Modeling Variability in Business Process Models Using UML

Maryam Razavian, Ramtin Khosravi
Department of Electrical and Computer Engineering,University of Tehran
m.razavian@ece.ut.ac.ir, rkhosravi@ece.ut.ac.ir

Abstract

Variability management is a major concern in successful exploitation of variabilities and commonalities of software product families which also affects different aspects of development activities. Modeling variabilities among products of a family is a key aspect of variability management discipline. To use software product line approach in information systems context, it is necessary to bring in variability in different phases of the life cycle, including business process modeling which is recognized as a key part of developing enterprise information systems. Consequently, modeling variability in business process models becomes an issue worthy of consideration. We use UML 2 activity diagrams to model business processes. Modeling solutions are proposed and classified based on the origins of variability at business process level. We have also studied the ways to avoid cluttering the model when including variability. An example case is utilized to clarify different aspects of our proposed method.

Keywords

Variability modeling; Software product line; Business process modeling

1. Introduction

Software product line engineering deals with development of families of related products, while making use of commonalities and variabilities among them [4,7,12]. *Variability management* is about handling introduction, use, and evolution of variability [15]. *Variability modeling* methods represent the variabilities among products of a family in such a way that it provides better understanding of variabilities and also assists engineers in task of variability management.

Different variability modeling methods have been proposed by various product line approaches [1,3,12 ,14]. There are two aspects in variability modeling that are considered in existing methods: representing variability in software artifacts and modeling variabilities and their relationships in a *variability model* including information about decisions in the domain space and relationships among them. In this paper, we restrict our attention to modeling variability in business process models which could be categorized as the first aspect.

The process of requirement analysis for family of software systems is called *product line analysis,* [4,5,12] which is the first phase in product line development process. One of the main activities of this phase, *problem analysis,* is the process of understanding the real world problem and proposing solutions to meet stakeholder needs. In the domain of information systems, *business modeling* is one of the major problem analysis techniques used to define a system as a set of business processes. Consequently, the role of business modeling in information systems development is to facilitate understanding the structure and dynamics of the business domain [9]. Since a business model effectively represents the system's major processes, roles, and responsibilities, it can be a rich source for requirement analysis. Additionally, it helps to ensure that customers, end users, and developers have a common understanding of the target organization [6,9,10].

Since in enterprise information systems, business process modeling is recognized as a commonly used technique, and it is not supported by the existing product line engineering approaches, the need for a specialized product line analysis method in information system domain is felt. As a result, the explicit representation of variability becomes an essential part of a business process modeling. To study the effect of variabilities on business processes, the analyst must be able to explicitly model variable elements in the business process models.

Our contribution in this paper is to present a method for modeling variability in business process models based on UML 2 activity diagram which has the following aspects:

- A *variability representation* method specified for this model based on UML 2 profile
- A *hierarchical representation* method intended for abstracting away complexities

Our proposed method has two important characteristics. First, the method is based on UML 2 as the business process modeling notation. An obvious

978-0-7695-3099-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ITNG.2008.132

benefit of UML is that it is a standard language, widely used among practitioners. Also, UML is extensible by means of standard extension mechanisms. Another important characteristic of UML is that it can be easily understood by both software experts and business people and it could be integrated with other software modeling mechanism effectively. Devising a variability modeling method based on a specific modeling language enables more precise modeling of variability concepts and it also provides a basis for more detailed analysis of the effects of variabilities in the business processes.

The second characteristic of our method is that it focuses on the business process model and the specific concerns associated to this artifact are addressed by the method and it allows a precise definition of variability in the business process. Our method for modeling variability is categorized based on the origin of variability (having variation in control flow, data flow and actions) and the type of variability (optional, alternative).

Another problem to be addressed is how to avoid cluttering the business process representation with process details and its associated variabilities. We propose a solution based on hierarchical representation of the process into sub processes such that the top level sub process encompasses the core activities and their associated variability while the lower level sub processes express all details related to higher level activities and variabilities residing in them.

The structure of paper is as follows: preliminaries to our method are represented in section 2. The proposed variability representation method is discussed in section 3, whereas section 4 presents the proposed method of abstracting complexities in business process representation. Our method is compared to the related work in section 5.

2. Preliminaries

2.1. Variability Concepts

Before explaining details of the variability modeling method, we review the basic concepts involved in the method. Variability is defined in [14] as the ability of a software system or artifact to be configured, customized or changed for use in a specific domain. In the context of software product lines, variability is the ability of a core asset to adapt to usages in different products that are within the product line scope. Realization of variabilities within core assets is made by variation points.

Considering variants as product specific realizations of variabilities, variation points can be more precisely defined as places in the core asset where a choice between zero or more variants is made. They should be explicitly represented on assets of different abstraction levels.

Different types of variation points are as follows [14]:

- An optional variation point is the choice of selecting zero or one from the one or more variants.
- An alternative variation point is the choice between one of the one or more variants.

2.2. Example Case

In this section, we introduce a sample procurement system to study our variability modeling method for business processes. Generally, procurement systems could be grouped into two main categories: B2B systems which the customer is an organization and B2C system in which an individual person is the customer. In B2B systems each customer has a contract with a supplier for purchases from that supplier. A customer could browse the catalogs and selects items to purchase. The customer order should be checked to see if there is a valid contract between customer and supplier. Having validated the customer contract the order is fulfilled by supplier and shipped to customer. Since the customer is an organization invoice is sent to customer by supplier, having the invoice approved by customer, an electronic payment is sent to bank.

Although the B2C procurement system has a lot in common with the B2B system they vary in some cases. The customer provides personal details, such as address and credit card information. This information is stored in a customer account. Having validated the customer's credit card, the order is fulfilled. The payment consists of charging customer's credit card account. The variabilities which occur at architectural level are depicted in Table 1.

Table 1: Variabilities of procurement system

Variability	Description
Product Information	Customers' reviews and pictures of the products are optional information in catalogs
Customer Account Validation	Validation is performed by checking customer's credit card or customer's contract
Charging Customer	Customer is charged by sending invoice or charging the credit card
Insurance	Delivery may contain shipment insurance or not

3. Representing Variability in Business Process

In this section we propose a solution for representing variability information in business process model. In our work, variability is realized by having multiplicity in business process elements. In other words, business process level variability is enabled by having optional or alternative elements.

As variabilities may occur in assets of all levels of abstraction, each software artifact induces particular challenges for variability representation. In order to define a variability representation method for a specific artifact, different types of variable elements in that artifact should be considered. Moreover, in order to derive a product specific model from the corresponding asset a choice between different types of variants should be made. Variability arises in three forms in business process models:

- Variation in control flow
- Variation in data flow
- Variation in actions

To support variability within business processes models, we propose an extension to UML activity diagram by defining a UML profile. Our method proposes a solution for modeling variability in various situations which are formed based on the type of variable element and variation point type, i.e. optional and alternative.

3.1 Variation in Control Flow

Sequence of performing activities of a process can be different among various products. Variability in business model could be represented by specifying a distinct path for each variant. The optional and alternative paths describe the variation in control flow and determine the characteristics of a given member of the product line. Variation points in this case are starting points where a single path splits to a group of variant paths. Since, in order to select each of variant paths a condition should be evaluated, the term variation point is basically equal to a simple decision. Hence, one way of modeling variation points could be by means of decision node element. The element which models a variation point is a decision node distinguished from the ordinary decision nodes in the process by assigning one of <<alt_vp>> or <<opt_vp>> stereotypes. In this case, an optional variation point indicates the decision about selecting zero or one from the one or more associated paths. An alternative variation point is the choice between one of the two or more associated paths. A sample of an alternative variation point is shown in Figure 1 when the main flow splits into two branches according to the customer type. In case the customer type is home customer the *Charge Customer* action is performed while the other sequence of actions takes place for the organizations. The optional control flow can be modeled by means of the decision node as well. However, in this case there is a possibility in which no associated path is selected. The modeler should take this case into consideration in order to avoid any problem in the business process model.

Another possibility of modeling variation in control flow which is inherently a decision is by means of join node. A join node is a control node that synchronizes multiple flows and has multiple incoming edges and one outgoing edge [16]. The join node can issue a token just when a condition is true. A join specification is a Boolean expression attached to a join. Each time a token arrives at the join, the join specification is evaluated and if true, an output token is emitted. The join specification here could be used for selecting a specific variant path. However, one drawback of this modeling technique is that since normally the specification of join node is not displayed the term variation point can not be represented explicitly. In this case variation points can be represented by means of UML notes.

3.2 Variation in Data Flow

In the structured analysis and design methods data flow diagrams (DFD) are one of the main artifacts which describe the system by showing how input data is transformed to output results through a sequence of functional transformations. Capturing the flow of data and its associated transformations is essential in information system domain. In the software product line approach variability should be captured in flow of data as well. Activity diagrams in UML 2 support the main building blocks for modeling data flows passing among processes. One of the major changes occurred to UML 2.0 activity diagrams is support for modeling data flows since this feature in UML 1.x is rather weak.

Activity diagrams offer pins for modeling the input and output data of actions and activity parameter nodes for the input and output data of activities. Data objects are also used to represent flow of data among business process elements. A data store node is a modeling element for indicating the persistent storage of data [16].

Variability in data flow arises in two forms: variation in data and variation in data stores. *Variation in data* arises when the data which is available at a specific

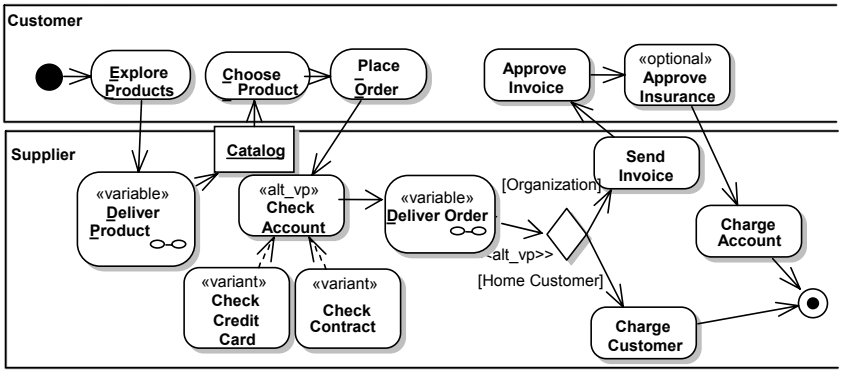

Figure 1: Modeling variability in sample procurement process

point of process and is used or created by particular set of activities (or actions) has different types among products. In other words, type of data which is used as an input to a particular activity or type of output data is different among products. In this case the data can be represented in forms of data objects, pins or parameter nodes. Here, each variant is marked with <<variant>> stereotype and is dependent to the variation point element. Variation point is modeled by the same UML element devised for modeling data (pins, parameters or data objects) and is named in a way that covers semantics of all variants. An example of this modeling method is represented in Figure 2, based on the example in [13].

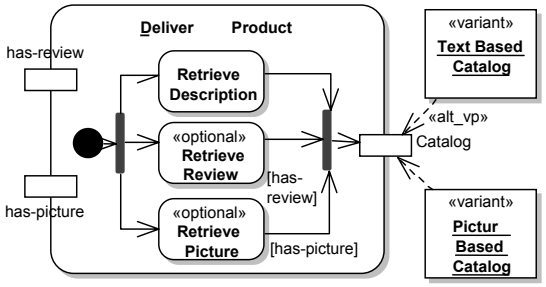

Figure 2: Modeling variability in "Deliver Product Info" activity

In this case the output of *Deliver Product Info* activity can have two types of *Picture Based Catalog* and *Text Based Catalog.* These two types of catalogs participate in the products alternatively. In this case variability occurs in form of having variation in data. Consequently, the flow of data between the *Deliver Product Info* activity and *Choose Product* action is variable as well. In some situations, a specific flow of data exists in just a set of products. In this case, an optional variability has occurred within the data flow and is modeled by marking the data object with <<optional>> stereotype.

Variation in data stores happens when the element used to define permanently stored data is variable among products. Since the data store element is basically a data object modeling variability in data stores is analogous to modeling variability in data objects. The semantic distinction between data objects and data stores is illustrated with <<data store>> stereotype.

3.3 Variation in Actions

Actions are pieces of behavior that are atomic and can not be decomposed into smaller actions. In software families a particular action may exist in a single process of one product and not in the same process of others. This implies that one process could be performed by different set of activities in a software product line. In this case, variability occurs within actions of a process. When an action of a particular process can just participate in set of products, the action is known as an optional action and is marked with the <<optional>> stereotype. In the procurement system example the *Retrieve Picture* and *Retrieve Review* actions just participate in products which deliver pictures and reviews in their product information. In this case variability has occurred in form of having optional actions (see Figure 2).

An alternative variability happens when a set of actions participate in a specific product exclusively. In procurement system example the actions of *Check*

Credit Card and *Check Contract* are used in B2C and B2B systems alternatively. In this case, an action which covers the semantics of both actions acts as the alternative variation point (*Check Account* action). The action variants are marked with the <<variant>> stereotype and are dependant to the variation point marked with <<alt_vp>> (see Figure 1).

4. Abstracting Complexities in Business Process Representation

As mentioned previously, business process models facilitates better understanding of how work is done within information systems. This model is also used as a basis to key requirements of those systems. In product line engineering, the main goal of business process modeling is better understanding of the key processes of the family and their associated commonality and variabilities. According to [6] the business process model cannot and should not contain all details of the business. The model which expresses all details associated to the process risks becoming complex and hard to comprehend. Consequently, one important goal of the business process model is to provide a big picture of how work is done by abstracting away complexities and representing the core activities. Variability defies this purpose by inducing complexities to the business process representation and on the other hand, we believe, its representation at the business process level is essential.

To solve this problem, we make use of the fact that business process can be modeled at different levels of detail. UML 2 allows processes to be refined into smaller activities hierarchically until it reaches an atomic level of actions. In this case the top level activity in the potential activity hierarchy represents the entire process. This feature of activity diagram facilitates representation of variability hierarchically. We propose using a simplified model of process model encompassing the core activities and their associated variabilities. At this level the details of variability modeling techniques employed inside the activities are excluded and just the activities encompassing some sort of variability are distinguished from others by <<variable>> stereotype. Whereas, lower level activities include details of variability information associated to them and their associated modeling techniques. Since inherently each activity is a process, variability modeling techniques proposed for business processes are applicable for activities as well. For instance in the procurement example *Deliver Product Info* and *Deliver Order* are composite activities which include some kind of variability (see Figure 1). However the details of types of variabilities and their associated modeling technique is depicted in lower level representation of these two activities (Figure 2,3).

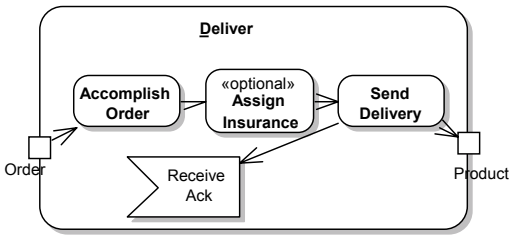

Figure 3: Modeling variability in "Deliver Order" activity

5. Related Work

Over the past few years several variability modeling methods have been developed.. Some of these methods such as COVAMOF [14], FODA [11] and VSL [2] are rather general for all software artifacts and do not support the concerns associated to the specific artifacts such as business process models. Consequently, we do not discuss them in our comparison.

A number of variability modeling methods have been devised for artifacts related to problem analysis phase such as use case models and class diagrams. Halmans and Pohl [8] and Gomaa [7] propose methods for modeling variability in use cases. In their work UML is used as the modeling language and any model element may potentially be a variant in a product line context. All these methods propose UML profiles in which stereotypes and tagged values are defined in order to specify variation points and variants. The drawback of these approaches is that extensive use of stereotypes and tagged values results in a cluttered model and will decrease scalability of the model.

Pohl et al. [12] uses orthogonal variability models to represent variability at requirements level. These models improve separation of concerns by separating variability and requirements artifacts. However, their representation does not indicate exactly which elements are variable and how the variabilities should be realized. Although they consider activity diagrams, they ignore the specific characteristics of the model since they use a single modeling method based on their orthogonal variability model in all artifacts.

Schnieders [13] proposes set of variability mechanisms for process family architectures. In this work, representations of variability mechanisms in UML activity diagrams are also addressed. The proposed variability mechanisms are used to realize variability in family of process oriented software

systems. Here, business process models are design level artifacts which represent architecture of the process oriented systems. The proposed variability mechanisms are mostly about how to implement variabilities at this level. Also, the rules for deriving product architecture from the process family architecture for each variability mechanism are presented. The concerns associated to business processes in problem analysis phase including identification of origins of variabilities and modeling solutions for representing variability are not focus of this work.

6. Conclusion

In this paper, we presented a method for modeling variability in business process models. We devised a method based on UML 2 as the process modeling notation, in order to avoid proposing an abstract general method which does not consider specific details of modeling languages. It also makes our method more suitable for practical use. Another advantage is that it is an extensible language and therefore available UML based tools can be used for modeling PLA, by means of our UML profile. Also, as mentioned in section 4, the composite structure of activities makes it possible to have hierarchical abstraction in the process representation.

The method helps the analyst recognize how the variability arises at business process level by facilitating identification of origins of variability at this level. Moreover, based on the origin of variability (having variation in control flow, data flow and actions) and the type of variability (optional, alternative) modeling solutions are proposed. Also, the precise definition of variability in the business process model enables the analyst keep track of the effects of variabilities in different elements of the business processes.

In order to achieve an effective variability management method, it is essential to employ an orthogonal variability model including information about variabilities derived from the problem space. We believe that an orthogonal variability model based on UML can benefit from ease of integration with other models. Also this kind of variability model can facilitate developing tools intended for taking over some of variability management tasks. This is a complement to our method, which can be studied in future work.

Finally, we believe that our method could be integrated with workflow management systems to execute, monitor and coordinate business processes which have variabilities within information system product lines.

References

[1] F. Bachman and P. Clements, "Managing variability in software product lines", Technical Report No. CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon, September 2005.

[2] M. Becker, "Towards a general model of variability in product families", Proceedings of the 1st Workshop on Software Variability Management, February 2003.

[3] K. Berg, J. Bishop and D. Muthig,"Tracing software product line variability – from problem to solution space", Proceedings of SAICSIT, pp. 182-191.

[4] J. Bosch, "Design and use of software architectures: adopting and evolving a product line approach", Addison Wesley, 2000.

[5] G. Chastek, P. Donohoe, K. Kang, S. Thiel, "Product line analysis: a practical introduction", Technical Report No. CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon, June 2001.

[6] H. Eriksson and M. Penker, "Business modeling with UML: business patterns at work", John Wiley & Sons, 2000.

[7] H. Gomaa, "Designing software product lines with UML: from use cases to pattern-based software architectures", Addison Wesley, 2004.

[8] G. Halmans, K. Pohl, "Communicating the variability of a software product family to customers", Software and Systems Modeling, 2003, pp. 15–36.

[9] I. Jacobson, G. Booch, and J. Rambaugh, "The unified software development process", Addison Wesley, 1999.

[10] I. Jacobson, M. Ericsson, and A. Jacobson, "The object advantage: business process reengineering with object technology", Addison-Wesley, 1994.

[11] K. Kang, S. Cohen, J. Hess, W. Novak and S. Peterson, "Feature oriented domain analysis (FODA) feasibility study", Technical Report No. CMU/SEI-90-TR-021, 1990

[12] K. Pohl, G. Bockle and F. van der Linden," Software product line engineering foundations, principles, and techniques", Springer-Verlag Berlin Heidelberg, 2005.

[13] A. Schnieders, "Variability mechanism centric process family architectures", Proceedings of the 13th Annual IEEE International Conference on the Engineering of Computer Based Systems ECBS 2006, IEEE Computer Society Press, 2006.

[14] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "COVAMOF: A framework for modeling variability in software product families", Proceedings SPLC 2004, Lecture Notes on Computer Science Vol. 3154 (LNCS 3154), Springer Verlag, August 2004, pp. 197-213.

[15] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques", Information and Software Technology, Vol. 49, 2007, pp. 717-739.

[16] Object Management Group (OMG), "UML 2.0 Superstructure Specification", 2005.