Improving the Development of e-Business Systems by Introducing Process-Based Software Product Lines[1]

Joachim Bayer, *Mathias Kose, Alexis Ocampo

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1, D- 67663 Kaiserslautern, Germany
{joachim.bayer, alexis.ocampo}@iese.fraunhofer.de

*ehotel AG
Greifswalder Strasse 207, D-10405 Berlin, Germany
mkose@ehotel.de

In the e-Business domain, workflows are central artifacts that are used to specify application systems. To realize reuse at a large scale for e-Business application systems, therefore, workflows need to be reused systematically. To this end workflows must be classified, documented, and stored in a way that enables their identification, evaluation, and adaptation in order to integrate them in an application. Software product line engineering is an established and approved software engineering approach that addresses these issues by handling a number of similar software systems together, enabling large scale reuse during the development and maintenance of the different systems covered by the product line.

In this paper, we transfer the concepts of software product line engineering to the domain of e-Business systems by applying the product line techniques to workflows and present initial validation results.

1 Introduction

Survival in today’s highly dynamic business environments requires that organizations continuously adapt their business processes. Success and growth rather than mere survival require that this adaptation be rapid enough to realize the competitive advantage offered by new business opportunities. The conduction of business in the internet (e-business) including buying and selling but also services and collaboration can be seen as one of these new important business opportunities. Mechanisms for rapid description, implementation, and deployment of such business processes become important. Currently, business processes are often represented by business process models. Business processes models are partially implemented through workflows [9]

[1] This work has been partially funded by the PESOA project (Process Family Engineering in Service-Oriented Applications) funded by the German federal ministry of education and research (BMBF) (Förderkennzeichen: 01 ISC 34E)

and deployed and executed in workflow environments, which show graphically the different steps of a business process (i.e., the business logic). According to [16], business processes connect a set of business functions, where the connections are controlled by business rules. Those business rules are specific for an enterprise and specific at a certain point in time. However, changes in business rules and objectives are an everyday issue that demands capabilities to be able to react and adapt to such changes. Therefore, new rules and objectives can inevitably result in a large number of processes that vary in relatively minor ways. One way to control this proliferation and its attendant risks is to analyze commonalities and differences between the different process models in order to identify process variants and justifications for them [13], and to systematically integrate them in a software product line [6].

The following sections describe briefly the basic concepts of product line engineering and the mapping that we have done to process-based product lines, describe the details of the approach we have developed, and provide a preliminary validation (in terms of an example of its use).

2 Conceptual Foundation

2.1 Product Line Engineering Concepts

The underlying idea of product line engineering is to reuse common parts of related software systems. To this end, varying aspects of software systems, that is, differences among them are explicitly documented. Product line engineering distinguishes two development phases – domain and application engineering – as presented in Figure 1. The initial activity, scoping, defines which systems are members of a product line and which systems are outside the product line. Scoping is done by investigating a set of concrete products, be it already existing, planned, or envisioned products. The result of scoping is a set of products that make up the product line along with the features of the different product line members.

Based on a scope definition, domain engineering identifies the common features (commonalities) and the variable features (variabilities) of the identified products. Commonalities define the skeleton of the systems in the product line; variabilities

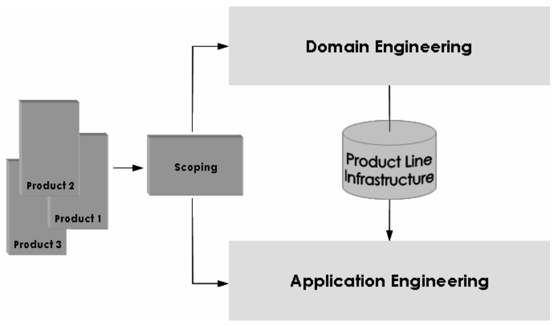


Figure 1. Product Line Engineering

bound the space of required and anticipated variations of the products in the product line. Each artifact produced during domain engineering contains the commonalities and specially labeled variabilities. These so-called variant-rich artifacts are stored in the product line infrastructure.

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

2.2 Process Based-Oriented Product Lines

A number of approaches for software product line engineering have been proposed [4],[7],[12]. Application domains that use processes, such as workflow or technical processes, as driving software development artifacts are, however, neglected to a large extent by product line research. The main problem in applying product line engineering techniques in such domains is that processes describe flows of activities and, consequently, variability covers different flows. The techniques traditionally proposed in product line engineering, however, provide means for the modeling of static diagrams rather than for dynamic ones. For example, the modeling of variability that results in different sub-processes that are exchanged for different products is not well supported. Another issue is that software generation traditionally also focuses on static models.

In this section, we present our approach for process-based product line engineering. The approach is based on PuLSE™ [2] (PuLSE™ is a registered trademark of Fraunhofer IESE) that is an approach for product line engineering that is developed and used in technology transfer projects since 1997. To adapt PuLSE™ for process-based product line engineering, we combined it with variability mechanisms [14] and software generation [8].

The core concepts of our approach to process-based product line engineering are variant-rich workflows or processes, which are workflows or processes that contain variabilities. To augment workflows used to model e-Business systems with the possibility to model variability in an explicit way, we use the approach proposed in [11]. This approach forms the basis for variability and decision modeling in PuLSE ™ and

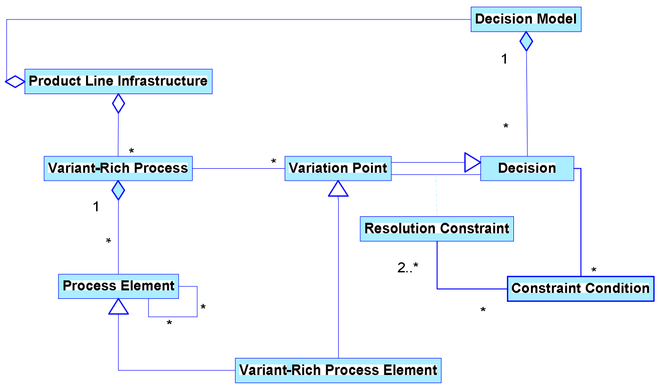


Figure 2. Process-based Product Line Concepts

provides a systematic way to extend any given software engineering artifact to be generic, that is, to enable the explicit modeling of variability in that artifact.

As presented in Figure 2, a product line infrastructure contains variant-rich processes and decision models. A variant-rich process contains variation points that represent its variability. A decision model contains the relationships among the variations of a product line infrastructure. Such decision models contain decisions, which are variation points that constrain the resolution of other variation points. A variant-rich process contains process elements, for instance activities, inputs, outputs, or roles. Those process elements that contain variation points are called variant-rich process elements.

Figure 3 provides an example that illustrates these concepts. It shows the flow between the “Create Order”, “Pay Order”, and “Send Invoice” process elements of an online shop. The “Pay Order” process element contains three alternatives (telephone, credit card, and bank transfer). The process element has one interface that interacts with the “Create Order” process element. At this point three alternatives split, and one of them must be chosen in order to resolve this variation. The resolution of the variation determines the path taken by the flow. The three alternatives converge in another interface that joins them. This interface is used to communicate the output of the “Pay Order” process element. The same can be observed in the case of the “Invoice” process element, an output that contains two alternatives (America or Europe), and two interfaces. This means that depending on the continent of destination, the invoice to be sent to the customer will have different fields of information (e.g., currency, address). One optional variation point can be assigned to the “Email” output process element. “Email” has only two alternatives i.e., yes or no. Therefore, once the varia-

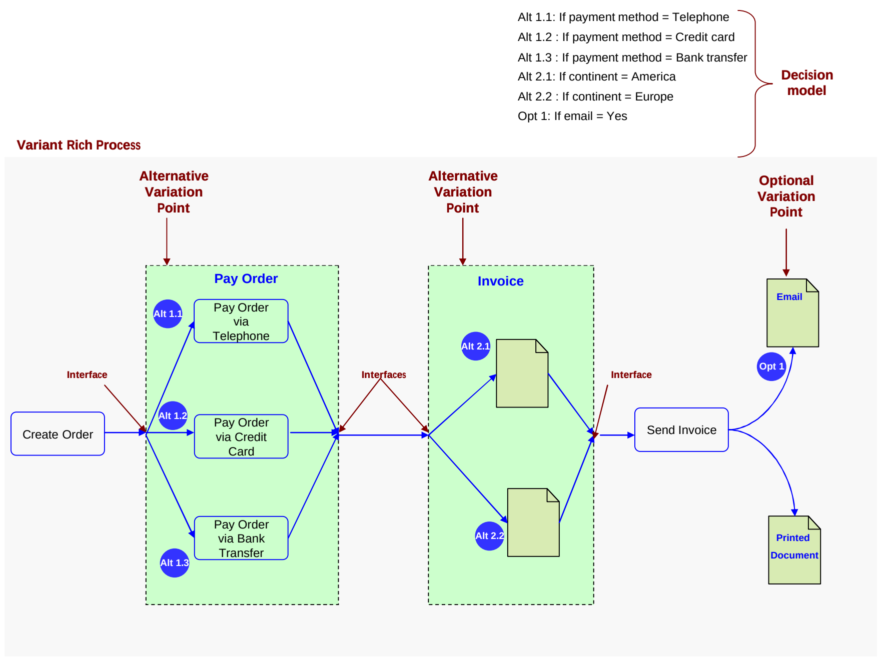

Alt 1.1: If payment method = Telephone
Alt 1.2 : If payment method = Credit card
Alt 1.3 : If payment method = Bank transfer
Alt 2.1: If continent = America
Alt 2.2 : If continent = Europe
Opt 1: If email = Yes
Decision model
Variant Rich Process
Alternative Variation Point
Alternative Variation Point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

Figure 3. Variant-rich Process Example

tion points are resolved, the client has the possibility of receiving the invoice both as printed document or email.

2.3 The Systematic Approach for Developing Process-Based Product Lines

Figure 4 shows our systematic approach for developing a process-based product line.

As mentioned above, the initial activity in product line engineering is *scoping*. The underlying idea of *scoping* is on the premise that one shall obtain as much return on investment as possible from the effort of establishing a process-based product line infrastructure. Using as input an existing or a planned set of process-based products a subset of such products is selected. Afterwards, the selected products are related to the features that they should offer. This information is recorded in a *domain scope definition*.

The *domain analysis* begins by using the defined domain scope as input for identifying relationships among features (e.g., consists-of, requires). Afterwards, in the activity *model features*, such relationships are captured in a hierarchical structure [10] or a tabular representation.

The resulting feature model can be used as basis for *identifying* and documenting the requirements for those processes that will be part of the process-based product line infrastructure. Such processes shall be conceived as building blocks that can be reused.

The domain design begins with the *design processes* activity. Here, using as input the list of identified processes, a commonality analysis among processes is performed in order to identify variant-rich process elements. At the moment there are not many techniques or approaches on how to perform such a comparison. One idea can be

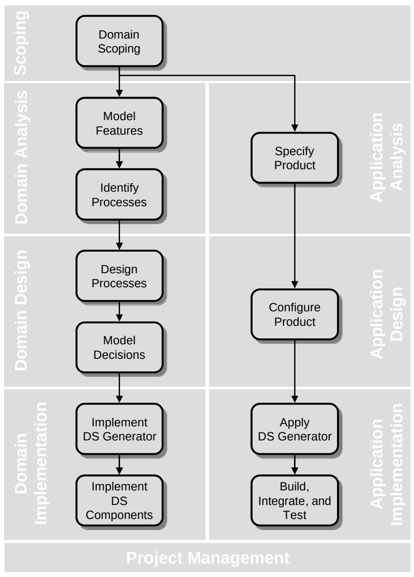


Figure 4. Process-based Product Line Engineering

taken from [12], where a systematic comparison of a set of software process models is illustrated. The commonalities and variabilities detected among variant-rich process elements are then integrated into their respective variant-rich process.

Relationships among variation points are identified and documented in the *decision model*.

This way, a process-based product line infrastructure that contains variant-rich processes elements, process elements, and a decision model has been produced.

The next step is *domain implementation*. It starts with the activity *implement domain-specific generator* that consists of identifying the domain-specific functionalities to be covered by a generator based on the commonalities and variabilities contained in the process-based product line. Code fragments implementing these functionalities are defined. They are connected to the process' variabilities, that is, each variation point is annotated by one or more code fragments.

Once the domain-specific functionalities have been identified, *DS components are implemented* as follows: First, functionalities that are to be implemented by generic components are identified based on the commonalities present in the process-based product line. Such components are referred to as *runtime components*. Then, components that are needed to process the generator's output are identified. They are referred to as *infrastructure components*. Once the DS components are implemented, a process-based product line infrastructure can be used for automatically generating new products according to new requirements.

The first step to derive a concrete product from the product line infrastructure is *application analysis*. It starts by *specifying the new product* based on the scope definition of the existing process-based product line infrastructure, and the feature model. Those features that are estimated to be realizable are mapped to the actual products from the process-based product line infrastructure. Such mapping must be documented in a *product feature model*. Those features that are not yet planned in the process-based product line shall be documented in a list of not covered features, which will be later integrated in the scope of the process-based product line.

The next step is to *configure the product*, in which the decision model is used for resolving the variation points based on the new product features. The resolution of the variation points and their relationships are documented in the *resolution model*.

Finally, the *appliance of the domain-specific generator* starts with importing data from a resolution model, followed by triggering the generation of *target code*. If there are additional variabilities that are not part of the process-based product line, for example technical ones specific for the target platform, they can be configured and resolved before triggering the code generation.

The generated target code is subject to further processing by the use of infrastructure components, including the domain-specific ones. The resulting executables have to be *built and integrated* with the needed runtime components. Together they form the product that might be *tested* in order to complete the implementation.

More details on the approach can be found in [3].

3 Validation

ehotel AG is a technology organization that specializes on developing software suitable for processing hotel reservations. It distinguishes because of its software development experience and know-how in the traveling business but especially in the hotel industry. ehotel AG develops and operates a software platform that supports hotel booking operations. The platform can be accessed through a browser interface or through a XML-/Web service interface. The XML-/Web service interface allows the integration of the platform in external IT-Systems. The rationale behind having such an interface was to integrate ehotel's solution with as many different systems as possible such as traveling systems of large corporate groups, traveling services offered by other web-Sites, or travel companies' internal applications. It was found that those systems supported a common hotel booking process. However, due to the different needs and scenarios of such systems, different types of requirements applied for functions such as search, select, reserve, or cancel. Each system type, therefore, needed a customized version of ehotel's product.

This is a classical situation where the product line approach can be used for better reusability of software products. ehotel has followed this approach systematically in the context of the PESOA project. The PESOA project's main goal is the design and prototype implementation of a platform for process family engineering and their application in the e-business and automotive areas. This goal is addressed by enhancing the approved technologies from the area of domain engineering, product line engineering, and software generation with new methods from the area of workflow management. The following sections present example of artifacts produced when process-based product line engineering was applied at ehotel in the context of the PESOA project. We focus in the case study on analysis and design, and thus leave out implementation. More details on the case study can be found in [14].

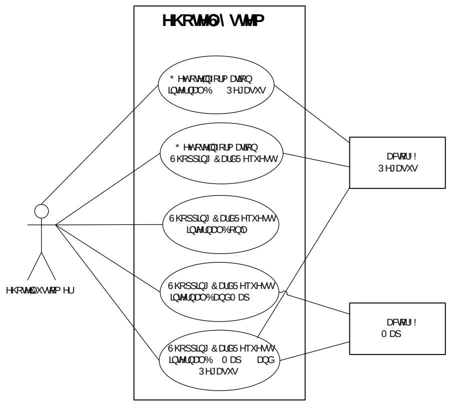


Figure 5. Informing Use Case Diagram

3.1 Domain Scope Definition

The selection of a subset of e-hotel’s process-based software was driven by the customer’s point of view. Use cases helped to sketch this point of view and to identify the following set of sub-processes: “informing”, “booking”, “canceling”, and “charging”. Figure 5 shows the respective use case diagram for the “informing” sub-process that identifies the different ways ehotel customers can retrieve information.

3.2 Feature Model

A feature model captures and relates the characteristics of the different product line members. Common and varying characteristics are distinguished in feature models.

Figure 6 shows an excerpt of the feature model for ehotel’s booking engine. The features for the “informing” and the “booking” sub-processes are modeled in detail. For the booking engine, there are common characteristics (denoted by full circles), like hotel details expressing that every booking system provides the possibility to acquire information on hotels. There are also optional characteristics (denoted by hollow circles). For example, pictures, description, and map in the hotel details express that these are the different possibilities for hotel details that are provided by the different booking systems. The third type of characteristics shown in the figure is alternative. Alternatives denote different ways to realize characteristics from which one is chosen for a specific booking system. In the example, an alternative feature is the map that can be realized either as static map or as dynamic map. The figure shows that for the varying characteristics all possible values are captured.

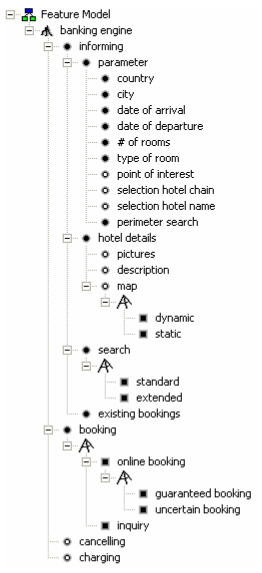


Figure 6. Feature Model (modeled with fmp [1])

3.3 List of Processes

The next step in process-based product line engineering is the elicitation of processes that are needed to provide the features collected in the previous step. The list of processes mostly reflects the hierarchical organization of the feature model. The list of processes that was elicited based on the feature model in Figure 6 is:

- Informing
  - Search
    - Standard
    - Extended
  - Parameter
    - get countries
    - get cities
    - get Points-of-Interest (POIs)
    - get hotel chain
  - Hotel details
    - get Pictures
    - get Description
    - get Map
    - existing bookings
      - get All Bookings
      - get booking details
- Booking
  - online booking
    - guaranteed booking

        - uncertain booking
    - inquiry
- Canceling
- Charging

These are the (sub-) processes that have been identified for the ehotel booking engine and that will be modeled as variant-rich processes in the next step.

3.4 Variant-rich Processes

Variant-rich processes are the core artifact in a process-based product line. They describe the behavior of the different product line members and thus determine the process-based product line. Variant-rich processes contain variation points to determine process elements that vary between different product line members. We use the variability mechanisms described in [14] for modeling variation points. These variability mechanisms enable the expression of different types of variation using stereotypes and other notation-specific modeling mechanisms.

Figure 7 shows the booking engine top-level process using the BPMN notation [4]. The top-level process contains “informing”, “booking”, “canceling”, and “charging” as sub-processes. The process contains three types of variation points. The “charging” and the “cancellation” sub-processes are optional, denoted by the Null stereotype that expresses that the respective sub-processes are either present or not in a specific booking engine. “Booking” has an abstract stereotype; this means that there are different realizations possible for this sub-process.

The variable stereotype for the “informing” sub-process expresses that there are

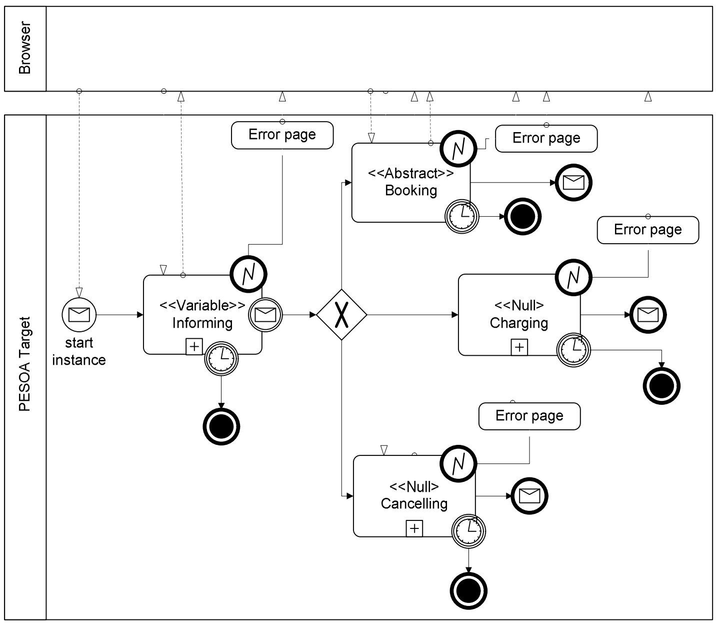
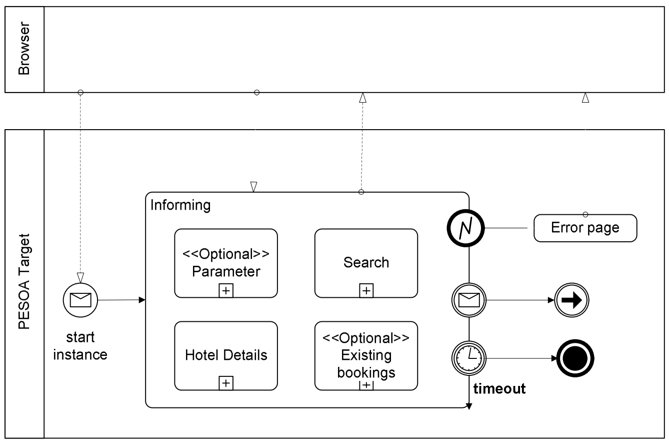


Figure 7. Variant-rich Booking Engine Process



variabilities within the sub-process. This is shown in Figure 8 that depicts the “informing” sub-process. Figure 9 refines the “search” sub-process and shows for the abstract activity perform search two possible realizations, a standard and an extended search.

3.5 Decision Model

The variation points in the variant-rich processes must be resolved in order to derive specific processes that describe concrete booking engines. This resolution is sup-

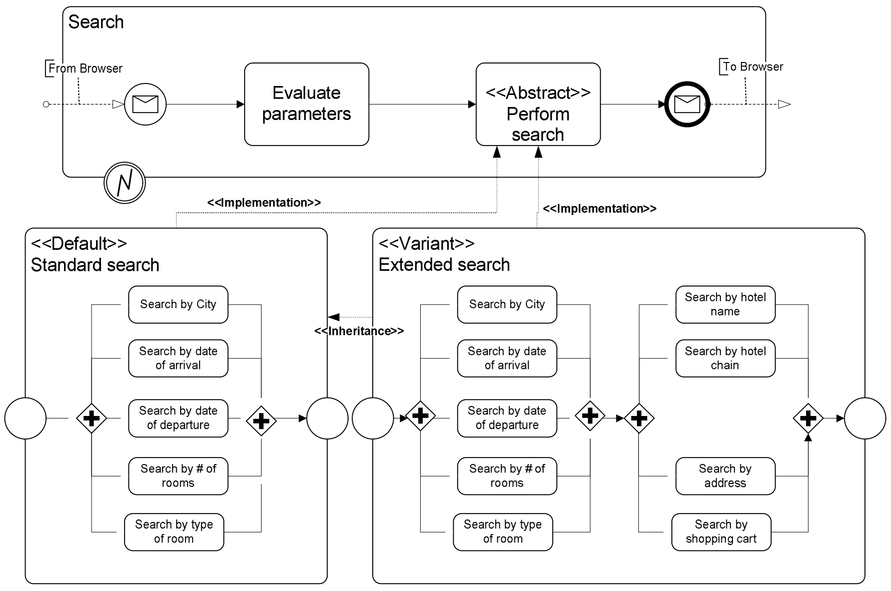


Figure 9. Variant-rich Searching Process

ported by decision models that relate features to variation points and document how a variation point must be resolved if a booking engine provides a given feature. Table 1 shows the decision model excerpt for the "searching" process shown in Figure 9.

Table 1. Decision Model Excerpt

ID	Process	Question	process element	Resolution	Effect
Search-ing.1	searching	Is extended search required?	Perform search	yes	Perform search = extended search
				no	Perform search = standard search

The decision shown in Table 1 describes how the "searching" process is instantiated for the two possible cases, standard and extended search. When the effect is applied to the respective process, the abstract activity "perform search" in Figure 9 is replaced by either a sub-process realizing the standard or the extended search, respectively, depending on the decision taken.

The decision model is a collection of the decisions for all variation points in the different variant-rich processes.

3.6 Product Feature Model

In the following, we describe the instantiation of the variant-rich processes for a hypothetical ehotel customer that uses an instance of the ehotel booking engine derived from the process-based product line infrastructure.

As a first step, the required features from the feature model (compare Figure 6) are selected. The result is shown in Figure 10.

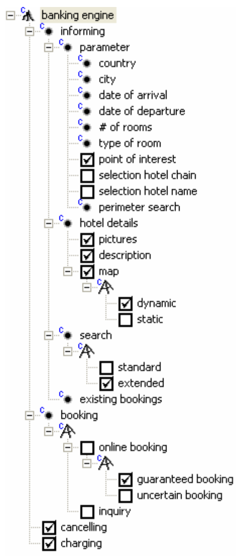


Figure 10. Product Configuration (modeled with fmp [1])

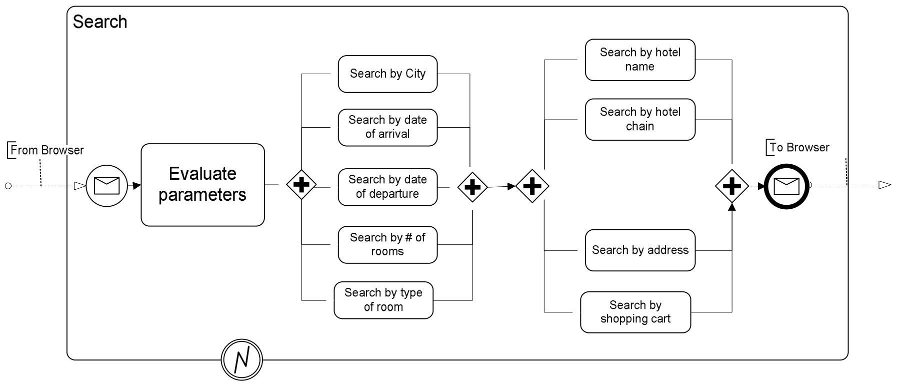


Figure 11. Search Process Instance

3.7 Configured Product

Using the selected features, the decision model can be instantiated by answering the different questions. The application of the appropriate effects on the variant-rich processes resolves the processes leading to concrete processes for the product. In Figure 11, the variant-rich search process in Figure 9 is instantiated using the features selected in the product feature model in Figure 10. The result is a “search” process providing extended search features.

4 Summary and Outlook

The booking engine plays a dominant role in the software system of the ehotel AG. A large variety of functionalities are implemented because of different requirements of individual users as well specific requirements of corporate customers. The result of the different market requirements is of high complexity for the ehotel-system. The process-based product line engineering shows a practical way to handle this complexity.

Based on existing specific business processes a generic, variant-rich process is derived. With feature diagrams and decision models this generic process can be configured. By using software generators customer specific software instances can be produced.

Process-based product line engineering forces a better structuring of the existing ehotel software system and future developments. After the setup of the process-based product line infrastructure, a faster and more reliable delivery of a customized version of the booking engine to new customer requirements is possible. The quality of the overall software system is improved and the time to market is reduced. This improved agility helps the ehotel AG on the customer side for example, to offer products to niche markets and has therefore a positive impact to the company. Overall the planning process of the development is improved; this results in higher delivery reliability. At the end ehotel achieves a higher customer satisfaction.

Acknowledgements

We want to thank our colleagues in the PESOA project that supported us in the development of the presented example, namely Paul Bouché (ehotel AG), Dennis Plötner (ehotel AG), Thomas Hering (University of Leipzig), and Andrej Werner (University of Leipzig). We especially want to thank Frank Puhlmann (Hasso-Plattner-Institut at the University of Potsdam) for supporting the example development and for reviewing an early version of the paper.

References

[1] M. Antkiewicz, K. Czarnecki.: FeaturePlugin: feature modeling plug-in for Eclipse, Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, p.67-72, October 24-24, 2004.
[2] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J. –M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In Proceedings of the Fifth Symposium on Software Reusability (SSR'99), May 1999.
[3] J. Bayer, W. Buhl, C. Giese, T. Lehner, A. Ocampo, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland, M. Weske. Process Family Engineering: Modeling variant-rich processes. PESOA-Report No. 18/2005, Juni 2005.
[4] G. J. Chastek (ed). Software Product Lines. Proceedings of the Second International Software Product Lines Conference (SPLC2), San Diego, California, USA, August 2002.
[5] Business Process Management Initiative (BPMI): Business Process Modeling Notation (BPMN), Version 1.0, www.bpmi.org, Mai 2004.
[6] P. Clements and L. Northrop. Software Product Lines. Practices and Patterns. Addison-Wesley, 2002.
[7] 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.
[8] C. Giese, H. Overdick, W. Buhl. Realisierungsstrategien für Prozessfamilien: Werkzeuge für Modellierung und Generierung. PESOA-Report No. 15/2005, Process Family Engineering in Service-Oriented Applications, Juni 2005.
[9] D. Hollingsworth. The Workflow Reference Model. Technical report, Workflow Management Coalition, Hampshire, 1995.
[10] K. Kang, S. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson; "Feature-Oriented Domain Analysis (FODA) Feasibility Study". Technical Report CMU/SEI-90-TR-21, 1990.
[11] D. Muthig: A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. Stuttgart: Fraunhofer IRB Verlag, 2002 (PhD Theses in Experimental Software Engineering Vol. 11). Kaiserslautern, Univ., Diss., 2002.
[12] R. Nord (ed.). Software Product Lines. Proceedings of the Third International Conference (SPLC 2004), Boston, MA, USA, August - September, 2004.
[13] A. Ocampo, F. Bella, J. Münch. Software process commonality analysis. Software Process: Improvement and Practice. Vol. 10(3), pp. 273-285, 2005.
[14] D. Plötner, M. Kose, T. Hering, A. Werner. Prozesse im E-Business am Beispiel ausgewählter Geschäftsprozesse des Partners ehotel AG. PESOA-Report No. 20/2005, Juni 2005.
[15] F. Puhlmann, A. Schnieders.: Process Family Engineering: Variability Mechanisms, Technical Report PESOA-Report No. TR 17/2005, Process Family Engineering in Service-Oriented Applications, Jun. 2005.

[16] G. van de Putte, T. Benedett, D. Gagic, P. Gersak, K. Krutzler, M. Perry. Intra-Enterprise Business Process Management. IBM Corporation. IBM International Technical Support Organization. IBM Reedbook. October 2001.