PESOA

Process Family Engineering in Service-Oriented Applications

BMBF-Project

Domain Engineering Techniques and Process Modeling

Domain Engineering for Families of Service-oriented Applications

Authors:
Joachim Bayer
Michael Eisenbarth
Theresa Lehner
Frank Puhlmann
Ernst Richter
Arnd Schnieders
Jens Weiland

PESOA-Report No. 09/2004
30 October 2004

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 Ltd.
- Fraunhofer IESE
- Hasso-Plattner-Institute
- Intershop Communications Inc.
- University of Leipzig

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

www.pesoa.org

Abstract

The goal of the PESOA project is to design and implement a platform for families of related service-oriented applications. The envisioned platform is used to manage process variants for families of service-oriented applications and to enable the process-based instantiation of such service-oriented application families.

An important aspect of this goal is the modelling of variant-rich or generic processes that capture processes for a number of related applications. In this report, we present a number of analyses that together investigate the current state-of-the-art and state-of-the practice for modelling variant-rich processes for the e-Business domain and automotive application domain. The results of these analyses will be used in the further course of the PESOA project to develop techniques for modelling variant-rich processes and for using them to specify families of service-oriented applications.

We analyze domain engineering techniques as a way to bring variability in processes. Additionally, we look at requirements posed by the two considered application domains, the e-Business domain and the automotive domain, as well as languages for process modelling.

Keywords: PESOA, Service-oriented Application, Software Product Lines, Domain Engineering, Process Modeling, Automotive, eBusiness.

Table of Contents

1 Introduction

This report analyzes several aspects related to the modeling of variant-rich processes and their use to specify families of service-oriented applications. The results of the different analyses will be brought together in the further course of the PESOA project to enable the modelling of variant-rich or generic processes that capture processes for a number of related applications.

The analyzed aspects are:

- Domain Engineering Techniques (chapter 2): in this chapter the requirements for modeling generic processes are elicited and the process support of different existing product line engineering approaches is discussed.

- E-Business Domain (chapter 3): this chapter presents the special characteristics of the e-business domain with respect to modeling e-business processes.

- Automotive Domain (chapter 4): this chapter presents the special characteristics of the e-business domain with respect to modeling e-business processes.

- Process Modeling Languages (chapter 5): in this chapter the current state with respect to process modeling languages is presented. A special focus is set on modeling processes in the two domains presented in chapters 3 and 4.

Chapter 6 finally concludes this report and provides a brief outlook into how the results presented here will be used in the further course of the PESOA project.

2 Product Line Engineering for Process Families

In the following chapter, we will motivate the introduction of a Product Line Engineering approach for Process Families. First, we will describe how a product line engineering for product families looks like and introduce the two main phases of a product line engineering approach, domain engineering and application engineering. After that, we briefly introduce an incremental product line engineering approach that could be used for process family engineering in the context of the PESOA project. We will describe to what degree the incremental product line engineering approach must be adapted for process families and how well already existing activities are capable for the process engineering task. In the final section of this chapter, several product line and domain engineering methods are briefly described and their qualification for process family engineering is discussed.

2.1 Motivation

Product line engineering stands on a middle ground between specific assets, which are used for single system development and only once used, and general assets, like general use libraries or components that are applicable anywhere. The idea is to define a Product line, which captures the intended scope of reuse. The products included in a Product Line are determined to have sufficiently common characteristics to make it more efficient to study the commonalities and variabilities for all products of the Product line and to build reusable assets, than to study and build all the products separately. Product line engineering approaches define how to leverage the commonalities and create reusable assets that increase the efficiency of developing the products [Bayer99a].

One benefit of the Product Line approach is the idea that all members of the Product Line are based on a single set of assets. Thereby, the maintenance effort is reduced to this single asset base. Hence, existing products must always follow the evolution of the asset base. This is done by instantiating the changed domain model while reusing the existing resolution of the domain decision model.

2.1.1 Product Line Engineering

Product line engineering can be described as a technology providing methods to plan, control, and improve a reuse infrastructure for developing a family of similar products instead of developing single products separately. This reuse infrastructure manages commonality and controls the variability

of the different products. Examples for Product Line approaches are PuLSE [Bayer99b], Fast [Weiss99] and the SEI Product Line Practice Initiative [Clements01].

The core idea of applying a product line engineering approach to process families is to analyze a set of processes and exploit their commonalities systematically rather than modelling process by process individually. This implies that information in a process family context is mainly concerned with multiple processes and their variations. A solution to this is structuring such information by comparing a set of processes but keeping the information on each individual process separately visible. Distinct from single system software development there are two life cycles, domain engineering and application engineering (cf. Figure 1). Domain engineering analyzes information on individual processes, integrates it by consideration of commonalities and variabilities, and stores the integrated information as part of the product line infrastructure. Application engineering uses the integrated information and specializes it according to the needs of a particular service request.

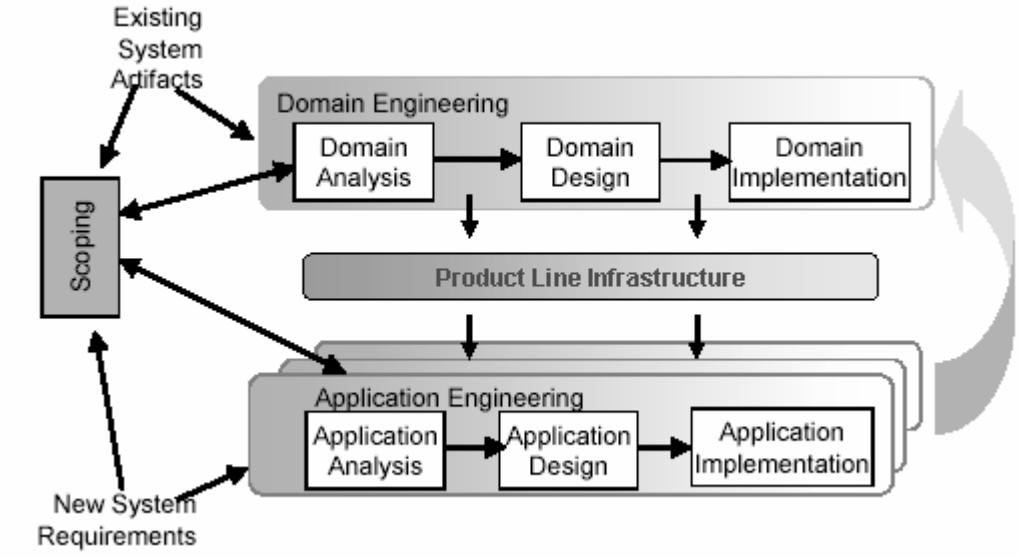


Figure 1- Product Line Engineering two lifecycle approach

Three activities performed during domain engineering are the domain analysis, which means determining what the family is about, the domain design, which complies with deciding which platform components are needed, and the domain implementation that involves building and buying components and supporting infrastructure. According to this, application engineering also involves three activities. Application analysis determines what the products should be, the application design selects the appropriate components to make these products and the application coding combines

the components by using the infrastructure and possibly additional product-specific code.

In a product line engineering development approach [Chastek02], [Donohoe00], [Nord04], the characteristics of multiple related systems are handled and captured in an integrated way. That is, information is captured by product line artefacts that are focused on comparing common and varying characteristics of particular similar systems. A product line artefact can be defined as an artefact that captures product line concepts such as commonalities or variabilities in an integrated and explicit form. A product line artefact that captures no variabilities is identical to an artefact used in a single-system context. Example product line artefacts are feature models, textual requirements documents, business process models, UML class diagrams, or C-source-code files.

In the context of process family engineering, it would be necessary to capture additional information about variant process parts in a similar way, which means that process artefacts that are focused on a specific application domain have to be added to the product line engineering approach. In that case, the process artefacts contain commonalities as well as variabilities of processes.

2.1.2 Domain Engineering

Domain engineering analyzes an application domain, its abstract concepts, entities, and relationships in order to build a reference model for systems in the domain including domain-specific reusable artefacts. The term artefact subsumes all kinds of work products manipulated by development activities. Concrete applications are then constructed mainly by reusing the domain specific artefacts, which represent the domain concepts or features required for the concrete application. In the case of process family engineering, the development of reusable process assets is the prominent goal. The domain engineering task will be to build up a reference process model for the processes in the analyzed application domain and the explicit specification of the domain-specific reusable process artefacts.

Domain analysis or domain modelling is requirements engineering for product lines. The goal of any domain-analysis approach is to identify and document requirements on a set of products in the same application domain in order to make development and maintenance activities more efficient. The results are captured in domain model. The domain model must capture both the common characteristics of the products and their variations. This domain model is then the basis for creating other reusable assets like a domain specific language or a component-based architecture. At an abstract level, the domain analysis process starts with an application domain, analyzes all the diverse requirements on products in that application domain, identifies common and variable requirements, and finally documents this information

in form of an integrated domain model. Example variant requirements that are captured by a domain model are:

• Optional requirements: requirements that do hold for a particular system or do not hold

• Alternative requirements: a set of requirements of which only one or a subset holds for a particular system

• Range requirements: requirements that specify the potential range for a numerical value, which is supported by the domain model instead of the specific value as required by a single system

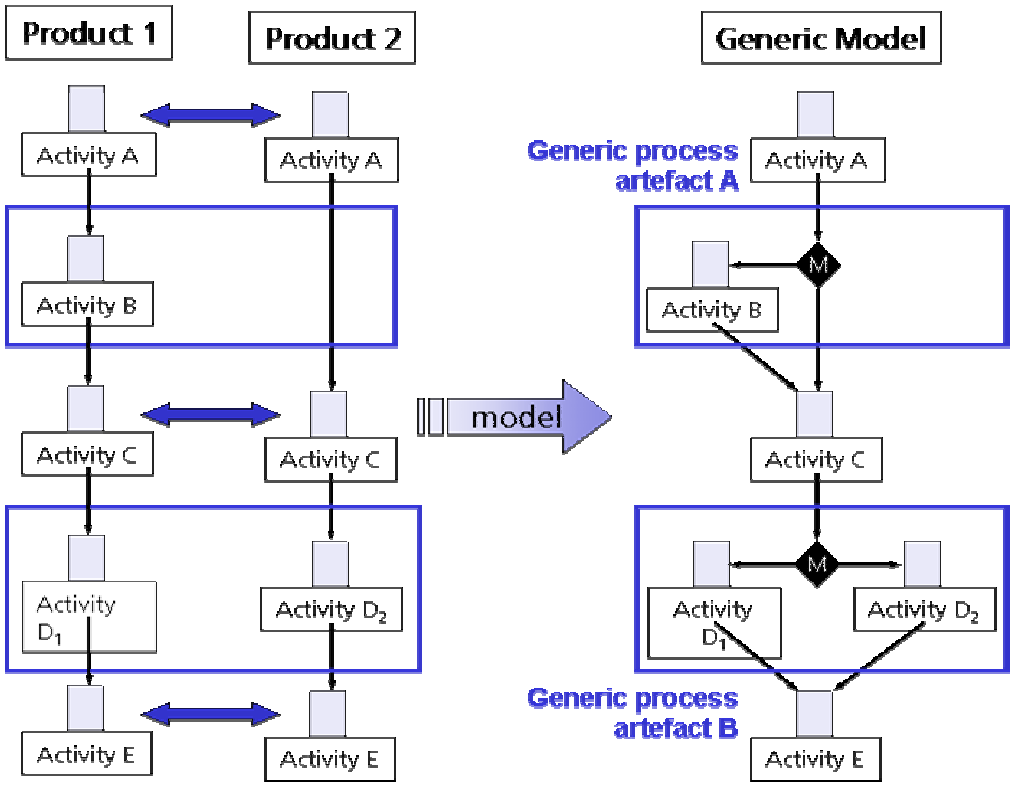


Figure 2 - Generic process model

For a domain analysis method to be applicable it must be appropriate and it must provide enough guidance so that it can be carried out. As in other areas of software development, the context for each domain analysis application varies, and methods that are appropriate in one context will not be in others. This fact is especially important for domain analysis because of the compound effects of inappropriate models over multiple products and over the whole lifecycle. Therefore, a generally applicable domain analysis method should be customisable to the context of the application.

The analysis of a domain for process family engineering comes up with a domain model that captures all aspects of the processes in that application domain. These aspects, common or variant for the diverse processes, are then integrated into the generic process model.

The above picture illustrates the domain analysis task. Two products or processes of an application domain are analyzed during the domain analysis step of the incremental product line engineering approach. The result of that task, the generic process model, captures both the common and the variant parts of the application domain processes by integrating the information in generic artefacts. The application engineering task would then be to use these generic process artefacts to specialize the generic process for a specific application.

At this point in time, an emerging issue is how to model and notate such a generic process model and especially the variable parts, which means the generic artefacts, of the generic model. The notation used in the above picture is just an example notation based on the ARIS process modelling language. A mechanism for dealing with generic components and their integration into product line artefacts is variability modelling, which is described later in the incremental Product Line Engineering section.

2.2 Incremental Product Line Engineering

An incremental Product line engineering approach for product families consists of several activities that have to be added to the development processes concerned with requirements, architecture, design, implementation, or testing for single-systems. The advantage of an incremental approach is that existing products and processes can be further used with minimal adaptations. The product line aspects are then integrated in the development process by adding additional activities to it.

An incremental product line modelling approach described by [Muthig02] introduces three activities that are added to the software development process in order to comply with product line aspects. These activities are:

- Variability Modelling
- Decision Modelling
- Application Engineering

In the beginning of an incremental product line development process, a new product is developed as in a standard single-product development process. If a new product is modelled, or a new asset is created that has not been required by previous products or a new functionality is added to an already

existing generic artefact, the new asset as a whole is considered as an optional asset for the product line asset base and no variability must be modelled.

Variability Modelling requires that the product line context must be taken into account by identifying all elements of an artefact that are variable across different products. Generic artefact elements are then used to generically model these variabilities. The variant parts of a generic artefact yield so called variation points that must be resolved later to specialize the artefact to the needs of a particular product context. Variation points of a generic artefact capture what may vary in the artefact and how it may vary.

The relationships and dependencies among variation points across all generic artefacts of a product line infrastructure are typically not explicitly captured as part of the generic artefacts themselves. These relationships are defined separately in form of a decision model. A decision model is a product line artefact that captures relationships among variation points in a set of generic artefacts. This is done during decision modelling by integrating the variation points into the decision model by relating them with other variation points, as well as with decisions from related generic artefacts.

The picture below illustrates the relationship between variation points of generic artefacts of different development levels, i.e. requirements, component, implementation.

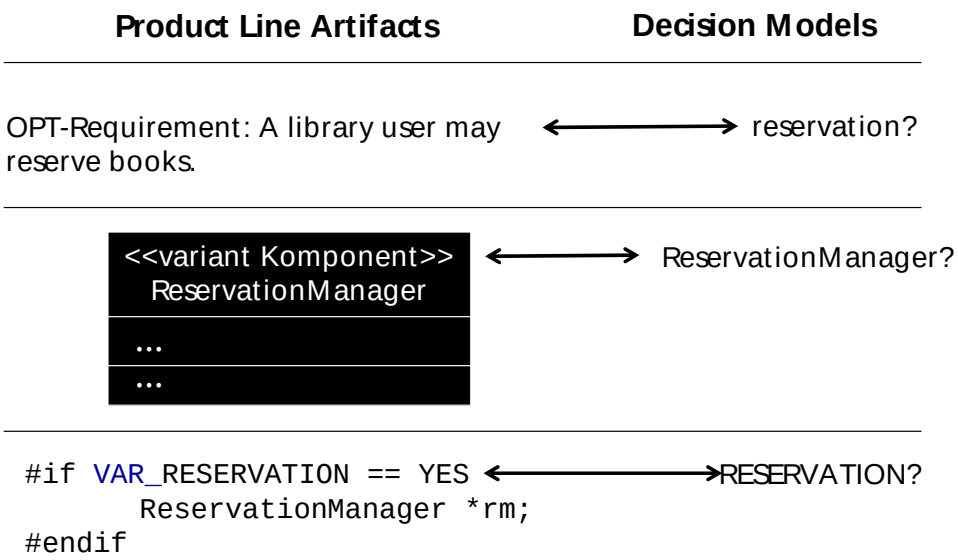


Figure 3 - Variation Points and Decision Models

The decision model is used to capture the relationship of variation points describing the same variability but on different abstraction levels. In the

example, the variability described in the above picture is the possibility to reserve books. This variability is expressed in different generic artefacts all over the product line development process. The first row describes the variability in a requirements document, the second row corresponds to the design document and the last row models the variation point on the implementation level.

The instantiation of generic infrastructure artefacts by resolving all variation points for a particular context is performed during application engineering.

In order to apply such an incremental product line engineering approach for process family oriented development, several activities must be modified by adding mechanisms for generic process family engineering. In the following section, the activities of Variability Modelling, Decision Modelling and Application Engineering are introduced in more detail concerning the modifications required for process family engineering.

2.2.1 Variability Modelling

The variability modelling activity is concerned with the generically integration of variability into a product line artefact. This can be done by using generic product line artefacts. The solution is to replace all elements related to a specific variability with the corresponding specialized generic artefact. The modelling element that covers the variability in a generic product line artefact is a variation point. Variation points of a generic artefact capture what may vary in the artefact and how it may vary. The relationships and dependencies among variation points across all generic artefacts of a product line infrastructure are typically not explicitly captured as part of the generic artefacts themselves. These relationships are defined separately in form of a decision model, which is created during decision modelling.

In order to introduce generic artefacts for process family engineering, variation points must be included in the process models and process specifications. As it was already mentioned in the previous sections, variation points occur on different specification levels, like requirements, design or code. Thus, the integration of variation points to product line artefacts for process family engineering depends on the respective process specification level that means variation points must be expressed by graphical elements in mere graphical process descriptions like ARIS but also in process specification languages like BPEL. Therefore, the exact structure of a variation point inside a generic artefact for process family engineering depends on the used representation form of the generic artefact.

2.2.2 Decision Modelling

The decision modelling activity produces and maintains the decision model. That is, it captures the relationships between newly modelled variation points, as well as among new variation points and previously existing variation points. Typically, decision modelling follows a bottom-up approach. First, the constraints among newly modelled variation points are identified and captured by constraining decisions. Then, these new decisions are integrated into the decision model by identifying and modelling how they are constrained by already existing decisions. Typically, the existing decisions are related to artefacts that stem from the previous life-cycle stage and thus have been input to the actual instance of product line modelling. For example, decisions related to implementation artefacts are related to design decisions, design decisions are related to requirements-related decisions, and so forth. In general, all decisions must eventually be related to other decisions because only the variability is modelled that has been requested. That is, the goal is to get the simplest generic artefact that exactly covers the needed variability.

An example decision model is illustrated in the table below.

Name	Relevance	Description	Range	Selection	Constraints	Binding Times
Memory	System_Mem = True	Does the sytem have memory?	TRUE, FALSE	1		Compile Time
Memory_Size		The amount of memory the system has (KB)	0..100.000	1	Memory=TRUE => Memory_Size > 0	Installation, System Initialisation
Time_Measurement		How is time measurement done?	Hardware, Software	1		Compile Time

Table 1 - Decision model

A decision variable is a unique identifier for a variation and corresponds to a specific row in the decision model. A decision variable can be used for several variation points (e.g. a decision variable SERVICE_INPUT is used at each variation point where the service input has to be chosen). Each of the decision variables that is defined in the decision model is in turn described by the following information:

- Name: The name of the defined decision variable; the name must be unique in the decision model

- Relevancy: The relevancy of a decision variable for an instantiation may depend on other decision variables, e.g. the decision variable describing the memory size in the above example is only valid if the decision variable describing the

existence of memory is true. This can be made explicit by the relevancy information.

- Description: A textual description of the decision captured by the decision variable

- Range: The range of values that the decision variable can take on. This can be basically any of the typical data types used in programming languages.

- Cardinality: The cardinality defines how many of the values of a decision variable can be assumed by it.

- Constraints: Constraints are used to describe interrelations among different decision variables. This is used to describe value restrictions imposed by the value of one variable onto another variable.

- Binding times: A list of possible binding times, describing when the decision can be bound. This can be source time, compile time, installation time, etc.

2.2.3 Application Engineering

Application engineering is the process that interacts with customers of particular product line members. When a new product is initiated due to a customer or market request, application engineering is started. It reuses as much as possible from the product line infrastructure to build a product. Based on the decision model that contains all points of variation in the infrastructure and that relates them to functions of the resulting applications, a specific product line member is derived from the infrastructure. The result is an application that can be deployed. Therefore, it instantiates the generic infrastructure artefacts by resolving all variation points for the particular context. Then, required product characteristics that are not covered by the infrastructure are added.

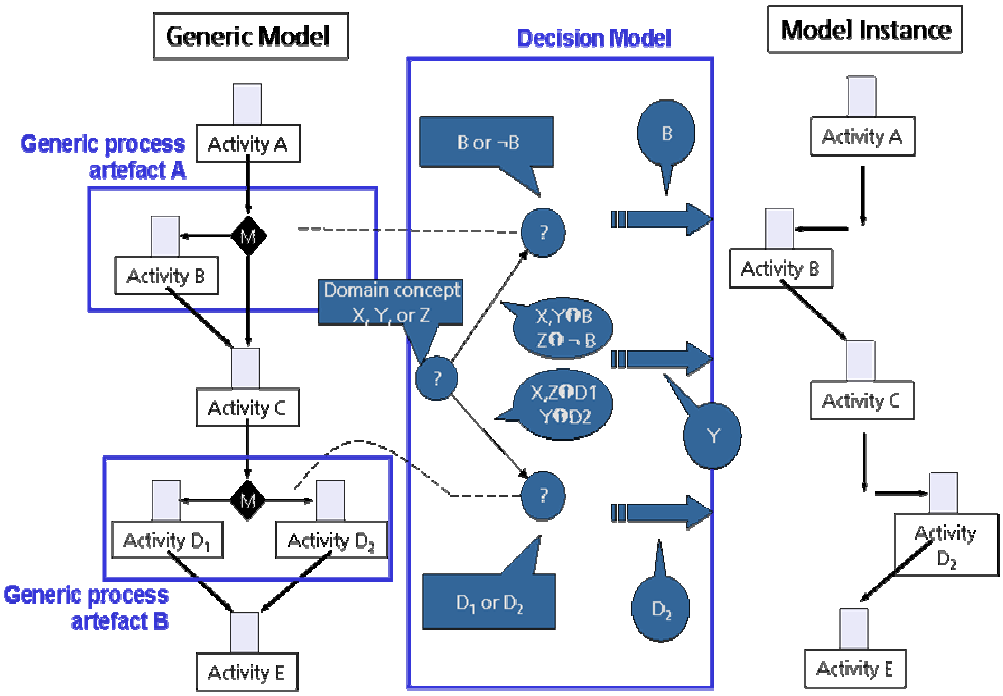


Figure 4 - Process family application engineering

Application engineering for process families would instantiate the generic process models by resolving the variation points in the generic artefacts. This includes the instantiation of the generic process models that are part of the process family infrastructure. The above picture illustrates the instantiation of the generic process model, containing the variation points. By solving the decisions of the decision model and thus keeping domain specific characteristics in mind, a model instance is derived that fulfils the intended process. Especially in the context of service orientation, this phase would instantiate the generic models by resolving the decision model containing also relations of the different services. The result would then be a service-oriented application that can be applied.

2.3 Feature-Oriented Domain Analysis (FODA)

The Feature-Oriented Domain Analysis (FODA) method was originally developed by the Software Engineering Institute (SEI). According to [Kang90] the method was developed to “support the systematic exploration of related software systems in order to discover and exploit commonality.” The underlying idea is the analysis of features, which the user expects to be in the application.

In the context of the PESOA project and the underlying task to support service-oriented applications and families of processes, the FODA method

could hence be used to analyze the services, which the user expects to be in the application and thus “support the systematic exploration of process families and the discovery and exploitation of common services”.

The FODA process consists of three major activities: Context Analysis and Domain Scoping, Domain Modeling, and Architectural Modeling. The process is visualized in Figure 5. In the following, we briefly introduce the three activities and explain the impact of service-orientation to the original activities. Although FODA uses the term *feature* for the underlying modeling concept and is thus used in the following description of the FODA method, it is possible to substitute the term *feature* with *service* to comply with the service-orientation theme.

2.3.1 Context Analysis and Domain Scoping

The definition of the scope of a product line is an essential task of product line modeling. This part of the FODA process is responsible for the scoping task and the results are documented in the context model. A context model describes what is “in” and what is “out” of the product family’s boundary. FODA uses *structure diagrams* and *data-flow diagrams* (context diagrams) to describe the context model. Structure diagrams show the relationship of the target domain with other surrounding domains, e.g. higher-level domains, sub-domains, or peer-level domains. The data-flow diagrams describe the data-flows and communication links between the domains, identified in the structure diagram.

The context analysis specifies if a component is reusable in different contexts. It is important to know how a component should be structured so that it can be adapted to different contexts. For example, the user interface part of a software application should be modeled in a way that it can be used for different terminals without extensive modifications. Thus, a context description is especially important if the system or the component is intended to be used in various contexts of the same domain. If a domain contains a large degree of variation in its usage contexts, it should be rescoped to a narrower domain [Kang90].

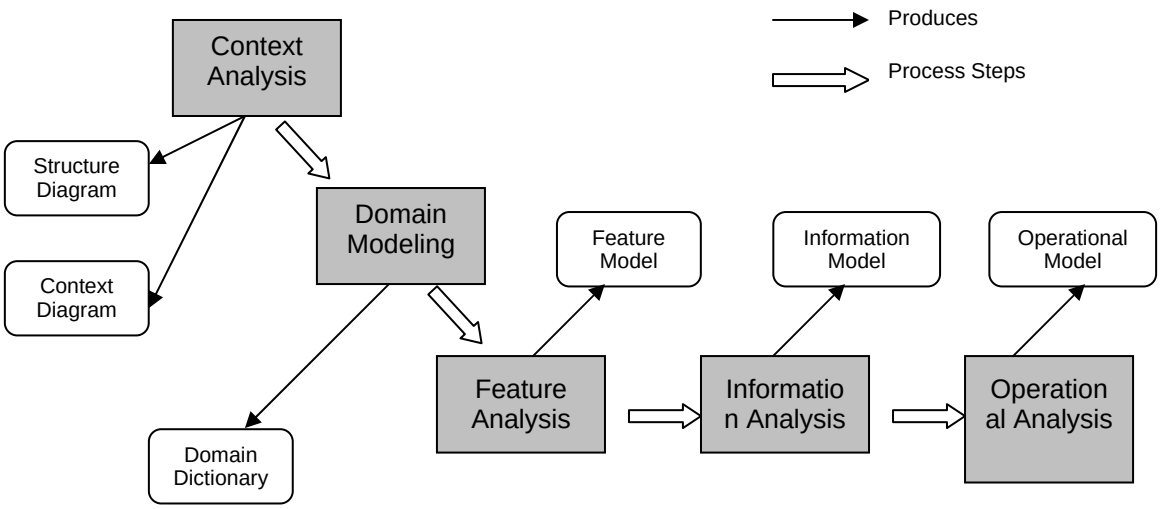


Figure 5 : The FODA-Process

2.3.2 Domain Modeling

In this phase of the FODA method, the main task *feature analysis* is performed. The purpose is to identify the functionality the user expects to be in the application and to build a model of the product family that captures this information. With the resulting feature model, the commonalities and variability of the product family members are described. Adapted to the service-oriented applications, the feature analysis would be used to identify the services the user requests and to build a model of the process family. FODA uses several concepts to describe certain aspects and interdependencies of features [Kang90].

Types of features: FODA supports three types of features: mandatory, optional, and alternative features. These concepts describe if a functionality or in our terms service must be available, could be available or is variant.

Groups of features: FODA allows the grouping of certain features with the relationship “consist-of.” This proceeding is performed to deal with large and complex feature models.

Feature Dependencies: FODA supports the dependencies “mutual-exclusive” and “requires.”

Feature binding time: FODA supports the following binding times: compile-time, load-time (e.g. for systems that are configured at the beginning of the execution), or run-time.

Feature categories: FODA distinguishes features after the following categories: operational features (active functions carried out by the application), context features (environmental characteristics), and representation features (how information is presented).

During domain modeling, two additional work products are produced. First, an *entity-relationship model* (information model) is generated to capture domain knowledge in the terms of domain entities and their relationships. And second, an *operational analysis* is performed to structure the common operational features of the domain.

2.3.3 Architectural Modeling

In this phase, a technical solution to the domain model is developed. The architectural model represents a high-level view on the architecture and its different layers. The layering is performed so that reuse can occur at the appropriate layer for a given application.

2.3.4 Representation Form

As product line modeling methods are often described in the form of process models, they also have to make use of notations to describe the contents of their models. In the case of FODA, no specific notation is prescribed. This leads to the problem of identifying a proper notation for the particular context. FODA often proposes the use of hierarchical based diagrams to represent the feature model. Feature diagrams are modeled as trees with the root representing a basic concept or product and leafs representing single features. Additional graphical elements are added, to describe feature types and constraints. E.g. a circle denotes an optional feature, while a double line represents an alternative. A simplified feature diagram is presented in figure 6.

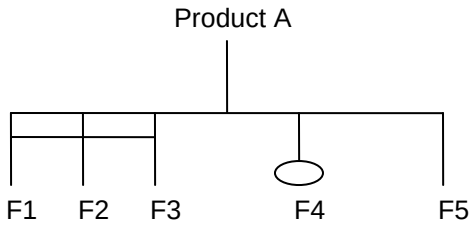


Figure 6 : FODA Feature Diagram

2.3.5 Summary and generic process engineering support with FODA

The major terseness of the Feature-Oriented Domain Analysis (FODA) method for process family engineering is that it only addresses the domain analysis and modeling task of the domain engineering phase. As already discussed in section 2.2 an incremental product line engineering approach has to address not only domain engineering but also application engineering and variability modeling issues. Both aspects are not covered by FODA. Support for the variability modeling is only provided as it identifies the variabilities in the domain. A further support for generic process artifacts is not provided. FODA does not support the instantiation of generic models to specific instances during application engineering except that it provides input for the decision model, which is used to capture constraints of the variation points. The FODA method as introduced in the former section can be used only for domain analysis and modeling issues, required for the decision modeling aspects of product line development. A process family oriented adaptation of the FODA method could therefore only be useful for domain modeling when integrated into a holistic process family engineering approach.

2.4 Family-oriented Abstraction, Specification, and Translation Process

The Family-oriented Abstraction, Specification, and Translation Process (FAST) was developed by David M. Weiss at Lucent [Weiss99]. FAST is a domain modeling process for product lines, which aims at providing a fast development of software products and an automated generation of new product line members. The necessary product specifications for the automated generation are described in domain-specific languages. The generated product line members share common requirements, designs, and code. As FAST is a complex domain modeling process, only the part of the commonality analysis will be presented in this report.

In the following description of the FAST process, the term product family or product line is used to describe the initial process. The usage of the term process family instead of product family seems to qualify FAST to be applied in the service-orientation and PESOA context.

2.4.1 Commonality Analysis

The commonality analysis part of the FAST process aims at identifying and modeling all common and variable parts of a product family. The representation form of this specification document is a simple textual notation. Independent of the specific domain of the product line this section is always described in the same form. The specification of commonality and variability contains two parts, one for each of them. The commonality part contains all requirements which are common to all members of the product line. The variability part covers all requirements varying for member to member. The only difference in describing common or variable requirements is a specific label which identifies a requirement as common “C” or variable “V.” The next figure shows an example document of FAST for the domain of Car radios [Doerr02].

Commonality

C-1 A cassette player is mutual exclusive to a CD player.

C-2 The disc naming feature requires a CD changer
… …

Variability

V-1 Some products have an MP3 decoder

V-2 The number of supported discs in the changer varies
… …

Figure 7 : FAST Document Example

2.4.2 Comments on the Example

The different types of requirements in the above shown example represent different types of variability and decision constraints. The common requirement C-1 describes a “mutual-exclusive” constraint and C-2 specifies a “requires” constraint. The variability V-1 is an “option” and V-2 represents and “alternative,” without specifying the exact value of the variable component. This specification is performed with the help of the “parameters of variation,” which are documented in another section of the commonality analysis document. The “parameters of variation” section is a specific table which describes the range of variation for the different variabilities. In addition, the binding time of the decision and a default value are specified. The respective entry for the variabilities V-1 and V-2 would be:

Parameter of Variation	Value Space	Binding Time	Default
MP3 decoder	Yes, no	Specification time	no
CD Changer Capacity	6,10	Specification time	-

Table 2 : Parameters of Variation

In the introduction it was stated that FAST uses a domain specific language for describing the software specification. This issue refers to the concepts and terms used in the description of the commonalities and variabilities.

2.4.3 Summary

As shown in the above example, FAST uses a strong textual based representation form for describing commonalities and variabilities. As nearly every type of terms and identifiers can be used for the specification, the method seems to be appropriate for many different domains. As already addressed in the summary part of FODA, FAST is also only a description method for domain models and thus supports only the domain analysis part of a process family engineering approach. Varaibility modeling with generic artifacts and application engineering is not supported by FAST.

In addition to that, especially in the case of strong technical based software processes or products with a large specification the documentation complexity increases very fast. The reason is that domain specific aspects must be described with natural language. In addition to the increased complexity, the chance of ambiguities is also increased. Besides the

ambiguous requirements, interrelations between different functionalities or requirements are also hard to detect. Domains with many interacting features or system components are hard to describe with FAST.

As discussed above, the applicability of FAST for generic process families is not given as to many aspects of a process family engineering approach are not covered.

2.5 PuLSE (Product Line Software Engineering)

PuLSE is a method for enabling the conception and deployment of software product lines within a large variety of enterprise contexts. This is achieved via a product-centric focus throughout its phases, customizability of its components, an incremental introduction capability, a maturity scale for structured evolution, and adaptations to a few main product development situations. The domain models captured during the development process and the identified variabilities in the domains provide the characteristics and features for a domain-specific system.

In the following, we briefly introduce the PuLSE components and the impact of generic process orientation to the PuLSE components. Figure 8 shows an overview of PuLSE.

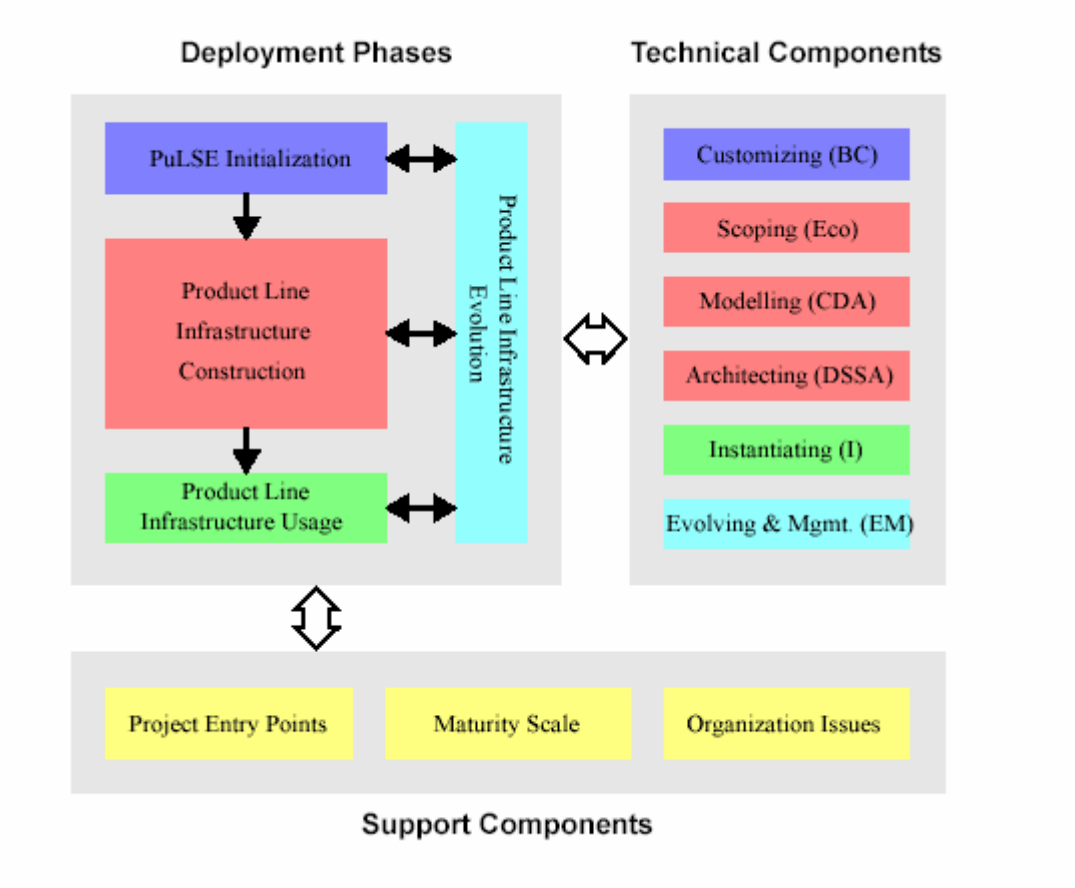


Figure 8 PuLSE overview

PuLSE is centered around three main elements: the deployment phases, the technical components, and the support components.

2.5.1 Deployment Phases

The *deployment phases* are the logical stages of the product line life cycle. They describe the activities performed to set up, use, and evolve product lines. The impact of generic processes to the deployment phases is not significant, as the phases can be applied in the same way for service-oriented applications with generic processes as for products. Therefore, we only briefly introduce the deployment phases as described in the initial PuLSE method. The deployment phases are **PuLSE initialization phase**, which is used to customize PuLSE to the context of its application, the **Product line infrastructure construction phase**, which sets up the product line infrastructure by scoping, modelling, and architecting the product line, the **Product line infrastructure usage phase**, which uses the infrastructure to create a single product line member, and finally the **Product line**

infrastructure evolution phase that handles the evolution of the product line.

2.5.2 Technical Components

The *technical components* provide the technical know-how needed to operationalize the product line development. They are used throughout the deployment phases. Thus, the impact of process family and service orientation is inherent to these components. The technical components will experience the most significant impact of service and process family orientation.

The initial technical components are:

Baselining and Customization (PuLSE-BC)

This component baselines the enterprise and customizes PuLSE. The result is an instance of PuLSE — that is, instances of the other technical components — tailored to the specific application context.

Economic scoping (PuLSE-Eco)

PuLSE-Eco is used to identify, describe, and bound the product line. This is done by determining the characteristics of the products that constitute the product line. Economic scoping in PuLSE means that the scope is determined with respect to business objectives and planned products. The output of PuLSE-Eco are the product characteristic information and the scope definition. These outputs together describe the contents of the product line. The product characteristic information describes the common and variable characteristics of all products in the product line. The scope definition identifies the range of characteristics that systems in the product line should cover. The basis for the scope definition is a product map that relates the characteristics to the different products. A product map is a table, which lists the characteristics mentioned in the product characteristic information as its rows and the products as its columns. The table cells contain a cross when a product contains a characteristic.

At the moment, product maps are only used for products and functionalities. Concerning the goal to develop service-oriented applications, the product map must be modified. The modifications require that the map does not relates products and functions, but service and processes. A possible service map is shown in the picture below.

	Service: User Identification	Service: Availability Check	Service: Credit Payment
Process: Book Reservation	Yes	Yes	No
Process: Flight Reservation	No	yes	Yes

Table 3 - Service Map

The example shows two reservation processes, one for books in a library and one for flights. As it is shown in the table, the service to identify a user at the beginning of a process and the service to check the availability is required by the book reservation process. The flight reservation process also requires an availability check service, but it is not needed to have a user account, but therefore you can pay with a credit card. This is just a small example of how to analyze domains for service-oriented applications.

Customizable Domain Analysis (PuLSE-CDA)

The elicitation of the requirements for a domain and the documentation of them in a domain model (a.k.a. product line model) are done by PuLSE-CDA. A product line model is composed of multiple workproducts that capture different views of a domain. Each view focuses on particular information types and relations among them. In the workproducts, common requirements (commonalities) and requirements that vary for the different systems (variabilities) are modeled. Therefore, they are referred to as generic workproducts. There are three types of variabilities: optional, alternative, and range requirements. Each generic workproduct has defined meta elements for each variability type. Meta elements indicate points of variation and enable the instantiation of the workproducts. The variabilities (expressed by meta elements) are connected to decisions that, when completely resolved, specify a particular system, a member of the product line. The decisions are at different levels of abstraction and are hierarchically structured based on constraints among them. The decision hierarchy is called the domain. To specify a particular system in the product line, the product line model is completely instantiated. The instance of the product line model is generated by passing all resolutions of the decisions to the connected meta elements, which instantiate their corresponding part of the product line model.

As already discussed in the domain engineering section, an emerging issue is how to model and notate generic process models and especially the variable parts, which means the generic artefacts, of the generic model. The

meta elements described above will influence the most significant modifications as they have to be adapted to express process related information. It is also necessary to analyze, to which degree, additional variabilities must be included in a domain model. At the moment only optional, alternative and range requirements can be expressed.

Domain Specific Software Architecture development (PuLSE-DSSA)

The development of a reference (or domain specific) architecture based on the product line model is done by PuLSE-DSSA. A reference architecture description consists of multiple models that describe different views on the reference architecture. Each of the views is composed of view-specific components and connectors that describe the architecture from a different perspective. Similar to a product line model, a reference architecture description is an architecture description that also captures variability in the architectures for the different systems in the product line. During the reference architecture development, certain decisions arise that are not driven by the domain. These decisions may introduce domain-independent variabilities. The resulting decision model is called the architecture decision model. An optional output of PuLSE-DSSA is a prototype that may have been created.

The impact of process orientation to this component is especially during the view creation phase of PuLSE-DSSA. As stated above, a reference architecture description consists of multiple models that describe different views on the reference architecture. Additional views describing the processes and the service mapping must be included.

Instantiation (PuLSE-I)

The Instantiation component is used to specify, construct and validate one member of the product line. This encompasses the instantiation of the product line model and the reference architecture, the creation and/or reuse of assets that constitute the instance, and the validation of the resulting product. Additionally, reusable assets that are needed, that have not been created yet, are developed and put into the reusable asset base.

This phase is not affected much by the process family orientation as the required information for generic process families must be provided by the above introduced components. If the required information is available, an instantiation process will work as for product families.

Evolution and Management (PuLSE-EM)

Guidance and support of the application of PuLSE throughout the deployment phases initialization, construction, usage, and evolution is done with PuLSE-EM. PuLSE-EM is centered around three basic tasks: product line management, evolution, and learning. Product line management provides means for scheduling and coordinating the technical components, as well as for observing the product line and its environment to be able to respond quickly to emerging needs. Product line evolution supports systematic change request processing. This includes the evaluation of change requests and the assessment of their effects on existing parts of the product line infrastructure. Learning analyzes the product line and changes that occur over time. The goal is to learn about patterns of product line evolution that would allow for acting in anticipation of future problems, needs, or changes. Additionally, PuLSE-EM includes the configuration management framework that underlies and supports the product line infrastructure.

This component requires no changes for generic process support.

2.5.3 Support Components

The *support components* provide guidelines that support the other components. The support components need not to be changed to support generic process families:

Project Entry Points

Project entry points are guidelines to customize PuLSE for a set of standard situations. For example, in reengineering driven PuLSE projects, legacy assets are a major source of information and guidelines on how to integrate them are given in the respective entry point.

Maturity Scale

It is used to evaluate the quality of a PuLSE process application in enterprises with the intention to identify and improve weak points. The levels on the scale are: initial, defined, controlled, and optimizing.

Organizational Issues

For PuLSE to be most effective, an organization structure has to be set up and maintained that supports the development and management of product lines. Guidelines on how to do that are given here.

2.5.4 Generic Process support in PuLSE and summary

When talking about families of processes or generic processes for service-oriented application development, the product-centric focus of PuLSE is obviously not sufficient to cover the relevant aspects introduced in section 2.1. Process relevant aspects must be covered throughout the whole development cycle. But as PuLSE is only a framework for Product line engineering it seems to be capable for generic process support. The changes that must be made for generic process support are maily inside of the individual components of PuLSE. The changes, which have to be done have been briefly discussed in the respective descriptions of the PuLSE components. The techniques for incremental product line engineering, introduced in section 2.2, variability modelling, decision modelling and application engineering, are already an integral part of PulSE and supported by the different technical components of PuLSE.

To summarize the above discussion, PuLSE directly supports incremental product line engineering and can be customized towards supporting process family engineering by modifying the technical components.

2.6 The KobrA Method

The KobrA method represents a synthesis of several advanced software engineering technologies, including product line development, component-based software development, frameworks, architecture-centric inspections, quality modelling, and process modelling [Atkinson01]. These have been integrated into the KobrA method with the basic goal of providing a systematic approach to the development of high-quality, component-based application frameworks.

All products are organized around, and oriented towards, the description of individual components. This means that, as far as possible, there are no global or system-wide products - all products (and accompanying processes) are defined to carry information only related to their particular component. The advantage is that components (and the products that describe them) can then easily be separated from the environment in which they were developed and therefore can be reused independently.

From a product line perspective, the KobrA method represents an object-oriented customization of the PuLSE method. The infrastructure construction phase of PuLSE corresponds to the framework engineering activity, the infrastructure usage phase of PuLSE corresponds to the application engineering activity, and the product line evolution phase of PuLSE corresponds to the maintenance of the frameworks and applications. The purpose of the framework engineering activity is to create, and later maintain, a generic framework that embodies all product variants that make up the family, including information about their common and disjoint

features. The purpose of the application engineering activity is to instantiate this framework to create particular variants in the product family, each tailored to meet the specific needs of different customers, and later to maintain these concrete variants. A given framework can therefore be instantiated multiple times to yield multiple applications. In fact, the framework and application engineering activities both result in descriptions of components in terms of a mixture of textual and UML-based (graphical) models. The difference between the two is that the framework models potentially contain variabilities, while the application models do not. The advantage of using the UML is that the KobrA method is model-based and, therefore, frameworks and associated application are independent of any particular programming language or component technology (e.g., Java Beans, COM, CORBA). The transformation of an application into an executable form is carried out in a distinct set of activities that are essentially orthogonal to the framework and application engineering activities. The implementation activity takes instantiated UML models and maps them, through a series of well-defined refinement and translation steps into an executable representation (e.g., high-level source code) [Bunse01]. Finally, the build activity actually creates binary load modules ready for deployment in the target environment.

2.6.1 Framework Engineering

In the KobrA method, a framework is the static representation of a set of components organized in the form of a tree. Each component is described at two levels of abstraction: a specification, which defines the component's externally visible properties and behaviours, and thus serves to capture the contract that the component fulfils, and a realization, which describes how the component fulfils this contract in terms of contracts with lower level components. A framework, therefore, is a tightly coupled arrangement of component specifications and realizations.

Figure 9 shows the general set of UML models, which make up component specifications and realizations. To start the framework development process, the context of the component at the root of the tree is modelled. Since this takes the form of a realization it is known as the context realization. Subcomponents are then identified, their specifications derived from the context realization models, and finally the subcomponents realizations are designed. This is performed recursively until no further subcomponents are required. The framework is a reuse infrastructure for creating systems within the application domain. The family aspects are captured by decision models, which are a part of all specifications and realizations. The decisions relate to variabilities in the domain that are explicitly reflected in the models of the generic framework. The explicit modelling of variability is done using stereotypes in UML diagrams and tags in textual models.

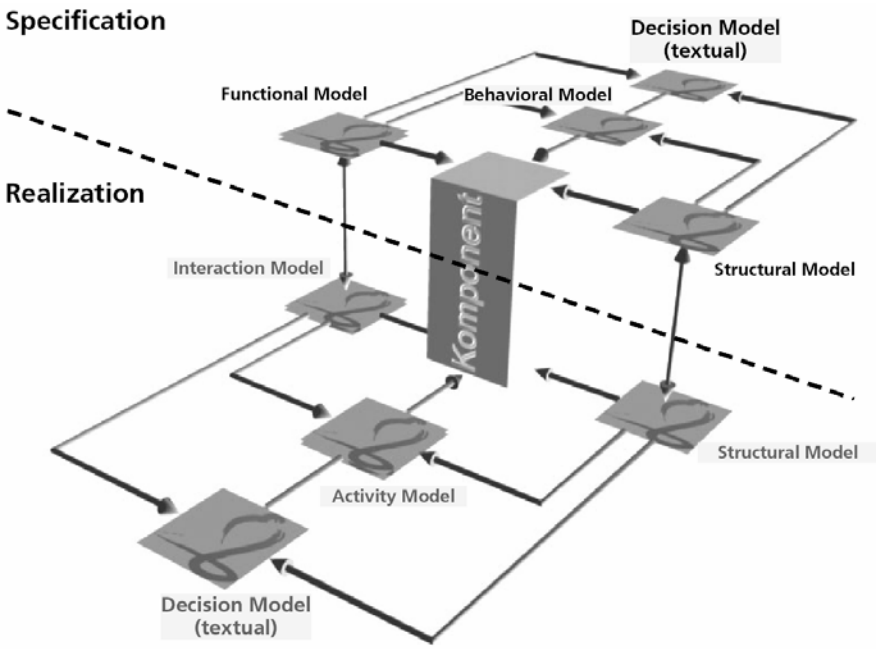


Figure 9. KobrA Component Model

Context Realization

Framework engineering starts with the elicitation of the environment properties for the planned system family, including the determination of the framework's scope. However, the application of KobrA requires a particular set of models at the end of context realization, which is needed to begin the recursive KobrA development process. These models correspond to the models used for realizing components. The current focus of modelling is set on enterprise resource planning applications. Therefore, enterprise models that capture the important concepts and processes of an enterprise application are the starting point for developing the context realization. The other models used are the same as in a component realization.

Component Specification

The goal of component specification is to create a set of models that collectively describe the externally visible properties of a component. As such, the specification can be viewed as defining the interface of a component and describing the services a component provides to its parent. The specification of a component is comprised of four main models: the structural model, the behavioural model, the functional model, and the decision model. The structural, behavioural and functional models constitute

the specification models for a component as it is used in all applications covered by the framework. The decision model contains information about how the models change for the different applications.

The specification of a component is comprised of the following four models:

- Structural Model: Captures the nature of the classes and relationships by which a component interacts with its environment, as well as any structure of the component that is visible at its interface. The structural model consists of a number of UML class diagrams that captures the externally visible structural elements a subject component interacts with and a number of UML object diagrams that capture the externally visible parts of run-time configurations of a component and the components it acquires.

- Behavioral Model: describes the reaction of the component to external stimuli using UML statechart diagrams.

- Functional Model: addresses the functionality of a component by describing the externally visible effects of the services provided by the component. The behavioral model contains an operation schema for each of the services a component provides.

- Decision Model: The structural, behavioral and functional models constitute the specification models for a component. If the component is a generic product line component, an optional decision model contains information about how the models change for the different instances of the product line component.

Component Realization

The goal of component realization is to create a set of models that collectively describe the private design of a component. As with all design, the basic requirement is that the realization must realize the component's specification. A component's realization is comprised of four main models: the interaction model, the structural model, the activity model, and the decision model.

A component’s realization is comprised of the following four models:

- Structural model: captures the classes and relationships from which the component is realized, as well as its architecture. The realization structural model is a refinement of the specification structural model. It consists of a number of UML class diagrams that capture the structural elements a service component interacts with and a number of UML

object diagrams that capture the run-time configurations of the component and the components it interacts with.

- Activity model: covers the realization of the functional aspects by describing the algorithms by which the services of the component are realized using UML activity diagrams.

- Interaction model: provides different aspects on the algorithms used to realize operations, from the perspective of instance interactions rather than flow control (as in the execution model). UML collaboration diagrams are used in the interaction model.

- Decision model: as in the specification, the optional decision model describes the model changes for the different instances of a product line component.

Another possible way of realizing a specification is to reuse pre-existing components such as COTS components or reengineered legacy components. To achieve this, parts of the specified interface are matched to the interface supplied by the pre-existing component. When the two interfaces are the same they are said to be in "mutual interface" agreement and the supplier component can be integrated in the component framework. If the two interfaces are not initially the same, changes must be made to the reused component and/or the client component in the framework.

2.6.2 Application Engineering

Application engineering uses the framework built during framework engineering to construct specific applications in the domain covered by the framework. The application engineering process is centred on the given framework and driven by the framework's decision models. The framework is traversed in a top down manner, recursively resolving decisions until all the generic framework models are transformed into specific models for the particular application. According to the common separation of requirements engineering and system design, the application engineering process is split into two primary steps: context realization instantiation and framework instantiation. These are described in more detail in the following two subsections.

Application Context Realization

The instantiation of the framework's context realization is the first major activity of application engineering. It starts when the software development organization has established an initial contact to a potential customer who is interested in a software system in the domain of one of the organization's

frameworks. The outputs of this process are the context decisions and a concrete realization of the application's context. Ideally, a consultant handles interaction with the customer during this activity. The role of a consultant is played by a person who is an expert with respect to the application domain and to applications based on the existing framework. The consultant elicits the requirements for the application to be developed while working with the customer to identify problems.

The elicitation process is driven by a decision sequence derived from the decision model of the framework's context realization. This strategy for requirement elicitation is tightly coupled with the framework because exactly the alternatives supported by the existing framework are provided to the customer. The offering of a set of possible alternatives also simplifies the elicitation process because it corresponds to the selection of one of the provided choices. Only when none of the supported alternatives meets the customer's needs must the required properties be explicitly modelled during requirement elicitation. The framework alternative that is the closest to the required one serves as the input for the modelling activity. Hence, the alternative not yet supported by the framework can be expressed by means of differences to requirements supported by the existing framework. This approach supports the incremental product line engineering described above.

When all decisions in the decision model of the framework's context realization have been resolved, the main phase of the elicitation process is finished. The result is a concrete instance containing a set of models that realize the context of the particular application to be developed. In addition to the instances of the generic framework models, customer-specific requirements that are not part of the framework can be added to extend the application context realization.

Framework Instantiation

The instantiation of the framework is the second major activity of application engineering. It starts when the application context realization is (partially) created and thus also the context decisions (partially) exist. The context decisions are used to initially instantiate the generic component hierarchy of the framework. This is achieved by identifying decisions at lower levels in the component hierarchy that are connected to decisions resolved during the instantiation of the framework context realization. These lower-level decisions are then resolved in accordance with the resolution of the connected context. The intermediate result is a partially instantiated component hierarchy which is an application tree with unresolved points of variation, and decision models that contain the still unresolved decisions. These unresolved decisions relate either to design-related issues or user requirements that have not been handled during requirement elicitation. Both

kinds of unresolved decisions are fed back to the consultant who is responsible for their resolution. The consultant resolves them either personally, together with the customer, or together with the developers.

All resolutions are collected as decisions in the appropriate place in component hierarchy. In addition to the resolution of the decisions provided by the decision models of the component hierarchy, customer-specific requirements must be realized and therefore integrated into either the framework or the instantiated models of the particular application. If it is expected that other customers in the future will have the same requirements, the generic integration of the realization of customer- specific requirements is the preferred alternative. The determination of whether the framework can support the new requirements must, in general, be performed by the organization. If the new requirements are integrated into the framework, there will be a decision in the framework concerning the new requirements. The application engineering process then resolves the new decision and instantiates the new framework models so that the new requirements are part of the application tree. On the other hand, if the new requirements are not integrated into the framework, they must be modelled exclusively for the particular application in hand and integrated into the already instantiated framework models. The decision models support the integration process by indicating where in models points of variation already exist and where there are similar variants integrated or attached to the framework models.

Throughout the whole instantiation of the component hierarchy, consistency between adjacent layers, as well as the internal consistency of each specification and realization must be ensured. When no unresolved decision points are left, all customer-specific requirements are separately modelled and integrated and the application has successfully passed all quality assurance activities, the application engineering process is finished. The final results are the application decisions consisting of the context decisions and the component hierarchy decisions, together with the application realization and the application tree.

2.6.3 Domain Engineering Techniques in KobrA

The KobrA method is a complete product line engineering method that supports the development of generic frameworks of components. These generic frameworks can play the role of a product line infrastructure alone. The KobrA method specifically supports the three domain engineering techniques listed above. Variability modelling is supported by the UML conformant use of stereotypes to denote variability in UML models and by tags in textual models. Decision modelling is realized by the tabular, hierarchical decision models used in the KobrA method during the application engineering process.

2.6.4 Generic Process Support in KobrA

The KobrA method uses mainly UML diagrams for modelling generic components. The used models include activity diagrams in the realization of components. These diagrams, however, play a specific role in the modelling of components, that is, to capture the algorithms that realize operations a component provides. For the use of the KobrA method for process-based definition of families of applications, the existing activity diagrams can, therefore, not be used. Instead the context realization must be adapted to support the process-based definition of generic component frameworks. This can be done by replacing the currently present (rather simple) means to capture processes by appropriate process models. Using the present techniques for modelling variability and decisions, such integration can easily be done. The remainder of the KobrA method need then not to be changed.

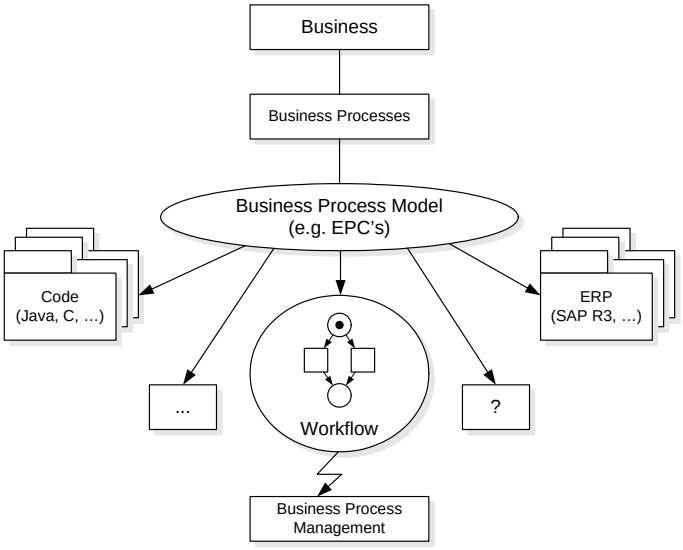
2.6.5 Summary

As discussed above, KobrA directly supports the domain engineering techniques and can be customized towards also supporting generic process families.

3 Requirements of the E-Business Domain

As other strong growing domains, the e-business domain is under continuous change and development. There is even no commonly agreed definition on the term e-business [Schildhauer03, Lück04, Mertens01]. The roots reach back to 1997, where IBM used it as a derivation of the terms e-mail and e-commerce. It resembles the conduction of business in the internet including buying and selling but also services and collaboration. In a broader sense, it deals with the application of modern computer systems and networks for business purposes.

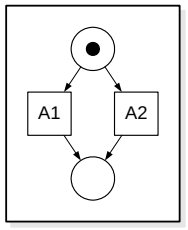
Figure 10: **Overview**



E-Business can be established by the use of varying techniques. Mail and telephone could be replaced by e-mail and voice-over-ip systems, warehouses could be managed electronically and even the whole accounting could be outsourced to another company with direct and immediately access. As the scope of the PESOA project is process-orientation, the business processes a company has are of main interest (see Figure 10). The business processes could be made explicit in so called business process models [Becker03]. Typical examples are Event-driven Process Chains (EPC) as proposed by the ARIS method from IDS-Scheer [Scheer01] or the newer Business Process Modelling Notation (BPMN) as proposed by the Business Process Management Initiative [BPMN]. Possible implementations of such business process models into the it-infrastructure traditionally resulted in specialised programs written in standard programming languages like Java or C or pre-selected process parts in

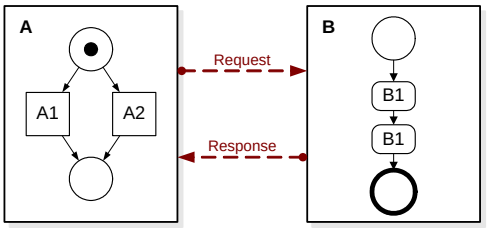
ERP-systems such as SAP/R3. All those solutions contain the business processes implicit in their code. But as the data-management has been split from the actual code in the 70's by the emerging of databases and the user-interface management has been brought apart in the 80's by the use of standard GUI's and frameworks, the explicit representation of the processes will gain further advantages. As a programmer has no need to code data-management and graphical interface runtimes for his product, a workflow engineer will have no effort in implementing the business processes into the it-infrastructure. He can use his potential to specify, define, optimize or administrate the processes. After having used the term workflow, it is defined as *"the computerized facilitation or automation of business processes, in whole or in part"* [Hollingsworth95]. This definition was given by the Workflow Management Coalition (WfMC), which also defined a reference model that is introduced later on. The term Business Process Management (BPM) resumes the concepts of mapping business processes to business process models and their workflow representation which can be used for execution, simulation, resource planning, optimization, and so on. Those workflows are the key issues of the PESOA project in the context of the e-business domain. Different but yet similar workflows should be generated by the use of Software Product Lines to adapt to specific customer needs in short times.

Figure 11: **Simple Workflow**



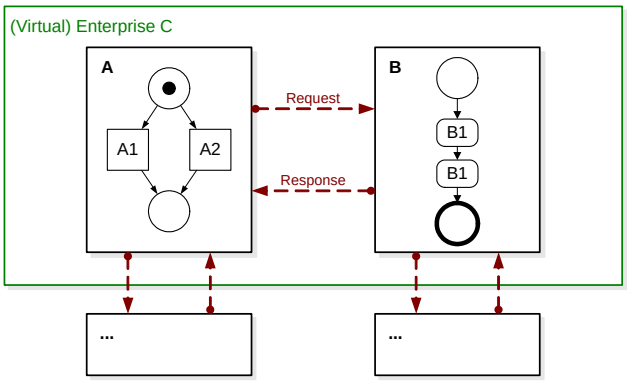
After introducing the general concepts of the e-business domain in the PESOA project, further key issues will be depicted. Figure 11 shows a simple workflow that a company might have. In this example, the workflow is represented as a Petri net. The company is represented by a box around. The workflow represents the parts of a business process that can be executed and supervised electronically. As the processes are made explicit, they can be recognized and changed easily. The workflows inside a company are also called intra-organizational workflows to distinguish them from workflows between different companies.

Figure 12: **Inter-organizational Workflow**



This case is shown in Figure 12. There is the introduced company, now labelled A and a new one, labelled B. Company B represents its intra-organizational workflow by the use of the Business Process Modelling Notation (BPMN). The companies communicate to work together and achieve an outcome. The process of the interaction between different companies that could involve different ontologies and process languages is called inter-organizational workflow.

Figure 13: **Blurring borders between intra- and inter-organizational workflow**



Unfortunately it is not that easy, as shown in Figure 13. A and B could be seen as different departments of an enterprise or as two companies that have been merged and now form a kind of virtual enterprise. This formation has internal and communication processes, but to the outside world it looks like a single company, represented by the box around. There is also communication with companies outside, represented by the boxes below.

The key issues of workflows are the activities or tasks. Those are steps that have to be done to produce an outcome, for instance A1 and A2 in Figure 11. They require an ordering, input data and humans, machines or computers to accomplish them. The Workflow Management coalition established the mentioned reference model that describes the execution environment of workflows. Workflows also have different perspectives, or aspects. They refer to the things that are required to execute a workflow and will be discusses later on. An in deep description of the topics around workflow can be found in *Production Workflow* [LEYMANN00], whereas

Business Process Management – The Third Wave [SMITH02] gives an outlook on future requirements.

Even as the term e-business is somehow fuzzy, it can be divide into three main topic areas. First, there are internal business systems, which resemble classical business processes like customer relationship management (CRM), enterprise resource planning (ERP), knowledge, document management, and so on. Those processes are called intra-organizational workflows. This workflows form the key issue of the domain e-business in the PESOA project. Another important topic in the area of e-business is enterprise communication and collaboration. This includes e-mail, forums, chat-systems, black-boards, conference and other collaborative work systems. As those are usually used in an ad-hoc fashion, without pre-defined processes, they are out of scope in the PESOA project. The third topic is electronic commerce; it includes business to business (B2B) and business to customer (B2C) relations. This topic resembles the communication aspect between different companies or market participants. As stated, there is no clear boundary between processes in and outside of a company. However, the interoperation of companies with others is denoted as inter-organizational workflow in contrast to intra-organizational workflow. Even the definition of the term e-commerce is somehow ambiguous. The Lexikon Electronic Business states that there exists no clear, explicit definition [Schildhauer03]. It can be seen as the digital execution of business processes between companies and customers by the use of global public and private networks [Mertens01].

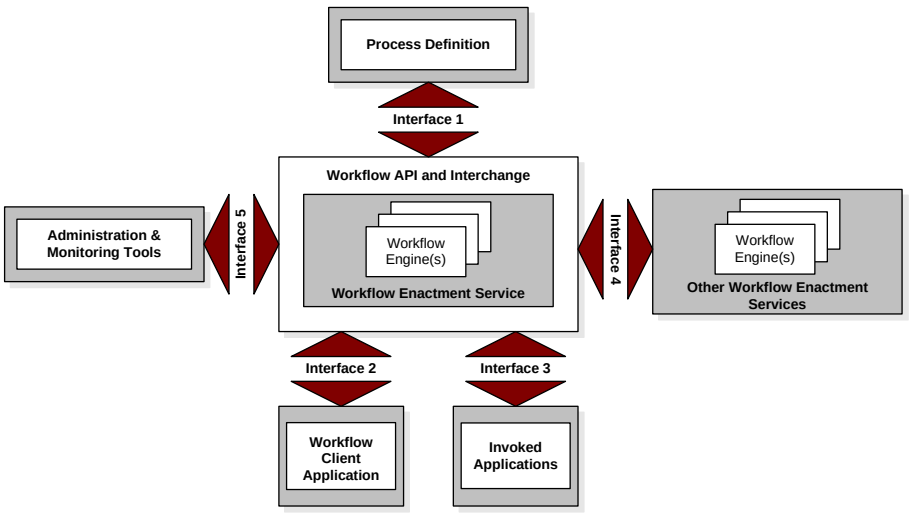
The last topic might become important for the PESOA project in the near future. In the service-oriented world, as propagated by the W3C, there is a difference between orchestration and choreography of web-services, which resemble the concepts of intra- and inter-organizational workflows [W3C]. Furthermore, several levels of contracts between different inter-organizational workflows could be established. They could regard the (1) syntactical and interface level, (2) behaviour represented by pre- and post conditions, (3) synchronizing contracts that resemble dependencies between different services companies offer, and (4) quality of service contracts which represent non-functional properties like response, throughput and costs. In a broader sense, there exists a bunch of standards for each area such as OWL [OWL], WSDL [WSDL], UML/OCL [OCL], CTR-S [Davulcu98] and so on. Other issues include transactions, as workflows can be seen as some kind of long-running transaction [Haugen02, Papazoglou02]. Also security must be regarded through all levels of abstraction, starting from an agreement of security goals, the security analysis, design and policies as well as network access and authorization down to protocol implementation issues. Transactions and security are out of scope in PESOA as they form a broad research area of their own. The other areas of e-commerce might come into play as needed.

The introduction to the e-business domain is continued with the workflow reference model that resembles the core requirements for workflow.

3.1 Workflow Reference Model

In 1995 the Workflow Management Coalition published issue 1.1 of their Workflow Reference Model [Hollingsworth95]. It describes a so called Workflow Management System that allows the explicit representation, controlled execution and monitoring of workflows. It provides the procedural automation of business processes and is thereby a core element of e-business from. Even if the Workflow Reference Model is a bit outdated, as in the time it was written the terms e-business or service-oriented architectures were quit unknown, it defines main components that are required for workflow enactment that still hold today.

Figure 14: **The Workflow Reference Model**



The reference model is shown in Figure 14. The core component is the workflow enactment service. This is a service that consists of one or more workflow engines and is responsible for creating, managing and executing workflow instances. Applications may use the workflow application programming interface (WAPI) to communicate with this service. A workflow engine provides run time environments for workflow instances. The workflow engine typically handles the interpretation of process definitions, controls the process instances, creates workitems, calls external applications, and so on.

The workflow enactment service has five different interfaces to external components as shown in the reference model. The first interface describes the connection to a workflow definition tool. This tool or a set of tools allow

for workflow analysis, design and definition. The workflow definition is converted to an interchange format and transferred to the workflow enactment service. This interface is the hot spot for the PESOA project in the e-business domain. The goal is to create executable workflow definitions which are generated out of a product-line. The type or format of these definitions depends on the workflow engines that are used by the customers. A modern standard is the Business Process Execution Language for Web Services (BPEL4WS) [BPEL] which is XML-based. The remaining components depend on the workflow definition but are out of scope in PESOA as there are a lot of different commercial solutions available on the market. The components will be outlined for a better view of the big-picture.

The interfaces two and three require some fetch-ahead of the next section, Workflow Aspects. Interface two defines a connection to Workflow Client Applications. Those are applications that show worklists consisting of workitems for particular persons. Each workitem represents a task that has to be done by some employee. The allocation of workitems to specific employees is part of the workflow enactment service. Interface three specifies how external applications that are required to perform a task could be invoked. This might for example be a printjob or something else.

Interface four defines workflow interoperability by passing workitems seamlessly between different workflow enactment services. This is useful if other departments or companies run their own workflow enactment services and interaction is required.

The last interface, interface five, defines the integration of administration and monitoring tools. This includes user and role management functionality, audit operations, resource control operations, process supervisory functions and process status functions.

Even as the Workflow Reference Model is quite old and lacks a lot of modern techniques and developments, it is useful to show the connection of Workflow Management Systems, which form a core element of e-business, to the PESOA project. A detailed description of the reference model can be found in [Hollingsworth95].

3.2 Workflow Aspects

Workflow modelling aims at specifying different perspectives of the activities that have to be done to produce an outcome, the ordering of the activities and the technical and organizational environment. Those different perspectives of workflow are made explicit by different workflow aspects.

The first thing is *what* has to be done; those are the steps or activities that are required to produce an outcome. This *what* can be hierarchically refined into smaller *whats* or *steps* until an atomic level is reached that should or cannot be parted anymore. This aspect is the *functional aspect* and it consists of complex and atomic activities. The whole workflow can be seen as a complex activity.

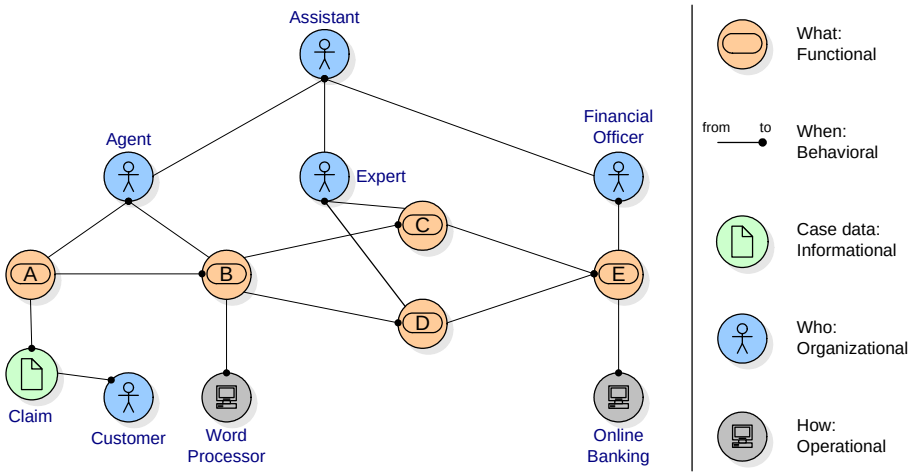
The activities require some ordering that defines how they are related with each other. For example, an activity pay invoice should only be executed after an activity check goods has been executed. The aspect that regards the ordering of activities is called the *behavioural aspect*; it can be represented by control flow. The behavioural aspect also defines other forms of relationships within a workflow like what a starting and what a terminating activity is.

Another important aspect is the modelling of workflow relevant application data. This is covered by the *informational aspect*. Each activity gets a set of input and output parameters assigned that are used to transfer the workflow relevant data through the process. The so called data--flow must not necessarily follow the control--flow. The initial data of a workflow instance is usually a specific case which has to be processed.

Workflows are not executed by themselves; this is done through people or machines that are categorized into different roles. This is called the *organizational aspect*. This aspect defines which role is responsible for executing an activity and the workflow engine has to resolve the roles to specific persons or machines.

For the successful execution of activities, tools and applications outside of the workflow engine are needed. The integration is covered in the *operational aspect*. It defines how external programs like Office, E--Mail or Online--Banking are integrated into the workflow.

Figure 15: **Example of the Workflow Aspects**



A detailed description of the workflow aspects can be found for instance in [Weske98,Weske00]. Figure 15 shows an example of how the different workflow aspects tie together. To abstract from certain existing notations, a simple approach based on circles that represent the different workflow aspects is used. The behavioural aspect is represented by a line. A legend is contained in the right hand side of the figure.

The example represents a simplified insurance claim. The workflow starts with activity A where a customer assigns a new claim. The claim is initially attached to the input of the first activity; it represents the case-data. Each claim belongs to a customer that is represented by the organizational aspect. As the process evolves, the claim is passed on to activity B and so on. Activity A is processed by a person of the role Agent. The same role processes activity B, where a letter to the customer is prepared that states that the claim is currently being processed. It requires the use of an operational entity, a word processor, to write the letter. After the letter has been written, the claim is analyzed by two independent members of the role Expert. This is represented through the activities C and D. If the claim has been accepted, a member of the role Financial Officer uses an online-banking application in activity E to initiate a payment. Note that several activities like writing a final letter or the reject of the claim are not modelled to keep the example simple. Additionally contained is the hierarchical grouping of the roles Agent, Expert and Financial Office into a role Assistant.

From the view of the PESOA-project, one important aspect is missing – an aspect that covers variability. This aspect has not yet been investigated as a workflow aspect. One goal of the PESOA-project in the e-business domain is the formulation of this aspect. Today there exist many similar workflows in different implementations at the customer's side. Often this is just one "master" workflow with variants. The pieces, or assets in the product line domain, consists of activities or compound activities like write a letter, charge a customer, backorder items, etc. Other assets might include role definitions, operational properties and data-containers. It is one goal of the PESOA-project to develop new methods and technologies that allow the efficient construction of workflows from an asset base. This is why variability is an important aspect for workflows in the PESOA domain and must be researched further on.

The behavioural workflow aspect can be seen as the most import one. As shown in figure 3.6, all other aspects follow this one. Activities without an ordering are useless as well as case or role information are. One could also see the functional aspect that is represented by the activities as central. However, this aspect is implemented by standard techniques and is therefore not this interesting for the PESOA-project.

Different types of behaviour in workflows have been collected as Workflow Patterns [van der Aalst00]. The main contributor, Wil van der Aalst has established a website where all patterns can be experienced interactive [WP]. The patterns are split into six categories. The first category covers basic control flow patterns like sequence, AND and XOR joins and splits. The second category contains advanced branching and synchronization patterns like OR join and split a discriminator and n-out-of-m joins. A n-out-of-m-join is a special kind of OR joins that waits for n out of m incoming paths before it continues. The third category describes structural patterns; those are arbitrary circles and implicit terminations. The forth category deals with pattern that involve multiple instances. The fifth category contains state-based patterns like a milestone, whereas the last category deals with cancellation patterns. The Workflow Patterns have been used to compare different workflow notations. The more patterns a notation supported directly, the better it was rated. More information can be found in [van der Aalst00, van der Aals03, van der Aals02b] and the website mentioned.

Another important kind of patterns are Communication Patterns that are based on the informational aspect. Those patterns are used to describe the interactions between companies – that is inter-organizational workflow. Those companies use messages and thereby data-flow to communicate. There are synchronous communication patterns like Request/Reply, One-Way or synchronous polling and asynchronous communication patterns like Message Passing, Publish/Subscribe and Broadcast [van der Aalst02a].

Another pattern – the Interruptional Pattern – interrupts a running activity by the use of communication and specifies a new outgoing control flow. This pattern can be found in modern notations like BPMN and BPEL, and can also be used to model exception and cancellation handling. It combines behaviour as well as communication represented by the informational aspect. This pattern has not been investigated well, although it can be found in recent notations like the BPMN where intermediate events are placed on the border of activities.

3.3 Formal Workflow Representation

After having introduced different workflow aspects, it has to be specified how to represent them. From the PESOA project description [PESOA] it can be assumed that a graphical notation to create variant process models as well as at least one execution language is required. They will be considered and evaluated in a separate chapter. This section considers another problem – that is the sometimes imprecise syntax and more often the ambiguous semantics of graphical workflow notations. Current notations that combine several workflow aspects, like UML activity diagrams, BPMN or EPK's, are only specified semi-formal. An informal approach might be easier to understand by non-expert users but has disadvantages in the generic E-Business domain as well as for the PESOA project. The PESOA project requires the automated code generation by the use of product lines, whereas many graphical notations are mainly used for human-communication which does not require formal foundations.

By having an explicit representation of the syntax and semantics of workflow, several key issues in the E-Business domain regarding the PESOA project could be solved. The most important issue is the mapping of graphical workflow notations to executable languages or other notations. This regards to the required code generation of the PESOA project. If a computer wants to transform a graphical workflow notation into an executable notation like BPEL, he needs an unambiguous, formal definition. By providing this, an automated mapping will give the results expected, while otherwise the results are unpredictable or the mapping will be impossible. The key point that needs to be done is selecting a consistent subset of an existing notation and adding a formal syntax.

The second important issue covers the reasoning about workflows. In addition to a formal syntax, this issue requires a formal semantic of the workflow. The first point is the correctness of workflows [van der Aalst02b]. This includes qualitative and quantitative aspects. The qualitative aspects define the so-called Soundness criteria that allow the proving of deadlock, lifelock and proper termination. Soundness is a key issue in workflow nets [van der Aalst97] but could be mapped to other notations as well. The

quantitative aspect covers things like performance, time and level of service. If a workflow conflicts with those criteria it can not be mapped and executed correctly or will show false behaviour. An adequate formalization allows avoiding those errors by proving the correctness regarding qualitative and quantitative aspects. Another point covers constraints or properties related to single activities. If each activity has formal defined pre- and post conditions that do not only regard the syntax but also the semantics, the consistence and completeness of a workflow could be proved. This include knowledge like activity B could only be executed right after A or non-functional properties. Similar, a workflow could have global and local constraints [Arpinar99] that have to hold. Those constraints extend pre- and post conditions by referring to different workflow instances to achieve a global maximum value. This refers for instance to workflows that implement storehouse and logistic where other concurrent workflow instances require material that has to be made available just in time to minimize costs and optimize performance. Such optimizations could only be done based on an unambiguous, formal definition. However, the most important issue for PESOA is the correctness of workflows regarding to variability concepts and mechanisms.

Of course, the reasoning capabilities depend on the formalization chosen (e.g. set-theory, graph-grammars, Petri nets, process algebra, ...). A programming language can be seen as a kind of formalization as it is usually based on an EBNF-grammar. However, this grammar only covers the syntax; the semantics is often unusable for reasoning other then creating traces of execution.

Other issues that might be reasons to formalize workflows are different ontologies and distributed workflow systems. Their usefulness for the PESOA has still to be evaluated; especially as those topics are quite complex of their own. Nevertheless, with formalization, they are explicitly permitted in later steps or extensions. The first issue is the automated adoption to different ontologies. As shown in the introduction example, there are several companies that conduct business. Each of those companies has an explicit or implicit ontology that describes the concepts used in the context or domain of the business. Those ontologies might be similar if the businesses are residing in the same domain or up to totally different in other cases. As usually competitors don’t do businesses which each other, there are often different ontologies that must be matched. When the different business processes are represented as different workflows in different companies, automatic matching of the syntax and semantics to make them work together might be wanted. While a human might be able to match semi-informal ontologies, a computer does not – he needs a unambiguous, formal definition [Schlenoff99, PSL]. The next issue covers large scale workflow systems. Those systems require the distributed execution of workflows on

several computers. This includes the off-line execution of worklist items as for example an agent processes some activities on client side using his laptop. To be able to split workflows onto different machines and make them available on and offline requires an unambiguous, formal definition. The key issues include synchronization aspects which would be impossible if the semantic of a workflow is imprecise [Alonso97, Bauer01].

The disadvantages of formalization include the complexity and incomprehensibility by non-expert users. Both problems can be solved by the usage of tools that are able to formalize graphical workflow notations and advice the user in creating, correcting and optimizing their workflows. Additionally, formalism would give an academic strength and correctness to the variant workflow definitions.

3.4 Summary

This chapter introduced the core requirements of the e-business domain for the PESOA-project. Business processes in the context of e-business are called workflows as they represent the computer executable parts of the business process. There is a distinction between intra- and inter-organizational workflows, where intra-organizational workflows represent single workflows within a company and inter-organizational workflows describe the interaction between different companies. The different workflow aspects such as functional, behavioural, organizational, operational, and informational have been introduced shortly. An aspect missing that has to be researched in the scope of the PESOA-project is variability for workflows. The behavioural aspect is the most important one and has been researched most. A result thereof is a list of commonly used patterns, called Workflow Patterns. Those patterns can be typically found in intra-organizational workflows. Another kind of pattern that is important are communication patterns which describe communication in inter-organizational workflows. In the last section it has been stated why the PESOA-project requires unambiguous workflow definitions as the scope is automated code generation based on product lines.

The PESOA-project requires the definition of variant workflow models in the e-business domain. Variant workflow models are the equivalence to variant process models in the broader PESOA domain. This includes intra- as well as inter-organizational workflows, the ease of configuration and changeability and the application to software product lines with the goal of generating executable workflow definitions.

The follow up research requires the development of concepts, methods and techniques to create, verify, and optimize variant workflow models that are

built of small workflow assets. Therefore, a generic framework that supports three key issues is needed. The main goal is a graphical notation for the design of variant workflow models which is backed by a formalization to reason about workflows and allow the third issue, the automated code generation, respective the generation of executable workflow descriptions. The project description requires the adoption of existing solutions to the PESOA-project. They will be discussed in another chapter. The notations and languages should at least support the behavioural workflow aspect for intra-organizational workflow as well as communication for inter-organizational workflows. They should be extendible to support the idea of variant workflows models.

4 Characteristics of Software-based Automotive Processes

Today, the functionality incorporated into the automobile is being implemented using electronic systems to a greater and greater degree. More than 80% of all future innovations are expected to be carried out in the field of electronic systems alone [Leen02]. The developments in this area are, in particular, driven by the demand for low fuel consumption and reduced emissions as well as the desire for enhanced safety and increased comfort.

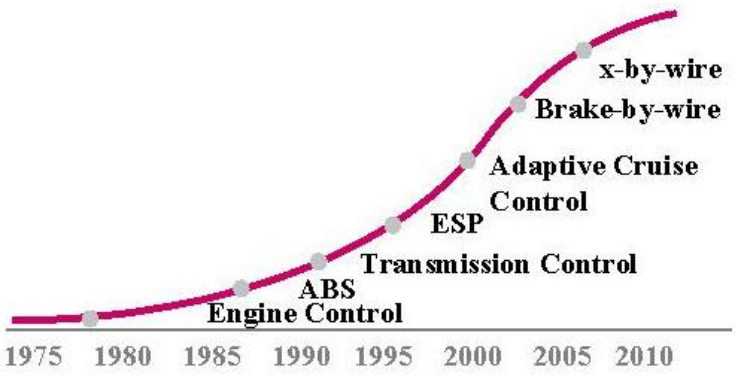


Figure 16: The Development of Electronic Systems in the Automobile.

Already we find that numerous functions in the car are monitored and controlled electronically. Compared to their mechanical counterparts, the benefits of electronic systems lie in their higher reliability (regarding wear), lighter weight, more compact packaging, and lower costs. And certain features such as the electronic stability program ESP could not be implemented without the integration of electronic systems.

This report sets out the characteristics of the electronic processes in the automobile. The focus lies on those processes that are based on software. Section 4.1 describes the characteristics of electronic systems in the vehicle, thereby presenting the context in which the processes considered here run. Finally, Section 4.2 addresses the characteristics of software-based processes in the automobile.

4.1 Characteristics of Electronic Systems in the Automobile

From a technical view, the electronic systems in the car are comprised of electronic control units, sensors (including directors such as the gas or brake pedal), and actuators. Figure 17 provides an overview of the sensors and actuators deployed in the control of a gasoline-powered internal combustion engine. The sensors and actuators themselves are frequently also systems with electronic, hydraulic, pneumatic and/or mechanical components.

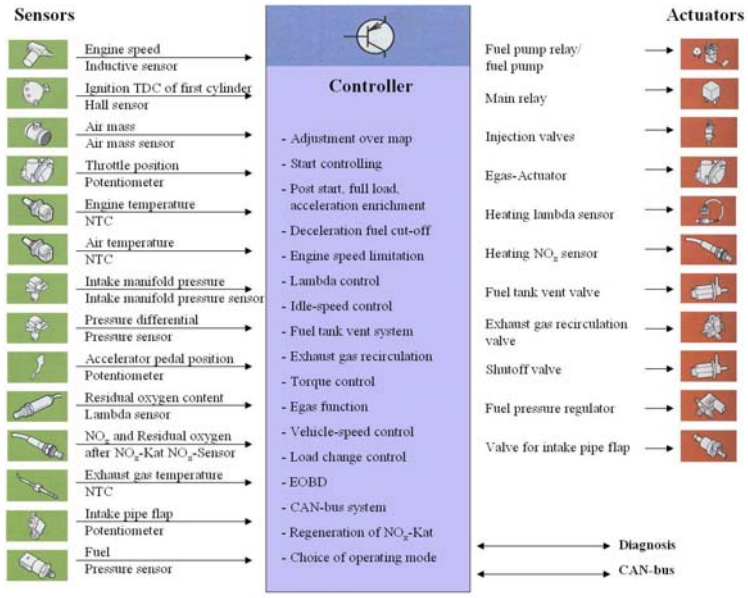


Figure 17: Input and Output Data of the Engine Control Unit for Direct Fuel Injection [EL03].

The electronic control units are the core components of an electronic system. They implement the functional logic: the sensor data received are processed in the control unit and control signals are then sent to the actuators as a result of the calculation. The control units themselves are comprised of microprocessors, program/data memory, and input and output units with the related software routines. Bus systems link the control units to form a complex network. More than 100 control units are typically found in the premium cars on the market today, and all these devices interoperate in a variety of ways.

Typical for the control unit is its hardware-oriented deployment in a technical environment. The vehicle operator and passengers merely have an indirect influence – if any at all – on the functionality of the control units. What is processed are primarily signals and physical variables. In our context, we thus also speak of embedded systems in connection with control units: the software running on a control unit is called an embedded system. In general, embedded systems can be described as dedicated hardware-software systems that take on specialized functions within a defined (and, in most cases, technical) context.

The control unit resources, for example, power rating, memory, communication, are typically limited, being tailored to their particular tasks. As cars are produced in large volumes – for example, over 440,000 Mercedes-Benz C-Class vehicles were manufactured in the year 2003 alone [DC03] – resource restriction and effort optimization in control units play key roles, especially with a view to economy.

4.2 Characteristics of Software-based Control Processes in the Automobile

Due to the great degrees of freedom in their deployment, control functions are increasingly implemented using software. Depending on the vehicle type, more than 80% of all vehicle functions built into cars today are supported by software.

Where, in the past, the mechanical connection between the gas pedal and the throttle valve was effected by means of a cable, so that pressing or letting up on the gas pedal had a direct impact on the opening or closing of the valve, today the signals from the gas pedal are captured electronically and transmitted to the engine control (e-gas). In the engine control, the signals are used together with the current state of operation of the engine to control the throttle valve drive. Since other influencing factors such as the exhaust gas recirculation and engine speed are also taken into account in the computation of the throttle valve opening or closing, the position of the gas pedal now has only an indirect impact on the position of the throttle valve (for example, when the maximum revs are reached, pressing on the gas pedal will have no influence at all on the opening of the throttle valve).

Additionally, using software to realize engine control enables more accurate, fine-tuned control of the engine. And this facilitates observance of the requirements mandated by environmental regulations.

Based on the background of the characteristics of electronic systems in the automobile as described above, the following section examines the key

attributes of and demands placed on software-based processes. In the given context, processes describe the possible consequences of the execution of the related control functions. They are linked to the performance of technical tasks, are executed in an event- or time-oriented fashion, and have a defined end, for example, a control that has been executed, a tuned value within a desired accuracy or a time limit.

4.2.1 Control Functions

Many functions in the car are of an open loop, that is, feedforward, or closed loop, that is, feedback, control nature. depicts a simplified model for control-oriented vehicle functions that illustrates how the torque is determined. Including gearbox and chassis in determining the torque has an impact on the control as well as on the controlled system. This enables more complex control-oriented functions. For reasons of simplicity, we will use the generic term control functions to encompass both feedforward and feedback control.

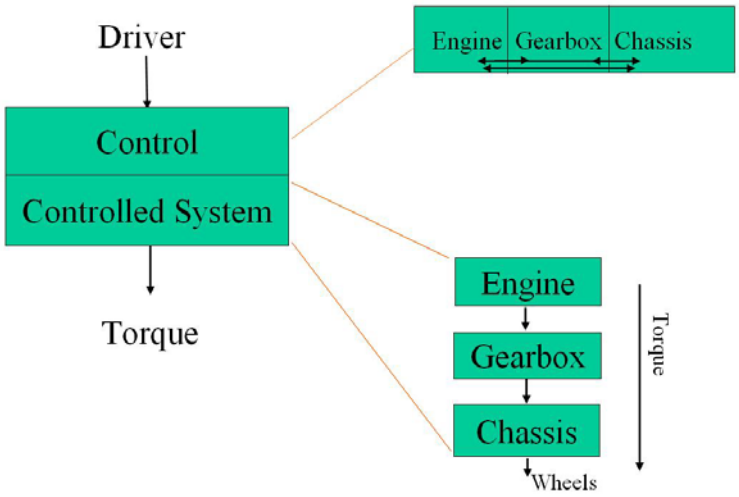


Figure 18: Example Control-Oriented Vehicle Function.

In feedback control, the process variable to be controlled, for example, the torque, is continuously measured or calculated from other measured values and compared to a defined reference value – in this case, the target torque. Depending on the results of this comparison, the variable to be controlled is adapted to the reference variable. In contrast to feedback, or closed loop, control, the so-called open loop control only feeds forward.

In general, control engineering speaks of technical processes in this context. According to DIN 66201, technical processes are characterized by the capturing and processing of (physical) state variables using technical equipment. Examples for physical state variables are the temperature, pressure, position. State variables unambiguously designate the current state of the process. They are captured by sensors, for example, temperature sensors, angle encoders, and influenced by the actuators such as valves and relays.

An example for a feedback control process in the powertrain system is the conversion of energy from the combustion of the gas-air mixture into kinetic energy under consideration of driver information. This process encompasses a variety of further (sub-)processes such as control of the throttle valve, fuel injection, ignition, exhaust gas after-treatment, all of which are themselves interconnected.

A key characteristic of physical variables is that they occur continuously: the physical variable is as a rule present in the control unit as an input signal in the form of a voltage. The value of the physical variable is determined by the height of the voltage being applied. In addition to the physical variables, digital information that can be assigned to certain events is also processed, for example, messages on the basis of the Controller Area Network (CAN).

When using software to realize control functions, the physical variables are mapped to function variables in the control unit. Refreshing of the function variables is dependent on the sample rate. This parameter defines the cycle time for measurement of the signals at the sensor and refreshing of the function variables using the values of the respective state variables.

The sample rate is contingent on the system to be controlled and is a key design parameter for engineering of the functions. The behavior of the control function in the control unit is strongly dependent on the sample rate selected. As a rule, a control unit processes several control functions, which are assigned different sample rates.

Principally, control functions have the following tasks:

- Capture and analyze the physical state variables associated with the technical process such as engine temperature, crankshaft angle, throttle valve opening, and knock signal.
- Influence the technical process by, for example, computing and adjusting the torque.
- Coordinate the process flow, for example, by coordinating the adjustment of the throttle valve opening, fuel injection, ignition, exhaust gas recirculation.
- Monitor the process flow.

4.2.2 Real-Time Requirements

Often real-time demands are placed on the execution of control functions in the vehicle. Strict observance of these requirements is paramount for many of the vehicle functions such as the ignition or fuel injection.

In real-time systems, the execution of software functions has to keep pace with the processes running in the system. For example, the engine control has to consistently ensure timely calculation of the amount of fuel to be injected and the time of ignition for each engine speed and each cylinder. A key demand set for real-time systems is thus timeliness. It is not the speediness that is decisive, it is the reaction within defined and predictable time limits. The correctness of the system does not hinge solely on the result of the calculations but also on when this result is produced. Due to the parallel processing of software (sub-)functions, both concurrency and synchronization are additional vital requirements for real-time systems.

Real-time demands with respect to software functions that are executed on a control unit include the observance of both the defined time intervals when processing signals and the maximum permissible latency time, for example, for reacting to outside events or signal changes. Figure 19 shows the levels of urgency when processing signals from sensors and to actuators in the area of engine control.

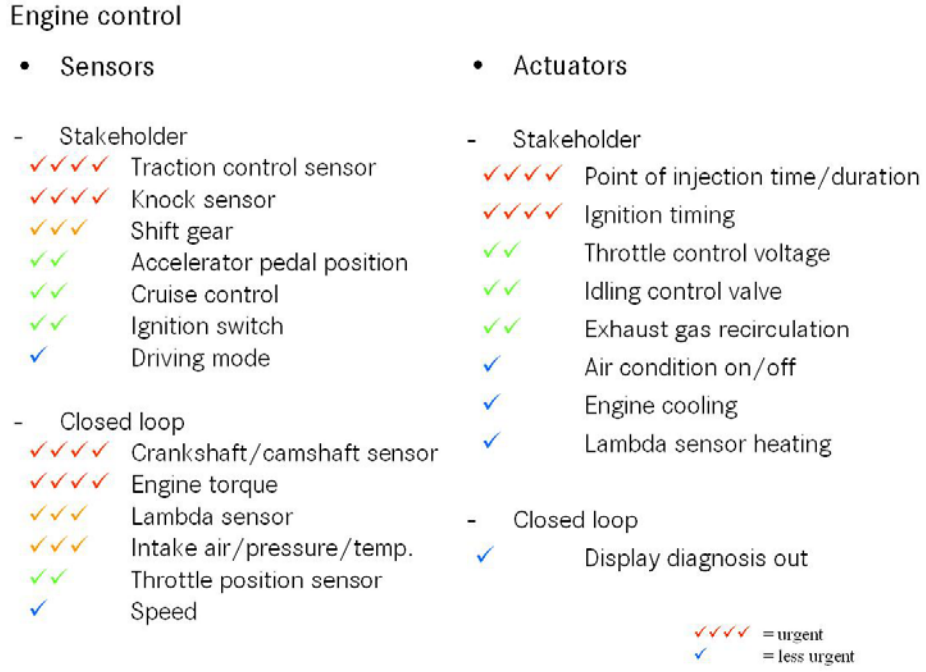


Figure 19: Prioritization Relevant for Signal Processing of the Engine Control.

According to [Schäuffele03], the sample rate is the underlying foundation for the definition of real-time requirements for software functions. If signals are exchanged between the functions of different control units, the sample rate yields the real-time requirements for the data transmission, that is, the periodic intervals in which messages are to be exchanged between the control units.

If several software functions with different sample rates are to be implemented using a single control unit or a control unit network, activation of the software functions is executed in different tasks. A task is a job unit that can potentially or actually be executed in parallel on a processor or a network of processors. A task encompasses a sequence of (sub-)functions with identical real-time requirements that are executed in a statically defined order. When setting up the specifications for a real-time system, software functions may need to be distributed to tasks, if necessary by splitting them up into sub-functions [Labrosse02].

Moreover, when a real-time operating system is to be used, the real-time requirements set for the tasks will need to be considered when configuring the operating system, for example, the number and priority of the tasks (also see [OSEK]).

The processing of incoming events or internal interrupts may lead to varying execution times and processing sequences for the software functions. Here, methods for schedulability analyses can be used to verify whether meeting the real-time requirements is feasible. In such analyses, the number of tasks and the specific basic conditions are examined to determine whether the function is executable. A given number of tasks is executable if each task keeps the deadline defined for its execution, that is, the point in time by which the task is to be fully completed at the latest.

4.2.3 Distribution and Networking

In modern vehicles, numerous control units are installed, most of which come with a steadily growing number of software functions. The functions exchange signals with each other in many different ways. Only when the software functions are networked can value-added functions such as the electronic stability program (ESP), anti-slip control (ASR) or adaptive cruise control (ACC) be realized.

From the technical view, the distribution of software functions to the control unit network and the communication between the individual control units are of particular importance. The distribution of the software functions to the control unit network has a decisive influence not only on the network load but also on the quality of the results of the software functions. The objective

is thus to limit the complexity as well as the effort required for communication when distributing software functions over the network. In addition to the basic conditions such as the necessary computation and communication performance or the fixed assignment of sensors and actuators to specific control units, meeting requirements such as real-time, safety, and reliability constraints needs to be considered [Schäuffele03].

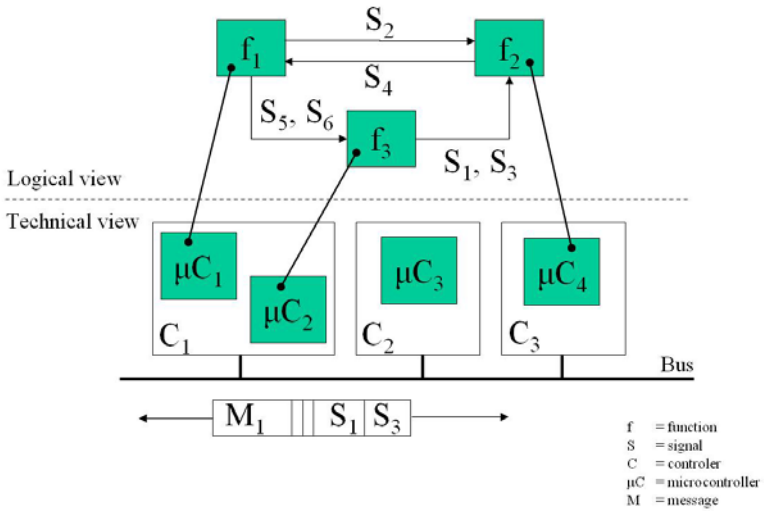


Figure 20: Assignment of Control Functions to Microcontrollers and Control Units.

The communication between software functions takes place between the different tasks on a control unit (inter-tasking) and the information exchange across control units. Information is exchanged both synchronously, through the time-wise coordination of processes, and asynchronously.

For the communication in the control unit network, only the time requirements are of relevance, that is the cycle time prescribed for the signal exchange between the functions of different control units. As a rule, the signals associated with a sending function are processed by several receivers with different time rates.

Two factors have a key impact on the quality of the results in signal transmissions: the value discretization of the signals and the transmission times enabled by the communication system. Moreover, the resolution expected by a function as the receiver of a signal and the range of the signal need to be considered.

The signals are assigned to messages in the communication system, for example, the CAN. Several signals may be associated with a message. A message is the informational unit that is exchanged between different control units. The assignments between senders and receivers of information,

signals and messages, and messages and cycle times for transmissions are done using a communication matrix (K matrix).

4.2.4 Reliability and Safety

A trend in the automobile industry is the rising number of safety-relevant electronic systems in the vehicle. Some well-known examples are the driver assistance systems, which are geared to help the driver in critical situations, and the mechatronics systems in the powertrain (keyword "x-by-wire"). And a prerequisite for the approval of a vehicle model for operation in traffic is proof of the required reliability and safety of such safety-relevant vehicle functions. Hence, corresponding statutory regulations, standards, and laws are defined to take this key aspect into account.

The safety requirements mandate the safe behavior 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 used instead of measured values for the necessary computations (an example is the so-called lambda probe). However, this is generally done to the detriment of other aspects such as environmental requirements. Reliability requirements are reflected in the demand for shorter repair times or longer maintenance intervals. The reliability of the systems may be impacted by failures or disturbances. Measures to enhance reliability are thus aimed at preventing the faults or errors that would lead to a failure or disturbance. Measures implemented to increase safety home in on a limiting risk and target preventing the dangerous effects of faults or errors, failures, and disturbances.

The requirements relevant for the reliability and safety of vehicle functions are defined within the framework of reliability and safety analyses. Failure rate and failure type analyses belong to the key investigations to be performed. Additionally, risk analyses – for example, according to DIN 19250 or IEC 61508 – can be used to derive the safety-technical requirements for the technical realization of the system as these frequently have a great impact on elements of the system architecture such as the hardware, software, sensors, and actuators – e.g. on the assignment of software functions to control units in distributed and networked systems.

The prerequisite for the triggering of safety reactions is the reliable recognition of disturbances and failures. Monitoring functions are essential to ensure both the reliability and safety of electronic systems. They also have a crucial impact on the system architecture. Often a monitoring system for electronic control units is realized through a combination of hardware and

software measures implemented for the monitoring of microcontrollers, directors, sensors, actuators, and the control functions.

4.3 Summary and Conclusions

Control functions in the automobile are more and more often realized using software. Processes describe the potential consequences upon the execution of the related control functions in an electronic, hydraulic, pneumatic and/or mechanical environment. Processes may be made up of numerous sub-processes. In our context, processes are invariably linked to the execution of technical tasks. They are initiated by an event and have a defined end, for example, a control that has been executed, a value that has been tuned to a desired accuracy, or a time restriction.

Many processes in automotive applications are to be considered under the aspect of feedforward and feedback control. Of particular importance is the great number of control-oriented processes in the vehicle, which are themselves interlinked. Oftentimes these processes are subject to real-time requirements. These include both the keeping of time intervals when processing signals and the observance of the maximum permissible latency time, for example, when reacting to external events or signals.

The number of control units and control functions integrated in the car is increasing steadily. And it is only through the networking of control functions that we gain networked systems, thus yielding value-added functions. Yet, the networking of control functions also leads to a growing information exchange between the processes involved. Thus, the distribution of software functions to the control unit network and the configuration of the communication system have a decisive influence on the network load and the quality of the results.

The increasing number of safety-relevant systems in the vehicle requires the transfer of safety-technical requirements to the system design. Such requirements greatly affect the system architecture, that is the hardware, software, sensors, and actuators, thus also exerting an influence on the execution of the processes in the automobile.

5 Languages for Process Modeling

One of the challenges of the PESOA project consists in finding a graphical process modeling language suitable for being used in the context of process family engineering. Existing process modeling languages may only be a starting point since in order to be used in PESOA they have to be enhanced by process family engineering specific concepts and notations. However, in consideration of the huge number of existing process modeling languages it wouldn't make sense to start from the scratch defining an own modeling notation. Instead it makes sense to consider existing approaches. In order to be appropriate for acting as a foundation for a modeling language meeting the PESOA needs the selected modeling approach in particular has to able to properly represent the processes in the PESOA domains of application, i.e. the e-business and automotive domain, which are characterized by different requirements. Therefore this chapter will deal with the analysis of existing process modeling languages concerning their suitability for being used as a starting point for a PESOA process modeling language for e-business and automotive processes. This examination will close with the sketch of a common PESOA metamodel comprising the modeling elements required in the PESOA project.

5.1 Modeling Automotive Processes

This section will examine the suitability of UML Activity Diagrams and UML State Machines for modeling automotive processes. These techniques have already been shown promising modeling the concrete automotive processes provided by DaimlerChrysler Research. In order to analyze the suitability of UML Activity Diagrams and State Machines the following subsection will outline the concepts a process modeling language must be able to represent in order to be suitable for describing automotive processes. The requirements for process modeling languages summarized here have been identified by the project partner DaimlerChrysler on the basis of their experience with automotive electronics. The subsection after the next will show that a combination of UML Activity Diagram and State Machine elements is capable of depicting the relevant aspects of the automotive processes considered in the PESOA project by analyzing them on the basis of the requirement list for automotive processes. In addition to the mere graphical representation there is a need for a formalization of the applied modeling concepts. An approach for dealing with this requirement is sketched as well.

5.1.1 Requirements for the Automotive Domain

This subsection briefly summarizes the relevant requirements of automotive processes.

A process modeling language suitable for representing automotive processes must be able to describe the flow of control as well as the flow of data between activities. Concerning the control flow especially routing constructs for modeling iterations are required since in automotive processes control cycles play an important role. Routing decisions may depend on variable values, which is one reason for the required support of variables. Since the electronics of a car is characterized by having distributed systems interacting with each other a process modeling language for automotive processes should provide some means for representing asynchronous as well as synchronous communication. Also the parts of the system involved in the communication should be identifiable. Just like in processes of other domains of application also in automotive processes the occurrence of an exception may lead to a deviation from the normal flow of control in case of an error. Concerning the data flow, modeling elements for representing data sources and data sinks are required as well as means for describing the corresponding input and output data. Data sources and sinks may also comprise data storages for the persistent storage of data. In automotive processes the execution of a subprocess often depends on the state of the system. The state change of the system and the execution of an activity are often triggered by the receipt of an event. Automotive processes are characterized by being subject to time requirements, which therefore somehow have to be part of the process model as well.

In addition to these functional requirements, which determine the modeling elements a suitable process modeling language has to provide, a modeling language appropriate for being used in Process Family Engineering has to have a number of additional properties that go beyond the pure graphical representation of the considered processes. First of all since one goal of the PESOA project is the generation of an implementation on the basis of a process definition, the utilized process modeling language has to have an unambiguous semantics. Else program code cannot be generated properly on the basis of the process description. An unambiguous semantics is also required for simulating the described process. Secondly qualitative as well as quantitative analysis of an automotive process should be possible for ensuring the accuracy and quality of the processes, which is crucial in the context of automotive electronics. Qualitative analysis comprises investigations of the correctness of a process like the absence of deadlocks, livelocks and dead nodes [Van der Aalst02c] while quantitative analysis here means particularly performance analysis. In order to be able to fulfill these additional requirements we need a mathematically sound and unambiguous definition of the process modeling languages used for modeling automotive processes. Since this typically doesn’t apply to graphical modeling

languages an additional mathematically precise representation is required. This shall be achieved by formalizing the process modeling language subsets required for modeling automotive processes.

5.1.2 Languages Supporting Automotive Processes

This subsection will analyze the process modeling techniques UML Activity Diagrams and UML State Machines regarding their support of the requirements for automotive processes as outlined in the last subsection. More information about Activity Diagrams can be found in [UML, Born04, Pender03, Schnieders04, Bock03a, Bock03b, Bock03c, Bock04a, Bock04b], while State Machines are described in [UML, Born04, Pender03].

Activity Diagrams. The UML 2.0 specification defines so called compliance points, which are parts of the UML specification and on the basis of which it can be determined, to what degree a vendor of UML 2.0 tools supports the UML specification. Depending on which compliance points he implements the standard, his tool can reach for example the compliance levels “basic”, “intermediate”, or “complete”. Basic Activities only comprise the most essential modeling elements, while Intermediate Activities offer the modeling elements needed in probably most applications. Complete Activities also offer some advanced modeling constructs. Additionally there is a structured level that allows for the modeling of traditional structured programming constructs and an extra structured level that also supports exception handling.

Concerning the control flow requirements for automotive processes, Activity Diagrams differentiate between control flows and data flows allowing for representing both types of flows explicitly. 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. In Basis Activities iterations can be realized using a combination of reentering arcs and decision and merge nodes. Alternatively with LoopNodes and ExpansionRegions StructuredActivities offer elements dedicated solely for modeling iterative behavior.

Concerning the data flow requirements Activity Diagrams offer Pins for modeling the input and output data of Actions and ActivityParameterNodes for the input and output data of Activities. Additionally CompleteActivites provide ParameterSets 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 CentralBufferNodes in IntermediateActivities and DataStoreNodes in CompleteActivities are

possible data sinks and sources in Activity Diagrams. A DataStoreNode is a modeling element for indicating the persistent storage of data.

Events in Activity Diagrams can be raised using SendSignalActions. An AcceptEventAction on the other hand is a type of Action that is activitated as soon as an event of the indicated type occurs. An AcceptEventAction can be triggered for example by a signal sent by a SendSignalAction or by a time event. In Activity Diagrams Exceptions can be used to indicate deviations from the normal flow of control. Exceptions can be handled locally using ExceptionHandlers (ExtraStructuredActivities) or they can be forwarded via certain output parameters indicated by the isException attribute (CompleteActivities).

Variables in Activity Diagrams are used for example in Guards, in decisionInputBehaviors for making routing decisions, in selectionBehaviors (in CompleteActivities) for reading data from a DataStoreNode, in the description of localPreconditions or localPostconditions of Actions (in CompleteActivities), or in the form of parameter names of Pins or ActivityParameterNodes.

Activity Diagrams allow for the modeling of asynchronous and synchronous communication between two processes in different Activities as well as between two subprocesses within the same Activity by means of ActivityEdges, SendSignalActions, and AcceptEventActions. Figure 10 and

Figure 22 give an example of how to model synchronous communication between two subprocesses located in the same and in two different Activities. Modeling an asynchronous communication could be modeled even easier by just omitting the transmission and receipt of a reply message.

Figure 21: **Synchronous communication between two subprocesses within the same Activity**

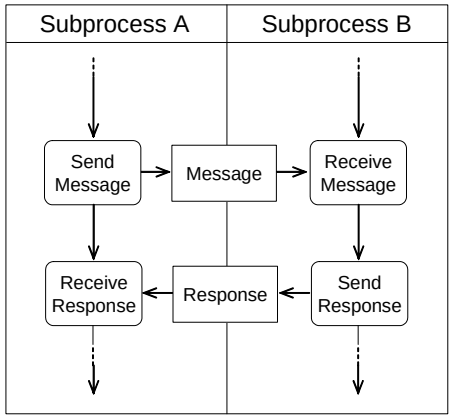
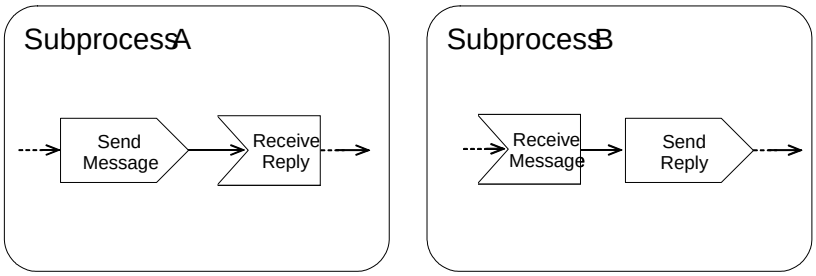


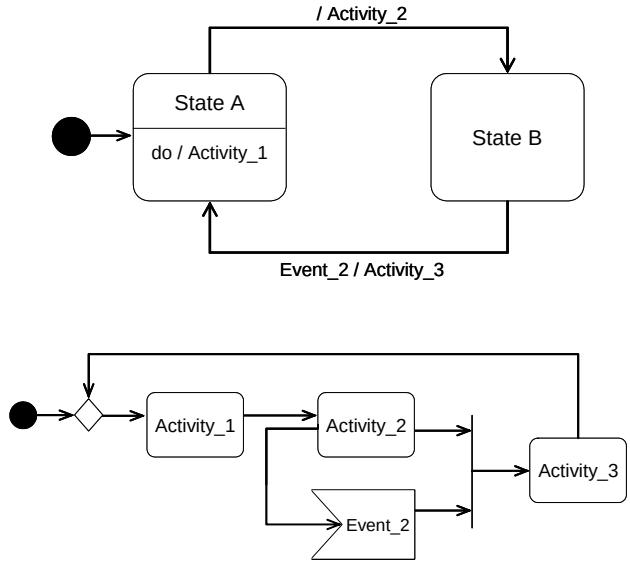
Figure 22: **Synchronous communication between two processes within different Activity**



Additional information about the context in which a process is executed can be provided using ActivityPartitions (IntermediateActivities) or UML Comments. ActivityPartitions are typically used in Activity Diagrams to assign domain-specific information to nodes and edges like for example the person or class by which certain parts the process are executed. Therefore a swimlane notation can be used. Alternatively a partition name can be placed in parentheses above an Activity name. Moreover UML Comments are a form of adding freeform text to any element in a model.

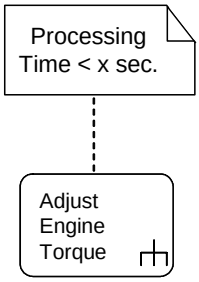
Concerning the representation of system states Activity Diagrams don't provide elements dedicated especially to this purpose. Even though it is possible to transform a State Machine into an Activity Diagram a one-to-one mapping of State Machine states to Activity Diagram elements is not possible. Only State Machine states representing a behavior can be mapped directly to an Activity Diagram Action while other types of states will only be implicitly present in the resulting Activity Diagram. Figure 23 gives an example for a State Machine and the corresponding Activity Diagram representation. While "State A", which is characterized by the execution of "Activity_1", is represented adequately by "Activity_1" in the corresponding Activity Diagram, "State B" isn't mapped on a concrete element and the information about the existence of the state gets lost.

Figure 23: **A State Machine and the corresponding Activity Diagram**



Except for AcceptTimeEventActions Activity Diagrams don't provide means particularly dedicated for adding time data to an Activity Diagram. Especially there are no means for indicating the time it takes an executable node to carry out a computation. A possible way for providing this information would consist in the employment of UML Comments as shown in Figure 24.

Figure 24: **Adding time constraint to Activity Diagrams using UML Comments**



The UML Activity Diagram specification doesn't contain a mathematically sound definition of the Activity Diagram semantics, but only a textual semantics description is provided which is scattered over the UML specification due the dependence of other parts of the UML specification and cannot be considered to be precise. Moreover parts of the Activity Diagram semantics description even seem to be ambiguous.

State Machines. Concerning the modeling of flows 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. 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 guardless transition. Additionally Activities may be assigned to a transition. These Activities are executed when the transition fires. The Activities integrated into a State Machine correspond exactly to the Activities as specified in the Activity Diagram metamodel. 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.

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 reentering transitions.

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. 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.

Just like in Activity Diagrams synchronous as well as asynchronous communication in State Machines can be realized using integrated Activities containing respective SendSignalActions. The receiving State Machine may then react to the signal by means of a SignalTrigger.

Context information of a State Machine can be provided using UML Comments. In contrast to Activity Diagrams State Machines don't dispose of a special notation for providing context information like ActivitiyPartitions in Activity Diagrams.

As the name suggests states are another core concept of State Machines. In state machines a state describes the condition of an object, which can be described by the object's attributes or by means of a behavior the object is processing.

Just like Activity Diagrams State Machines don't provide any means for displaying time data either. Again one possibility would be to assign UML Comments with time information to the respective Activities or states.

Like Activity Diagrams, also State Machines have a textual semantics description instead of a mathematically precise definition.

Figure 25 gives a summary of the requirements for a process modeling language suitable for representing automotive processes and their support by Activity Diagrams and State Machines.

Figure 25: Graphical process modeling languages and their support of automotive process aspects

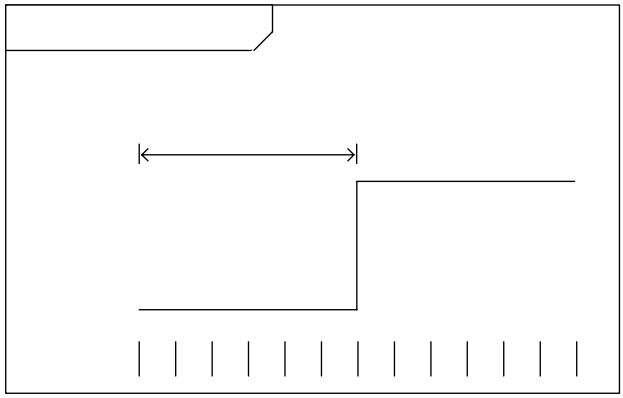
	Activities	Control Flow			Data Flow				Events		Utilization of Variables	Synchronous, asynchronous Communication	Context Information	System States	Time Data
		Expl. Representable	Routing		Expl. Representable	Input/Output Data	Data Sinks and Sources	Persistent Storage of Data	expressible	Exceptions					
			Basic Constructs	Iterations											
Activity Diagrams	X	X	X	X	X	X	X	X	X	X	X	X	X		(X)
State Machines	X	X	X	X		X	X		X		X	(X)	(X)	X	(X)

Representation of Time. Since in UML Activity Diagrams and State Machines there are no specific elements provided for representing time durations and time constraints further approaches for properly supplementing them with the missing time aspects will be considered. The techniques for expressing time requirements that will be investigated for being used for automotive processes in PESOA are UML Timing Diagrams [Pender03] and UML-RT and the UML Profile for Schedulability, Performance and Time. UML Timing Diagrams will be covered in this report while UML-RT and the UML Profile for Schedulability, Performance and Time will be addressed later in the context of further investigations.

UML Timing Diagrams provide means for assigning time information to the states of an object and the corresponding events responsible for the state change. In UML Timing Diagrams time constraints can be depicted as well as duration constraints, i.e. the date can be displayed in which a state has to be entered and left as well as the time the object will stay in a certain state. Some of the main features of a Timing Diagram shall be illustrated based on the exemplary Timing Diagram in Figure 26.

The diagram in Figure 26 depicts a part of a motor control unit process showing the transition of the motor from the state “Afterstart” to the state “Running”. In Timing Diagrams states are stacked on the left margin of the frame that contains the process that shall be depicted. The state timeline of the object shows the set of valid states and time. The flow of time goes from left to right. Changes in the level of the line represent changes from one state to another, which are induced by the receipt of messages. The name of the message is assigned to the lifeline at the point where the timeline changes its level due to the receipt of the message. In the example the motor state changes from the state “Afterstart” to the state “Running” upon the receipt of the message “Time for Afterstart expired”. Now this example contains some time duration constraints for the state “Afterstart”. It shows that the time the motor stays in the state “Afterstart” is limited to x milliseconds.

Figure 26: **Example for the application of Timing Diagrams**



5.1.3 Summary

The analysis has shown that Activity Diagrams meet almost all of the requirements arising from the employment with processes in the automotive domain. Activity Diagrams only lack of elements for properly modeling system states and time data. Taking a closer look at the requirements and the Activity Diagram specification it even turns out that already an Activity Diagram subset would provide the required modeling elements. With respect

to the Activity Diagram compliance levels IntermediateActivities could be taken as a fundament. In order to be able to model the persistent storage of data, DataStoreNodes (and selection behaviors) would have to be added as well as ExceptionHandlers and isException Pins for modeling exceptions and exception handling. Additionally ParameterSets for modeling alternative sets of input and output parameters could be added if necessary. This approach would set aside LoopNodes and ExpansionRegions, which don't increase the expressiveness of an Activity Diagram but are only supposed to enhance the structuring of a diagram.

State Machines on their part don't offer the modeling elements needed for depicting automotive processes properly by themselves, but have to be combined with Activity Diagrams. Since there are many ways for integrating Activity Diagrams with State Machines this can be done smoothly. In contrast to Activity Diagrams State Machines offer a variety of means for modeling states. Additionally some basic routing constructs provided by the State Machine pseudo states would be needed in order to properly connect the states.

Concerning the representation of time data, which is not supported explicitly by Activity Diagrams nor by State Machines, theoretically UML Comments could be used to add this information to the process diagrams. The drawback of this approach is that UML comments can potentially contain any information and therefore using them for displaying time data wouldn't lead to a very clear and descriptive representation of time requirements. Instead Timing Diagrams can be used for depicting time constraints as well as duration constraints for object states and the receipt of messages. Additionally the usability of UML-RT and the UML Profile for Schedulability, Performance and Time can be used for adding time data to the PESOA process diagrams, which can be addressed later in the context of further investigations if needed.

Regarding the requirement for a mathematically sound and unambiguous formalization of the automotive process diagrams the needed Activity Diagram elements could be formalized best using Colored Petri Nets [Jensen92, ISO02]. Reasons for this are the similarity of the Activity Diagram and Petri Net semantics, since both are token based. Moreover Petri Nets are a good choice since they are very popular and therefore familiar to most computer scientists. Due to the popularity of Petri Nets there are already various tools and algorithms available for the quantitative as well as qualitative analysis of processes represented as Colored Petri Nets. A Petri Net representation of the required Activity Diagram elements would make these tools and algorithms available to the PESOA project. An evaluation of dozens of tools for simulating and analyzing Petri Nets can be found for example under [Störrle98].

To sum up these considerations a combination of State Machines elements with IntermediateActivities expanded by a small number of additional Activity

Diagram elements would be suitable for graphically representing automotive processes. Additionally UML Timing Diagrams, UML-RT and the UML Profile for Schedulability, Performance and Time could be used for supplementing the process diagrams by the missing time data. Concerning the formalization of the elements involved in the modeling of the automotive processes, the formalization of IntermediateActivities by means of Colored Petri Nets is already in process. For the mapping of Activity Diagram Exceptions on Petri Nets there exists already an investigation [Störrle04], which could be taken as starting basis, while the formalization of Activity Diagram isException Pins, DataStoreNodes, selectionBehaviors, and the required State Machine elements still has to be investigated.

5.2 Modeling Processes in E-Business

This section summarizes the requirements for the E-Business domain from chapter 3 and discusses the Business Process Modeling Notation (BPMN) [BPMN] as a well suited approach to fulfill them. In addition to the graphical representation, there is a need for a formalization to achieve the goals of the PESOA project which is discussed thereafter.

5.2.1 Requirements for the E-Business Domain

As stated in chapter 3, the requirements for the E-Business domain can be divided into three main areas. The most important area is a graphical notation for workflow design which will be extended during the project to support variant workflow models. The graphical notation must fulfill several specific requirements which will be summarized in the next paragraphs. The second area includes a formalization of the graphical notation which allows reasoning and builds a formal foundation for the variability concepts. As the requirements for the formalization depend on the concepts of variability, which still have to be developed, they have to be investigated on demand. The last area covers target languages for workflow execution. As those requirements depend on the e-business partner, they must be investigated later on.

The graphical notation that will be used for the visualization and editing of variant workflow models is one of the key components of the PESOA project. An existing notation that will be adapted for the E-Business domain should be extendible to support variant workflows and already support the most important workflow aspects. The relevant workflow aspects include the definition of activities or tasks (functional aspect), the routing between the tasks (behavioral aspect), roles (organizational aspect), and the required case data (informational aspect). Another requirement is the support for intra- and inter-organizational workflows. Intra-organizational workflows typically contain behavior collected as Workflow patterns [van der Aalst00, van der Aalst03], whereas inter-organizational workflows rely on

communication. To support both types of workflow, the notation should provide sequence as well as message flow capabilities.

To support and enable the goal of the PESOA project in the E-Business domain, that is the automated process generation by the use of product line techniques, some additional properties are required. These properties include the unambiguousness of the notation regarding syntax and semantic as well as advanced reasoning capabilities to ensure the variability mechanisms.

5.2.2 Recommended Notation

This subsection proposes the Business Process Modeling Notation (BPMN) as appropriate to fulfill the requirements of the E-Business domain mentioned above. The advantages and drawbacks regarding the requirements will be discussed and analyzed. An approach to overcome the major drawbacks by the use of formalization is discussed thereafter.

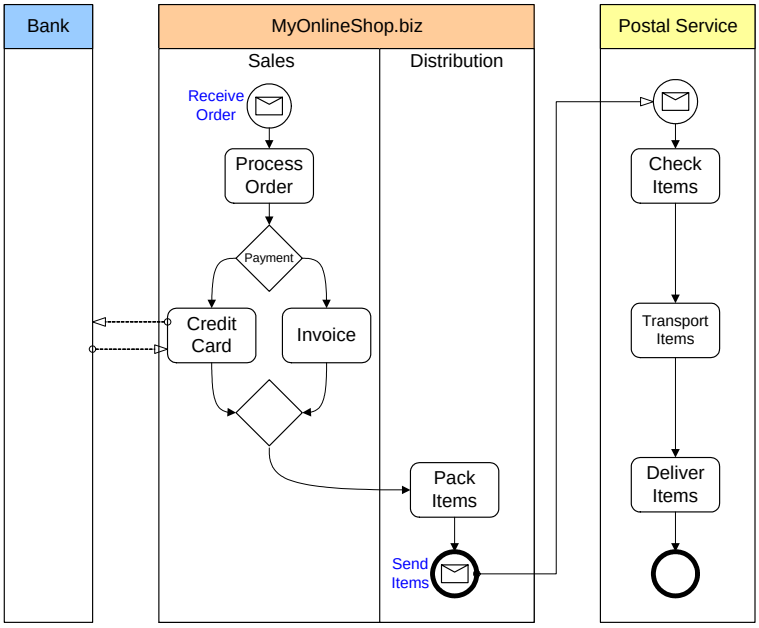
Business Process Modeling Notation. The Business Process Modeling Notation has been published by the Business Process Management Initiative (BPMI) [BPMI] and was introduced in an earlier PESOA report [Schnieders04]. The BPMN has been “proposed” as the future standard for business process modeling by a number of companies related to the BPMI. Among those advocators are global players like Bea, IDS-Scheer, Peoplesoft, SAP, and IBM as a leading force along with many other companies related to business process management. The BPMN is quite similar to a subset of the UML Activity Diagrams which will be used for the automotive domain. However, the BPMN does not complain to any of the predefined compliance levels of the UML Activity Diagram specification. Instead it focuses on the demands of business processes, which allows a light-weighted notation that can possible be mapped to UML Activity Diagrams if needed. Besides defining a meaningful set of modeling elements for business processes, the notation is much more business than technical related. This allows stakeholders from the business domain to specify and analyze business processes more easily. Since the BPMI does not see itself as a standardizing organization, there are ongoing efforts in integrating the BPMN as another view in the UML [White04].

Technically, the BPMN consists of a few, but yet powerful modeling elements that can be enhanced if needed. Those are grouped into four basic categories: flow objects, connecting objects, swimlanes and artifacts. The flow objects are the main graphical objects and consist of Events, Activities, and Gateways. Those are used to model intra-organizational workflows connected by Sequence Flows. Inter-organizational workflows are modeled by the use of different Pools which are connected by Message Flows. A Pool represents a business entity like a company that can be parted into different Swimlanes that resemble organizational units. Elements can have data or

annotations attached by using Association connectors. These are the core modeling elements that can be enhanced in several ways as will be explained later but always keep their general shape and look. This allows for simple recognition and classification of new or enhanced elements.

The mapping of the workflow aspects to the BPMN is as follows. The functional aspect is represented by Activities, which can be composed to Sub-Processes to build hierarchies. The behavioral aspect is represented through Sequence Flow inside a Pool and Message Flow between different Pools. This resembles the concepts of intra- and inter-organizational workflow. Activities or Events can be used to model communication between different Pools. Specialized Events can start, hold or stop the Sequence Flow depending on certain conditions like incoming messages, rules or timers. Gateways are used to decide, split or join the Sequence Flow. The organizational workflow aspect is represented by different Pools and Lanes which resemble different roles. The informational aspect can be modeled by the use of data annotations. Annotations might be a bit to abstract, but as the requirements on data-formats depend on the target environment, it is up to a modeling tool to enforce special types.

Figure 27 **BPMN Example**



An example is shown in Figure 26. It consists of a simple online-shop that is made up of three different Pools. The leftmost Pool, the Bank is represented as a black-box Pool, which mean that the internal processes are unknown to the outside world. The only thing knows is a communication interface which allows for validating credit cards. The Pool in the middle is the online shop, which consists of two different departments, modeled as Swimlanes. The

internal process starts with receiving an order event, processes the order and the payment, pack the items and finally send a message to a postal service to actually ship the items. The postal service is shown in the rightmost Pool. The inter-organizational workflow is modeled by the use of Message Flows, which connect different internal processes of the participants.

One aspect to measure the suitability of a notation for the workflow domain is the support of the so called workflow patterns. Those pattern specify common control structures for intra-organizational workflow [van der Aalst03] which are mostly represented in the BPMN by routing constructs like Sequence Flow, Gateways or Activity Markers. The BPMN has been designed to support all of the currently documented workflow patterns [White03]. The support of communication between different participants is given by the Pool and Message Flow architecture.

As the extendibility of the BPMN was already mentioned, it will be considered of how this can be done. The first approach is a graphical extension which modifies the core modeling elements. For example, the core Event is just a circle that is extended by a message symbol to derive a Message Event that could start a process. A thick lined circle represents an End Event that terminates the Sequence Flow. It can be enhanced with a message symbol to represent the transmission of a message. Another approach extends the attributes that provide properties for each of the modeling elements. Typical attributes of a Pool are for example the process it contains, the name of the participant that it represents as well as the lanes that might be inside. Events have for example a trigger or a time or process reference attribute. Those attribute sets could be extended to fit the requirements of the PESOA project.

A disadvantage of the BPMN is the absence of a formal defined syntax and semantics which is required for the automated code generation as well as a foundation for the variability mechanisms. The variability mechanisms must ensure the consistence and matching of interchangeable process parts which are based on formal models. The disadvantage of an imprecise semantic is shared with the UML Activity Diagrams and other graphical notations.

Process Algebra. Modern process algebra is recommended for formalizing the Business Process Modeling Notation. Process algebras are based on a lot of theoretical foundations for reasoning and are very minimalist at the other hand. Modern process algebras like the pi-calculus are based on mobile processes which change their configuration [Milner92, Sangiorgi03]. This allows an easy representation of intra- as well as inter-organizational workflows [Puhlmann04]. Process algebras have been successfully used to formalize programming notations and protocols for a long time, starting with CSP [Hoare78] and CCS [Milner89]. Examples can be found for instance in

[Mauw96, Bernardo02, Brogi04]. As the BPMN defines a mapping to executable languages which themselves are based on process algebra, like the BPML [BPML] is based on the pi-calculus, formalization should be possible.

Process algebras are a different view of formalization as for example Petri nets. The most important issue to choose process algebra is the existing relation between BPMN and process algebra by executable XML languages like BPML. A beneficial aspect is the quite small but yet precise formalization that a process algebra allows for. Another benefit is the extendibility of the semantic through additional transition rules.

Process algebra can directly formalize the concepts of a language or notation; whereas formalization based on Petri nets map a graphical notation to a formal representation. The later approach is required for reasoning about quantitative as well as qualitative aspects of processes [van der Aalst02c] as in the automotive domain. A formalism based on process algebra defines some kind of formal language that can be easily extended to meet further requirements like pre- and post conditions of process parts in one unified framework. It is possible to consolidate the advantages of Petri nets and process algebra as investigated by for example by [Basten98].

5.2.3 Summary

The analysis of the BPMN, which was shortly summarized in this section, has shown that the notation is well suited to fulfill most of the requirements of the e-business domain. These include the support for intra- as well as inter organizational workflow, the relevant workflow aspects, workflow patterns and communication. By further analyzing the requirements it shows out that the UML Activity Diagrams also fulfill most of them. But as stated in the previous section, the full Activity Diagram specification is way to complex for the automotive requirements, this holds even more for the e-business domain. The BPMN can be seen as a selected subset of the Activity Diagram specification that is specially suited to model business processes. The advantage of having two domain specific languages that share one meta-model will be discussed in the next section.

Unfortunately, the BPMN lacks a precise and formal definition of its syntax and semantics. These are required for the automated code-generation and as a foundation for the variability mechanisms and concepts that will be developed during the PESOA project. An interesting point is the formalization of the meta-model that will contain support for the variability concepts. One approach to create a view of this formalism in the e-business domain is by the use of process algebra. The pi-calculus has been evaluated for this purpose and it showed out that it supports all requirements of the e-business domain, including the capabilities for further enhancements. The analysis will be published in a technical report [Puhlmann04]. However it still

has to be evaluated where and when formalization supports variability mechanisms and concepts.

As the BPMN additionally contains a complete mapping to executable XML languages, like the Business Process Execution Language for Web Service (BPEL4WS) [BEA], this languages could be used as targets for the generator. The integration into software product lines is given by the similarity to the UML meta-model which allows for incorporating existing solution like KobrA [Atkinson02]]. The elements of the BPMN could also be extended graphically to allow the future representation of variant workflow models as well as on attribute level, which will support code generators and configuration tools.

5.3 Conclusions

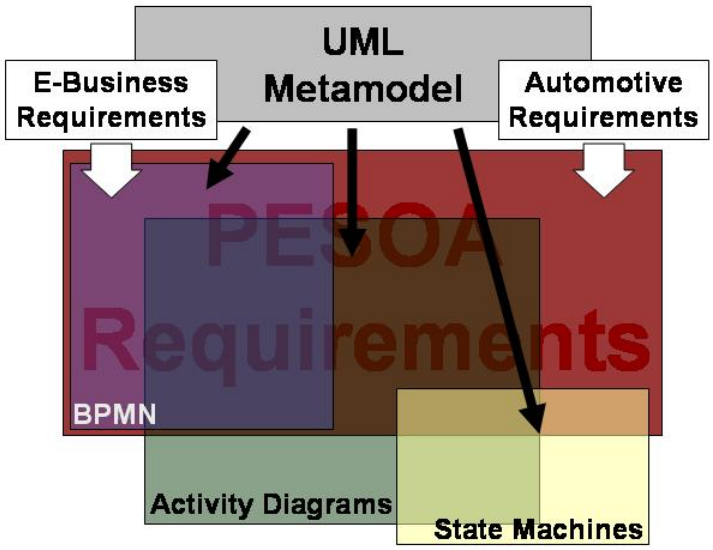
This section will summarize the results of the analysis of process modeling techniques for automotive and e-business processes on the basis of the specific requirements given by the respective domain. Moreover similarities of the modeling techniques suitable for representing e-business and automotive processes that may lead to the identification of a common metamodel for a PESOA process modeling language will be discussed.

Concerning the modeling of automotive processes a combination of State Machines elements with IntermediateActivities expanded by a small number of additional Activity Diagram elements would support all of the requirements given by the automotive domain. However, any of these two modeling approaches taken separately doesn't provide all of the modeling concepts needed for the graphical representation of automotive processes. UML Timing Diagrams, UML-RT and the UML Profile for Schedulability, Performance and Time could be used for supplementing the process diagrams by the missing time data, while the latter two approaches still have to be investigated concerning their applicability for PESOA processes.

The e-business domain includes in the first row the support for intra- and inter-organizational workflow. It is characterized by the workflow aspects denoted as the functional, behavioural, organizational, and informational perspective. The different workflow aspects can be represented using certain concepts like activities, data input or output. Commonly used patterns in the e-business domain employable for describing the behavioral aspect have been collected as Workflow Patterns. The Business Process Modeling Notation supports all of the abovementioned aspects. One aspect that is missing is variability as a key aspect of the PESOA project. The BPMN can be extended in several ways to support this aspect. Furthermore, the BPMN has an already defined mapping to executable XML languages which could be used for the code generation.

As a profound disadvantage, the graphical notations that have been chosen for the visualization of the processes, lack a formal foundation. Formalism is required to define an unambiguous sub-set of the chosen notations that allows for automated code generation. Furthermore, the formalization acts as a foundation for the concepts and methodologies for variability aspects. Petri nets and process algebra form two different views of the formalization of the PESOA meta model. As this model is still under construction, it has to be investigated which approach could be used best for certain purposes. Ongoing work investigates the possible mapping of UML Activity Diagrams to Petri nets for qualitative and quantitative reasoning as well as simulation by existing tools. The adequacy of modern process algebra for the purpose of fulfilling the requirements of the e-business domain of the PESOA project has been shown [Puhlmann04]. The next steps include the definition of a formal specification of the graphical notations which will underpin the ongoing research.

Figure 28 **Mapping of the PESOA requirements to process modeling languages**



An outlook of the future work of specifying a PESOA meta model is shown in Figure 28. The approach of having several domain specific notations that share one meta model has several advantages. The first advantage is the better usability by different stakeholders that could utilize the notations known to their domain. Technically the approach concentrates on necessary aspects that are required for each domain instead of specifying an overloaded notation. Thereby all notations share a common meta model. This allows the possible addition of other notations as well as the transferability of variability concepts among the notations. The figure shows

that the PESOA requirements are made up of a union of the set of requirements from the e-business as well as the automotive domain. Those requirements overlap heavily. For example events and exception from the automotive domain can be assigned to the behavioural aspect of workflow, whereas input and output can be mapped to the functional and informational workflow aspect. The PESOA requirements are fulfilled partly by the investigated notations. The BPMN is well suited for many of the e-business requirements and overlaps with the UML Activity Diagrams which are used to fulfil many requirements of the automotive domain. A special requirement of the last domain is the representation of states which are representable by UML State Machines. However, those State Machines contain many concepts that are not required for the PESOA domain. The uncovered parts of the PESOA requirements are fulfilled by formalization or other notations that have to be investigated as needed. All mentioned notations are derived from the UML meta model or are close to them. The PESOA meta model could possibly be oriented on the UML meta-model with extensions for supporting variability.

Regarding the integration of Activity Diagrams, State Machines, and BPMN into existing domain engineering techniques, a smooth integration of Activity Diagrams and State Machines into KobrA should be possible, since older versions of Activity Diagrams and State Machines are already supported by KobrA. And due to the similarity of the BPMN and Activity Diagram metamodel an integration of BPMN into KobrA should be feasible as well.

6 Outlook

This report analyzed domain engineering techniques to support the modeling of variant-rich processes, investigated the special requirements of two different application domains, and discussed existing languages for modeling processes.

The results that have been achieved will be brought together to enable the modeling of generic, variant-rich processes in the e-Business and the automotive domain. To this end, we will first investigate the commonalities and differences between e-Business processes and technical processes as used in the automotive domain. The goal is to reuse techniques and also experience between the two domains. In a next step, a common model will be developed that captures the concepts needed to model processes in the two domains. This conceptual will be augmented with means to model variability and decision hierarchies in order to enable the modeling of generic processes. The final step is then to develop domain-specific customizations of the conceptual model for the two considered domains.

7 References

[Alonso97] Gustavo Alonso, Divyakant Agrawal, Amr El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert*, 12(5), 1997.

[Arpinar99] Ismailcem Budak Arpinar, Ugur Halici, Sena Nural Arpinar, and Asuman Dogac. Formalization of Workflows and Correctness Issues in the Presence of Concurrency. *Distributed and Parallel Databases*, 7(2):199-248, 1999.

[Atkinson01] C. Atkinson, J. Bayer, C. Bunse, E.Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. Component-based Product Line Engineering with UML. Component Software Series. Addison-Wesley, 2001.

[Atkinson02] C. Atkinson et.al. Component-based Product Line Engineering with UML. Addison-Wesley, New York, 2002.

[Basten98] T. Basten. In Terms of Nets: System Design with Petri Nets and Process Algebra. PhD thesis, Eindhoven University of Technol-ogy, Eindhoven, The Netherlands, 1998

[Bauer01] Thomas Bauer, Manfred Reichert, and Peter Dadam. Effiziente Übertragung von Prozessinstanzdaten in verteilten Workflow-Management-Systemen. *Informatik-Forschung und Entwicklung*, 16(2):76-92, 2001.

[Bayer99a] J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In Proceedings of the First International Symposium on Generative and Component-Based Software Engineering (GCSE '99), Erfurt, Germany, September 1999

[Bayer99b] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In Proc. of the Symposium on Software ReUsability (SSR'99), pp. 122-131, 1999.

[BEA] BEA Systems, IBM, Microsoft, SAP, Siebel Systems. Business Process Execution Language for Web Services Version 1.1, May 2003.

[Becker03] Jörg Becker. *Process Management*. Springer, Berlin, 2003.

[Bernardo02] M. Bernardo, P. Ciancarini, and L. Donatiello. Architecting Families of Software Systems with Process Algebra. ACM Transac-tions on Software Engineering and Methodology, 11(4):386-426, 2002.

[Brogi04] A. Brogi, C. Canal, E.Pimentel, and A. Vallecillo. Formalizing Web Service Choreographies. In Proceedings of First Interna-tional Workshop on Web Services and Formal Methods. El-sevier, 2004 (to appear).

[Bunse01] C. Bunse. Pattern-Based Refinement and Translation of Object-Oriented Model to Code, Fraunhofer IRB Verlag, 2001.

[Born04] M. Born, E. Holz, and O. Kath. Softwareentwicklung mit UML 2. Addison-Wesley, München, 2004

[Bock03a] C. Bock. UML 2 Activity and Action Models. in Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 43-53.

[Bock03b] C. Bock. UML 2 Activity and Action Models Part 2: Actions. in Journal of Object Technology, vol. 2, no. 5, pp. 41-56.

[Bock03c] C. Bock. UML 2 Activity and Action Models Part 3: Control Nodes. in Journal of Object Technology, vol. 2, no. 6, pp. 7-23.

[Bock04a] C. Bock. UML 2 Activity and Action Models Part 4: Object Nodes. in Journal of Object Technology, vol. 3, no. 1, pp. 27-41.

[Bock04b] C. Bock. UML 2 Activity and Action Models Part 5: Partitions. in Journal of Object Technology, vol. 3, no. 7, pp. 37-56.

[BPEL] BEA Systems, IBM, Microsoft, SAP, Siebel Systems. *Business Process Execution Language for Web Services Version 1.1*, May 2003.

[BPMI] Business Process Management Initiative. URL: http://www.bpmi.org (October, 7 2004).

[BPML] BPMI.org. Business Process Modeling Language, 2002.

[BPMN] BPMI.org. *Business Process Modeling Notation*, 1.0 edition, May 2004.

[Clements01] P. C. Clements and Linda Northrop. Software Product Lines: Prac-tices and Patterns. SEI Series in Software Engineering. Addison-Wesley, August 2001

[Chastek02] G. J. Chastek (ed). Software Product Lines. Proceedings of the Second International Conference (SPLC2), San Diego, California, USA, August 2002.

[Davulcu98] Hasan Davulcu, Michael Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan.
Logic Based Modeling and Analysis of Workflows. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 25-33. ACM Press, 1998.

[DC03] DaimlerChrysler Annual Report 2003. http://www.daimlerchrysler.com.

[Donohoe00] P. Donohoe (ed.). Software Product Lines - Experience and Research Directions. Proceedings of the First International Software Product Lines Conference (SPLC1), Denver, Colorado, USA, August 2000.

[EL03] Europa Lehrmittel, *Tabellenbuch Kraftfahrzeugtechnik*, Europa Fachbuchreihe, 2003.

[Haugen02] Bob Haugen and Tony Fletcher. *Multi-party Electronic Business Transactions.* http://www.choreology.com/standards/commentary/multiparty.htm (September 10, 2004), 2002.

[Hollingsworth95] David Hollingsworth. TheWorkflow Reference Model. Technical report, Workflow Management Coalition, Hampshire, 1995.

[Hoare78] C.A.R. Hoare. Communicating Sequential Processes. Communications of the ACM, 21(8):666-677, 1978.

[ISO02] ISO/IEC. High level petri nets – concepts, definitions and graphical notation. ISO/IEC 15909, 5 2002.

[Jensen92] Kurt Jensen. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1. Springer, 1992.

[Kang90] Kyo Kang, Sholom Cohen, James A. Hess, William E. Novak, A. Spencer Peterson; “Feature-Oriented Domain Analysis (FODA) Feasibility Study”; Technical Report CMU/SEI-90-TR-21; 1990

[Labrosse02] Jean J. Labrosse. MicroC/OS-II. C6MPBooks, 2002.

[Leen02] Gabriel Leen and Donal Heffernan. Expanding Automotive Electronic Systems. In IEEE Computer, January 2002, pages 88-93.

[Leymann00] Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, New Jersey, 2000.

[Lück04] Wolfgang Lück. *Lexikon der Betriebswirtschaft*. Oldenbourg Wissenschaftsverlag, München, 2004.

[Mauw96] S. Mauw. The formalization of Message Sequence Charts. Computer

Networks and ISDN Systems, 28(12):1643-1657, 1996.

[Mertens01] Peter Mertens. *Lexikon der Wirtschaftsinformatik*. Springer, Berlin, 2001.

[Milner89] R. Milner. Communication and Concurrency. Prentice Hall, New York, 1989.

[Milner92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proc-esses, Part I/II. Information and Computation, 100:1-77, 1992.

[Nord04] R. L. Nord (ed.). Software Product Lines. Proceedings of the Third International Conference (SPLC3), Boston, MA, USA, August 2002.

[OCL] Object Management Group. *UML 2.0 OCL Specification*. Object Management Group, 2003.

[OSEK] OSEK Open systems and the corresponding interfaces for automotive electronics. http://www.osek-vdx.org.

[OWL] Mike Dean and Guus Schreiber. *OWL Web Ontology Language*. W3C, 2004.

[Papazoglou02] Michael P. Papazoglou. The World of e-Business: Web-Services, Workflows, and Business Transactions. In Ch. Bussler, editor, *Web Services, E-Business, and the Semantic Web (WES) 2002, volume 2512 of LNCS*, pages 153{173, Berlin, 2002. Springer-Verlag.

[Pender03] T. Pender. UML Bible. John Wiley & Sons, 2003.

[PESOA] PESOA. *PESOA Projektbeschreibung*, 2003.

[PSL] Process Specification Language Homepage. http://www.mel.nist.gov/psl/ (September, 10).

[Puhlmann04] F. Puhlmann. On the Application of the pi-calculus to Workflow. Technical Report, Hasso-Plattner-Institute, 2004 (to appear).

[Sangiorgi03] D. Sangiorgi, and D. Walker. The pi-calculus: A theory of mobile processes. Cambridge University Press, Cambridge, 2003.

[Schäuffele03] Jörg Schäuffele und Thomas Zurawka. Automotive Software Engineering. Vieweg, July 2003.

[Scheer01] August-Wilhelm Scheer. *ARIS – Modellierungsmethoden, Meta-Modelle, Andwendungen*. Springer-Verlag, Berlin, 2001.

[Schildhauer03] Thomas Schildhauer. *Lexikon Electronic Business*. Oldenbourg Wissenschaftsverlag, München, 2003.

[Smith02] Howard Smith and Peter Fingar. *Business Process Management - The Third Wave*. Meghan-Kiffer Press, Tampa, 2002.

[Schnieders04] A. Schnieders, F. Puhlmann and M. Weske. Process Modeling Techniques. PESOA Report No. 01/2004, Hasso-Plattner-Institute, 2004.

[Schlenoff99] C. Schlenoff, M. Gruninger, M. Ciocoiu, and J. Lee. The Essence of the Process Specification Language. In *Special Issue on Modeling and Simulation of Manufacturing Systems in the Transactions of the Society for Computer Simulation International*, 1999.

[Störrle98] H. Störrle. An Evaluation of High-End Tools for Petri-Nets. Tech-nical Report, Institut für Informatik/PST, Ludwig-Maximilians-Universität München, 1998.

[UML] Object Management Group. UML 2.0 Superstructure Final Adopted Specification. Object Management Group, 2003.

[van der Aalst97] W. M. P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997, volume 1248 of LNCS*, pages 407-426, Berlin, 1997. Springer-Verlag.

[van der Aalst00] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. Technical Report BETA Working Paper Series, WP 47, Eindhoven University of Technology, 2000.

[van der Aalst02a] W. M. P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern Based Analysis of BPEL4WS. Technical Report FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[van der Aalst02b] Wil van der Aalst and Kees van Hee. *Workflow Management*. MIT Press, 2002.

[van der Aalst02c] W.M.P. van der Aalst, and K.M. van Hee. Workflow Manage-ment: Models, Methods, and Systems. Cambridge, MIT Press, 2002.

[van der Aalst03] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5-51, 2003.

[W3C] W3C Glossary and Dictionary. http://www.w3.org/2003/glossary/ (September 10, 2004).

[Weske98] Mathias Weske and Gottfried Vossen. *Workflow Languages.* Handbook on Architectures of Information Systems. Springer-Verlag, Berlin, 1998

[Weske00] MathiasWeske. *Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects*. Habilitationsschrift, Fachbereich Mathematik und Informatik, Universität Münster, Münster, 2000.

[White03] S. White. Process Modeling Notations and Workflow Patterns. Technical Report, IBM Corp., 2003.

[White04] ADTmag.com. Q&A: Stephen White, IBM: BPMN spec aims to help join business and IT. URL: http://www.adtmag.com/article.asp?id=9412 (October, 5 2004).

[WP] Workflow Patterns Homepage. http://tmitwww.tm.tue.nl/research/patterns/ (September 10, 2004).

[WSDL] Erik Christensen, Francisco Curbera, Greg Meredith, and Weerawarana Sanjiva. *Web Service Description Language (WSDL) 1.1*. IBM, Microsoft, March 2001. W3C Note.