms described in [16] are categorized into variability mechanisms for realizing option, alternative and range variation points as described in chapter 4.This is analyzed for every variability mechanism in the remainder of this subsection.

Encapsulation of sub-processes. Encapsulation of sub-processes allows for hiding variant sub-processes behind an invariant interface. Thereby, a member of a predefined set of sub-process variants with a compatible interface can be inserted as an interface implementation thus supporting alternative and optional (i.e. with "real"- and "default"-variants) variation points.

Replacement/omission of encapsulated sub-processes supported by **interface separation**. The omission of encapsulated sub-processes allows for realizing optional variation points, while by replacing sub-processes alternative variation points can be realized.

Parameterization. Parameterized sub-process paths can be used for realizing optional or alternative variation points. Using parameterization a sub-process path can be activated/deactivated for example. In this case parameterization is used for realizing optional variation points. Alternatively, there might also be a set of alternative paths, from which only one is activated if a corresponding parameter is set. In this case an alternative variation point has been represented. Parameterization also allows for parameterizing encapsulated sub-processes. The alternative parameters allowed for configuring the sub-process represent a range variation point.

Templates. Templates refers to the techniques that support variability in data types. The data flow between activities may be optional (optional variation point) or only certain data types may be exchanged (alternative variation point).

Inheritance. Inheritance comprises the addition and replacement of sub-processes or individual elements. This allows for realizing alternative variation points.

Design patterns. Up to now the strategy pattern has been adapted for UML State Machine and Activity Diagram models [16]. The strategy pattern can be used for realizing alternative variation points.

Extensions/extension points. Extensions/extension points use a combination of sub-process encapsulation, delegation and aggregation of functionality and null-sub-processes for realizing optional variation points.

Figure 22 gives an overview of the variation point types supported by the variability mechanisms for the automotive domain.

Figure 22: **Overview of relationship between variability mechanisms and variation point types**

Variability Mechanism	**Variation Point Type**		
	optional	alternative	range
Encapsulation of sub-processes	**X**	**X**	
Replacement of encapsulated sub-processes		**X**	
Omission of encapsulated sub-processes	**X**		
Parameterization	**X**	**X**	**X**
Templates	**X**	**X**	

Inheritance		**X**	
Strategy pattern		**X**	
Extensions/ Extension Points	**X**		

6.2 Modeling Variant-Rich Workflows in the E-Business Domain

This section describes the mapping of the concepts introduced in chapter 3 to elements of the Business Process Modeling Notation (BPMN). The elements of the BPMN are then extended to represent variation points in a variant-rich business process diagrams.

6.2.1 Mapping of Relevant Concepts to BPMN

Like the conceptual model, also the BPMN is centred on the process definition. A process in BPMN is represented as a directed, (possibly) cyclic graph consisting of different node and edge types. The highest level node is of the type *FlowObject*, whereas the edges represent sequence, message, or associational flow. Specializations of *FlowObject* are i.e. tasks, gateways, or events. See the PESOA technical report #1 (Process Modeling Techniques) or the official specification for further details on BPMN [21], [22].

Figure 23: **Mapping of the concepts from the conceptual model to BPMN**

	PESOA Conceptual Model	**BPMN**
Process	Process	Process, Pool (for abstract processes)
	Composite Activity	Sub-Process
	Activity	Task
Control Flow	Sequence	Sequence Flow
	Parallel	AND Gateway
	Choice	XOR/OR Gateway
	Iteration	Iteration Marker / Sequence Flow
Data Flow	Input	InputSets, Inputs Attributes
	Output	OutputSets, Output Attributes
	DataSource	(Abstract) Processes
	DataSink	(Abstract) Processes
Environment	State	Status Attribute of Process
	Variable	Data Object
	Scope	Process, Sub-Process, Task, Group
	Event	Event

	Exception	Error Event
NFP	Transaction[1]	Transaction
	Costs	Associations, Text Annotations
	Security, etc.	Represented by the process structure

Figure 23 shows how the elements of the conceptual model can be mapped to BPMN. The table has been parted into the five areas introduced in chapter 3.

The first area of the conceptual model deals with the concept of a process. A process in the e-business domain is represented by a workflow; a technical description of how to achieve a specific outcome for a customer. The steps inside a workflow could be coarse or fine granular, i.e. ranging from composite to atomic activities. In BPMN, one or more processes are placed inside a pool, which represents a participant of the domain. A pool is also used at a higher-level concept as it can hide its inside processes, those defining an abstract process. A BPMN process is refined into sub-processes as well as tasks as the atomic unit of work.

Control flows between different activities as well as other elements of the BPMN (together called FlowObjects) are represented by sequence flows and gateways. Sequence flow can connect FlowObjects sequentially. Parallelism and choice is represented by different BPMN gateways, which route the sequence flow. Allowing more than one incoming our outgoing sequence flow from a sub-process or task also represents parallelism. Multiple incoming sequence flows are interpreted as an AND-join, whereas multiple outgoing sequence flows act as an AND-split. Iterations as well as other kinds of cycles are supported by backward sequence flows (the BPMN makes no restrictions on cycles). Sub-process and tasks can also be marked with a loop-marker, which indicates the sequential, repeated execution. Furthermore, the BPMN is capable of modeling all major workflow patterns [24], [23] as for instance different kinds of multiple instances, milestone, or cancellation.

Data flow is only elementary supported in the BPMN. However, all related concepts from the PESOA conceptual model are supported. Each Activity, sub-process or task, has attributes called input sets and output sets. Each set has a number of inputs or outputs assigned. The semantics is related to UML2 activity diagram parameter sets. An input set defines the data requirements for an activity, whereby each activity can have more than one input set. However, only one input set filled with all inputs is required to execute the activity (OR-behaviour). The same holds for the output sets – only one output set will be filled with outputs at the completion of the activity regarding to the implementation and so called input-output rules. An input-output rule defines the relationship between one input and one output set. If

[1] Although transactions are not contained in the conceptual model, they are required for the e-business domain and are therefore contained here.

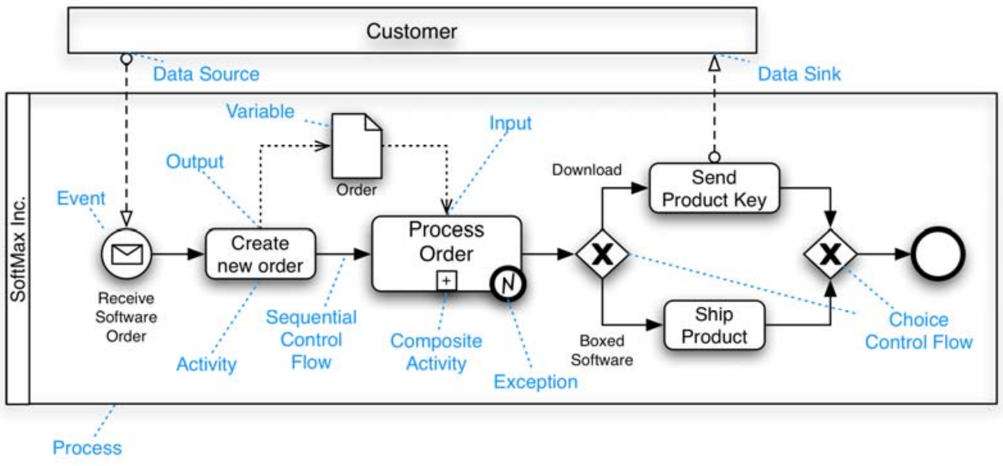
an input set, specified in an input-output rule, has been used to instantiate the activity, then the corresponding output set must be produced at the completion of the activity. The BPMN contains no counterparts to UML2 activity diagram central buffer or data store nodes. However, other processes, either concrete or abstract as a black box pool, could represent data sources and sinks.

The process state concept from the conceptual model is represented by an attribute status of each BPMN activity. The status is determined and maintained by a process engine that executes the corresponding process. The status of an activity might be used as a part of assignment expressions; e.g. for the routing of control flow. BPMN data objects represent data structures, which contain variables. Each data object might have its own state regarding to the allocated variables. However, this is outside the BPMN specification. Processes, sub-processes, tasks, and groups represent a scope in BPMN. Sub-processes and groups can have events attached, so they match the conceptual model most. Thereby the BPMN supports different kinds of events, ranging from events that start a sequence flow, over intermediate events that interrupt sequence flows and activities, as well as events that end a control flow. An exception is a special kind of event, called error event.

BPMN directly supports the concept of transactions by marking a sub-process with a double line as well as some special events. Associations and text annotations can represent costs and other non-functional properties. Concepts like security can be only represented by the process structure; therefore no special elements exist.

The mapping of the described concepts from the conceptual model to the BPMN are exemplary shown in Figure 1.

Figure 24: Example for the mapping of the concepts to BPMN



The sample process describes the workflow of a small software company called SoftMax Inc. SoftMax receives orders for her products using HTTP web forms. After a new request has been received, an internal order is created. The order is processed, which e.g. includes the billing activities belong others. Finally it is checked if the customer has ordered a boxed version (with a printed manual) or a download. If boxed software has been ordered, the product is shipped. If a download has been ordered, the key is sent to the customer. The related concepts are annotated in the example.

6.2.2 Modeling Variability in the E-Business Domain

For the visualization of optional and alternative elements of a business process diagram, extensions to the representation of variable elements are introduced. Further details can be found in the PESOA technical report #17 (Variability Mechanisms) [16].

A variant-rich business process diagram needs to contain the same information to describe variability as stated in section 5.1.2 (Modeling Variability in the Automotive Domain). Those are the 1) variation points, 2) possible resolutions, as well as 3) the variability mechanisms used to realize the variability. In chapter 4, three types of variation points have been identified, which will also be supported in a business process diagram.

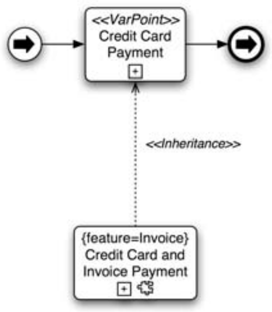
To fulfil requirement 1, the identification of variation points, the concept of a stereotype from the UML2 specification is adapted to BPMN. Each activity, association, and artifact can have a stereotype attached. The name of the stereotype is written in italic letters, placed between two angle brackets at each side. It is recommended to place the stereotype above the name of the object if used within an activity or artifact, or beside the association. For sub-processes, the stereotype can be placed before the name of the sub-process. For the purposes of a variant-rich business process diagram, the introduction of a stereotype called <<*VarPoint*>> is sufficient. This stereotype can also be expressed graphically as a puzzle-piece like marker at the button of an activity. However, if the graphical representation is used, the textual notation has to be omitted.

For an easier understanding of a variant-rich business process diagram and the variability mechanisms used, the stereotype variant can be refined with tagged values, as defined according to the UML2 specification. A tagged value can be written below a stereotype in curly brackets by using the keyword type: {tag=value}. Each stereotype can have two predefined tagged values, feature and type. The feature tag represents the feature the variant belongs to e.g., alternative, option, or range, whereas the type tag further indicates which kind of variability mechanisms is used e.g., inheritance, extension, parameterization. The values of the type tag can be optional, abstract, null, and default. Their semantics will be explained later on.

To safe screen as well as paper space, the four types of the stereotype <<*VarPoint*>> , which are represented as tagged values, can also be represented as own stereotypes, thereby specializing <<*VarPoint*>>. The corresponding stereotypes are <<*Optional*>>, <<*Abstract*>>, <<*Null*>>, and <<*Default*>>. Furthermore, the tagged values of a stereotype can be omitted in the graphical representation.

Possible resolutions to a variation point are either contained in the graphical representation of the variation point itself, thereby representing the default behaviour, as well as by using associations from the variation point to activities or artefacts which are marked as variant. An example is shown in Figure 25.

Figure 25: **An example for representing variability in a business process diagram using inheritance**



The following variability mechanisms from the UML activity diagram section (5.1.2) have been adapted to the BPMN. They are described in the remainder of this subsection.

Encapsulation of sub-processes. A BPMN sub-process can hide alternative variant sub-processes behind an invariant interfaces. Thereby, an interface is defined as the set of input and output events of an activity. The interface activity is marked with the stereotype *<<Abstract>>.* Possible realizations of the interface are connected using associations marked with *<<Implementation>>.* The default implementation should be marked with the <<Default>> stereotype.

Parameterization. Each BPMN attribute can be parameterized to support optional, alternative, or range variation points. For a graphical representation, the attribute is written beside the element and surrounded with a grouping box. If the connection between the attribute and the element can be misinterpreted, an association should be used. Association are also used to link variant data objects that contain the possible parameters to the grouping box that surrounds the attribute. The association is marked with the stereotype *<<Parameterization>>.*

Inheritance. Inheritance modifies an existing (default) sub-process by adding or removing elements regarding to specific rules. This allows for realizing alternative variation points. An association represents inheritance from the child activity to the parent activity when it is marked with the stereotype *<<Inheritance>>.*

Extension Points. Extension points use a combination of encapsulation and null sub-processes to realize optional variation points. An extension point activity is marked with the stereotype *<<Null>.* Associations marked with *<<Extension>>* connect optional implementations. If there is only one optional variant, it can be shown instead of the null activity, marked with an *<<Optional>>* stereotype.

Figure 26 gives an overview of the variation point types supported by the variability mechanisms for the e-business domain.

Figure 26: **Overview of relationship between BPMN variability mechanisms and variation point types**

Variability Mechanism	**Variation Point Type**		
	optional	alternative	range
Encapsulation of sub-processes		**X**	
Parameterization	**X**	**X**	**X**

Inheritance		**X**	
Extension Points	**X**		

7 Summary

This report presents the methodological foundation for process family engineering. This has been accomplished by transferring the main principles from the product line engineering approach to software engineering domains that use processes as a driving software engineering artifact, since processes determine the different characteristics of the developed systems in these domains. By means of establishing analogies between the terms product-process and feature, the product line engineering approach has been converted into a process family engineering approach.

In order to understand better what a process is in the e-business and automotive domains, a conceptual model has been defined and documented. Such conceptual model defines a common process vocabulary for both domains, which can be used as a basis for identifying variability, a central aspect in product-line engineering.

The conceptual model presents the process as the core concept, which in turn is associated to: a) concepts related to the process environment e.g., state, variable, scope, event, and exception b) concepts related to the process non-functional properties e.g., quality of service, safety, security, and real time c) concepts related to the data flow e.g., input, output, data sink, and data source d) concepts related to the control flow e.g., sequence, parallel, choice, and iteration.

Since variability modeling is broadly investigated and developed in the product line engineering approach, this approach is taken as basis for developing the process family engineering approach (the PESOA process) and introducing variability in the PESOA conceptual model. Therefore, it can be said that process family engineering focuses on finding the commonalities and variabilities of a set of processes in a given domain, and integrating them in a process family infrastructure. A process family infrastructure consists of variant-rich processes, feature and configuration models. A variant-rich process contains variation points that represent its variability. Relationships among variation points in a process family infrastructure are captured in a configuration model. A dependency is a variation point that constrains the resolution of other variation points.

The conceptual model and the variability concepts can be used in the PESOA process in order to either create/maintain the process family infrastructure (i.e., domain engineering), or create/generate a specific product (i.e., application engineering). The PESOA process is presented by means of a product flow, where activities and products are described at a high granularity level. This process is intended to be instantiated in order to cope with the goal of the PESOA project, which is to design and implement a platform for process families. The PESOA process serves as a framework for connecting specific research topics that are investigated and described in this and other reports, such as scoping definition, and modeling of variant-rich processes. Other research topics are left open, such as the commonality analysis of processes.

The applicability of the PESOA concepts has been demonstrated by mapping them to the process modeling languages used in PESOA for modeling processes in the automotive and e-business domain. These are: Activity and state diagrams for modeling automotive processes; and BPMN for modeling e-business processes. Examples on how to model concepts like variation point, variability and variability mechanisms have been introduced, while modeling variant-rich processes by means of variability mechanisms for process models is described in detail in the PESOA technical report #17 (Variability Mechanisms) [16].

8 References

[1] K. Schmid: Planning Software Reuse - A Disciplined Scoping Approach for Software Product Lines, Ph.D. Thesis, Fraunhofer IRB Verlag, 2003.

[2] J. Bayer, M. Eisenbarth, T. Lehner, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland: Domain Engineering Techniques and Process Modeling. PESOA Report TR09/2004.

[3] J. Bayer, W. Buhl, C. Giese, T. Lehner, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland: Process-based Product Line Engineering. Submitted to SPLC 05 (Software Product Line Conference 2005).

[4] B. Bruegge, A.H. Dutoit: Object-Oriented Software Engineering. Using UML, Patterns, and Java. 2nd ed. Upper Saddle River: Pearson Education (2004)

[5] M. Owen, J. Raj: BPMN and Business Process Management Introduction to the New Business Process Modeling Standard. Available http://whitepaper.techweb.com/cmptechweb/search/viewabstract/71885/index.jsp

[6] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A.P. Barros: Workflow patterns. Technical Report BETA Working. Paper Series, WP 47, Eindhoven University of Technology, 2000.

[7] D. Muthig: A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. Stuttgart: Fraunhofer IRB Verlag, 2002 (PhD Theses in Experimental Software Engineering Vol. 11). Kaiserslautern, Univ., Diss., 2002

[8] A. Ocampo, J. Münch, F. Bella: Software Process Commonality Analysis, Software Process Improvement and Practice, July 2005.

[9] M. Becker: Anpassungsunterstützung in Software-Produktfamilien, PhD Thesis, Technical University of Kaiserslautern, 2004.

[10] S. Thiel, A. Hein: Systematic Integration of Variability into Product Line Architecture Design, Proceedings of the Second Software Product Line Conference, LNCS 2379, Springer, 2002.

[11] J. van Gurp, J. Bosch, M. Svahnberg: On the Notion of Variability in Software Product Lines, Proceedings of WICSA 2001, 2001.

[12] J. Bayer, T. Forster, T. Lehner, A. Ocampo, E. Richter, J. Weiland: Feature- und Entscheidungsmodell-basierte Varianteninstanziierung im PESOA-Prozess. PESOA Report 21/2005

[13] W. Buhl, C. Giese, H. Overdick: Realisierungsstrategien für Prozessfamilien. PESOA Report TR 15/2005

[14] E. Richter, A. Schnieders, J. Weiland: Prozessanalyse und –modellierung in der Domäne Automotive. PESOA Report TR 07/2004

[15] A. Werner.: Scoping von Geschäftsprozessen und Software-Produkten eines Zielmarktes. PESOA Report TR 16/2005

[16] A. Schnieders, F. Puhlmann: Process Family Engineering: Variability Mechanisms. PESOA Report TR 17/2005.

[17] T. Pender: UML Bible. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

[18] OMG: Unified Modeling Language: superstructure. Version 2.0. 2003. Available at: http://www.omg.org/cgi-bin/doc?ptc/2003-08-02

[19] M. Born, E. Holz, O. Kath: Softwareentwicklung mit UML 2. München: Addison-Wesley 2004

[20] J. Rumbaugh, I. Jacobson, G. Booch: The Unified Modeling Language Reference Manual. Addison-Wesley 2005

[21] A. Schnieders, F. Puhlmann, M.Weske: Process Modeling Techniques. PESOA Report TR 01/2004.

[22] BPMI.org: Business Process Modeling Notation. Technical Report, 2004

[23] S. White: Workflow Patterns with BPMN and UML. IBM Technical Report, 2004. Available at: http://www.bpmn.org/Documents/Notations%20and%20Workflow%20Patterns.pdf

[24] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A. Barros: Workflow Patterns. Distributed and Parallel Databases (5)51, 2003.

[25] K. Kang, S. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson; “Feature-Oriented Domain Analysis (FODA) Feasibility Study”; Technical Report CMU/SEI-90-TR-21; 1990

[26] K. Czarnecki, U. Eisenecker, Generative Programming – Methods, Tools, and Applications, Addison-Wesley, Boston, MA, 2000