By Magnus Eriksson, Jürgen Börstler, and Kjell Borg

SOFTWARE PRODUCT LINE MODELING MADE PRACTICAL

An example from the Swedish Defense Industry.

In this article, we describe an approach integrating use case modeling and feature modeling to support the description and maintenance of a common and complete use case model for an entire family of systems. This approach, referred to as PLUSS (Product Line Use case modeling for Systems and Software engineering), is currently applied in several large-scale defense projects within BAE Systems Hägglunds AB.

Traditional use case modeling targets single-system development and provides poor support for variability management. To support effective variability management a single common system family model should be enforced. Variant information must be kept and maintained in one place. In PLUSS, we follow the same ideas as proposed by Griss et al. [6] and combine use case modeling with feature modeling (see Table 1).

FODA-like feature models give an overview of the features of a system family and their interrelationships [10]. Features can then be modeled by groups of use cases. However, there may also be variant use cases. We distinguish four kinds of variations in use case models for product families [2]:

- Use cases might or might not be included in a particular product.
- Certain scenarios of an included use case might be excluded from a particular product.
- The flow of events within an included scenario might vary.
- Cross-cutting aspects that can affect several use cases on several levels. For example, the existence of different sets of use case actors (system users) in different products.

In PLUSS, all such variations are managed using a common feature model, that is, the feature model provides a total overview of all variants within a product line use case model. A particular set of features for a specific product within a product line, therefore directly corresponds to a particular set of concrete use cases for that product. We accomplish this by relating use cases, use case scenarios, and steps within use case scenarios to features of appropriate types in the feature model (see Figure 1 for an example). To support cross-cutting variability, we include textual parameters into our use case descriptions. These parameters, which can be single-valued or multi-valued as illustrated by "$PARAM_1" and "$PARAM_2" in Figure 1, are represented in the feature model as well. If a product includes a use case description with such a parameter, it must also instantiate that parameter by selecting one (or several) feature value entities from the corresponding "parametric feature" in the feature model.

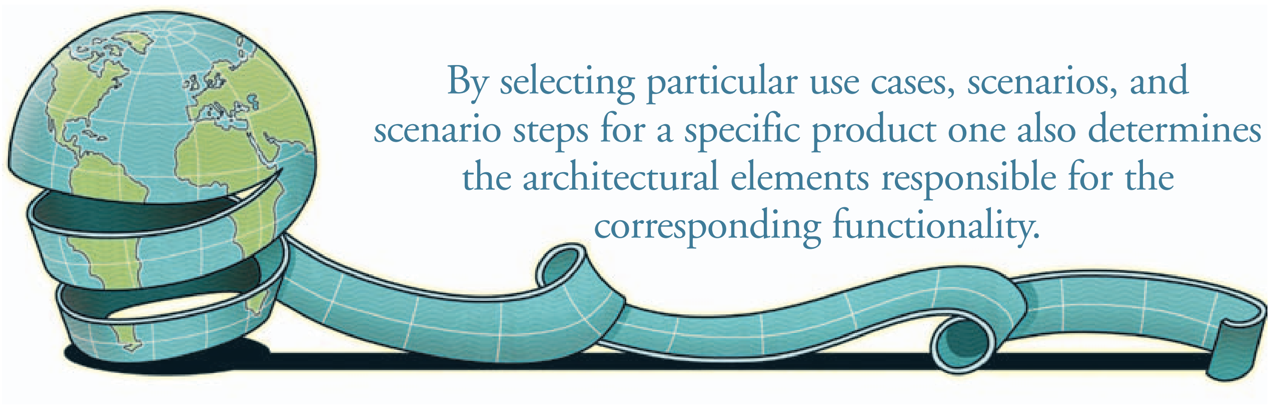
	Variability Mechanism			
Approach	**Use case**	**Scenario**	**Step**	**Cross-cutting**
Jacobson et al. [8] (RSEB)	Using the generalization and extend relationships in UML use case diagrams by using a different use case stereotype icon for abstract use cases.	Possible*	Possible*	Only within a single use case specification using textual parameters.
Griss et al. [6] (FeatuRSEB)	Using a feature model that is linked to the use case model.	Possible*	Possible*	Only within a single use case specification using RSEB parameters.
Fantechi et al. [4] (PLUC)	N/A	N/A	N/A	Only within a single use case specification using the tags "Alternative," "Optional," and "Parametric".
Gomaa [5] (PLUS)	Using UML stereotypes in use case diagrams ("kernel", "optional" or "alternative" use cases) and by modeling use cases packages as features in a feature model.	Possible*	Possible*	Only within a single use case specification using a section describing all variation points according to a variation point template.
John and Munthig [9]	Using UML stereotypes in use case diagrams ("variant") and marking sections of diagrams as optional.	Using XML-like tags to mark scenarios as optional or alternatives.	Using XML-like tags to mark steps as optional or alternatives.	N/A
Moon et al. [11] (DREAM)	Using UML stereotypes in use case diagrams ("common" and "optional" use cases).	Possible*	Possible*	N/A
Halmans and Pohl [7]	Using UML stereotypes in use case diagrams ("variant"). Using explicit "variation points"; mandatory and optional variants can be grouped and assigned cardinalities.	Possible*	Possible*	Variation points can be associated with several use cases within the same use case model.
PLUSS	Use cases are instantiated by selecting features in the corresponding feature model.	Correspondence to features in the feature model.	Correspondence to features in the feature model.	Use cases can be parameterized.

* Could be managed by describing variant scenarios and variant steps as separate use cases that extend the original use case. This strategy is, however, likely to fragment the use case model into too many and too small use cases when applied to product lines of non-trivial systems (see [5] for further discussion of this issue).

Table 1. Modeling variability in product line use cases. A brief overview.

Traceability and Reuse

For each use case, there are one or more use case realizations describing how different design elements collaborate to realize a specific use case goal (see Figure 2). In PLUSS, we utilize this "built-in" traceability by maintaining use case realizations together with their



By selecting particular use cases, scenarios, and scenario steps for a specific product one also determines the architectural elements responsible for the corresponding functionality.

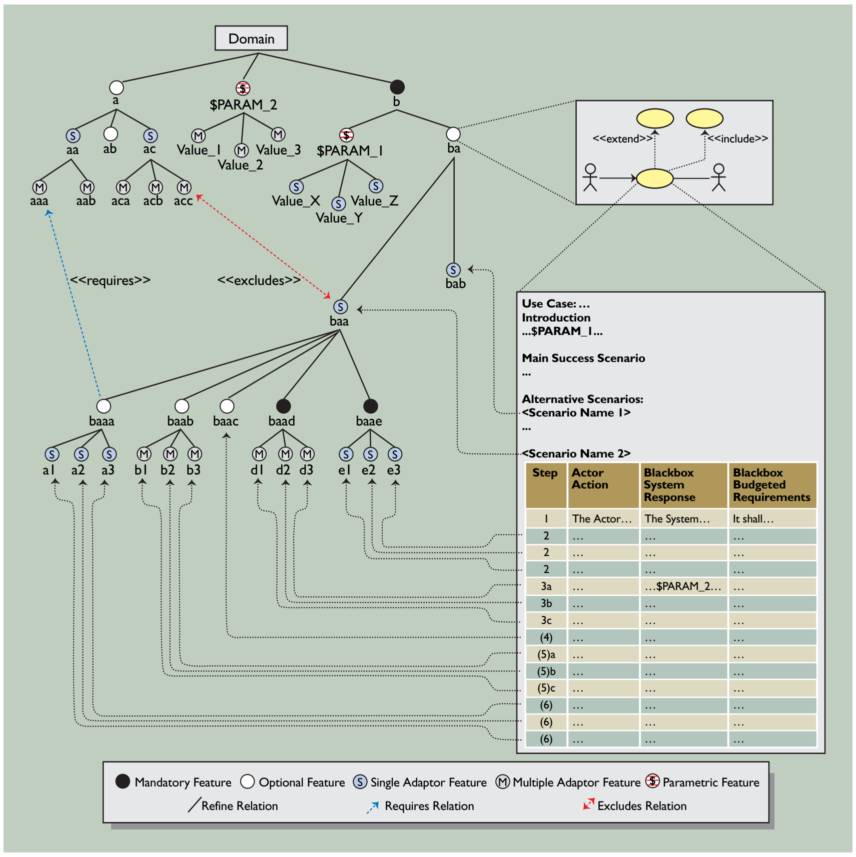


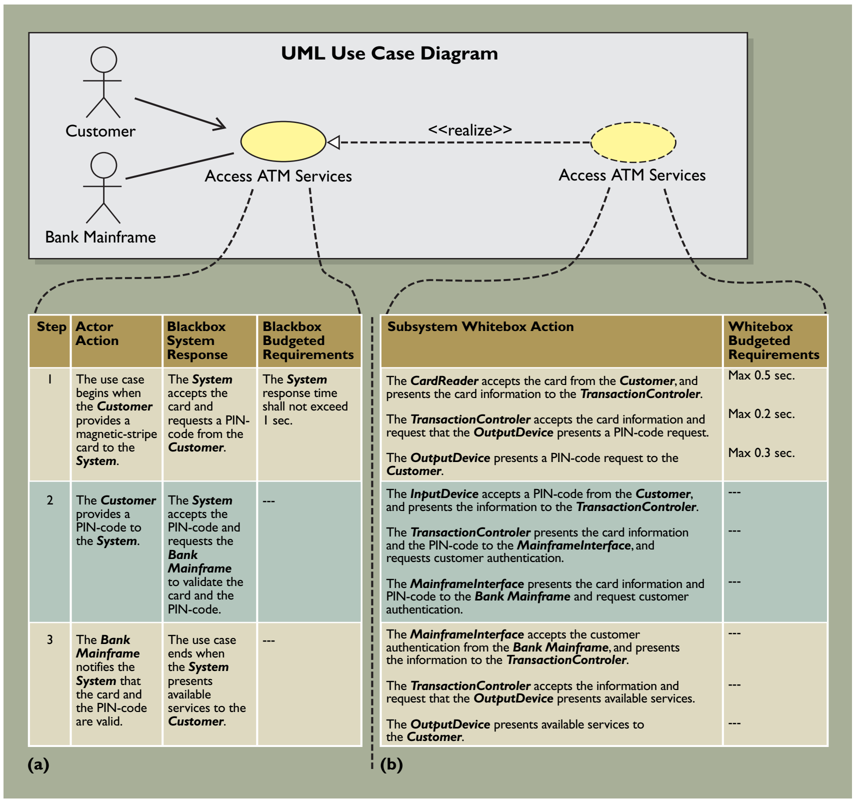
Figure 1. An example of the relationship between features and use cases.

corresponding product line use case models. By selecting particular use cases, scenarios, and scenario steps for a specific product one also determines the architectural elements responsible for the corresponding functionality. This leads to an improved reuse infrastructure.

Product derivation is basically done by first adding new requirements to the model(s) and then using the (possibly updated) feature model to choose among its variants. We use the selected features as a filter to view only the relevant parts of the full product line model. This yields three types of reports: A Use Case Model Survey, Use Case Specifications, and Use Case Realizations. The use case model survey gives an overview of all use cases for the product. For each use case in this survey, there is a use case specification describing black-box system level scenarios for this use case and use case realizations describing the corresponding white-box realizations on subsystem level (see [3] for tool support).

Change Cases and Maintenance

Change cases [1] are basically use cases that specify anticipated future changes to a system. They provide a relation "impact link" that supports traceability to affected use cases, if the change case is realized. Modeling change cases allows product line designers to plan for and, more effectively, accommodate anticipated future requirements in a domain. In PLUSS, we utilize change cases as a means for developing Change Requests (CR) and Engineering Change Proposals (ECP) to our product line. New requirements are modeled as change cases to (1) provide an overview of the current delta between the original and the current model, and (2) to provide stronger support for



Step	Actor Action	Blackbox System Response	Blackbox Budgeted Requirements
1	The use case begins when the ***Customer*** provides a magnetic-stripe card to the ***System***.	The ***System*** accepts the card and requests a PIN-code from the ***Customer***.	The ***System*** response time shall not exceed 1 sec.
2	The ***Customer*** provides a PIN-code to the ***System***.	The ***System*** accepts the PIN-code and requests the ***Bank Mainframe*** to validate the card and the PIN-code.	---
3	The ***Bank Mainframe*** notifies the ***System*** that the card and the PIN-code are valid.	The use case ends when the ***System*** presents available services to the ***Customer***.	---

Subsystem Whitebox Action	Whitebox Budgeted Requirements
The ***CardReader*** accepts the card from the ***Customer***, and presents the card information to the ***TransactionControler***.	Max 0.5 sec.
The ***TransactionControler*** accepts the card information and request that the ***OutputDevice*** presents a PIN-code request.	Max 0.2 sec.
The ***OutputDevice*** presents a PIN-code request to the ***Customer***.	Max 0.3 sec.
The ***InputDevice*** accepts a PIN-code from the ***Customer***, and presents the information to the ***TransactionControler***.	---
The ***TransactionControler*** presents the card information and the PIN-code to the ***MainframeInterface***, and requests customer authentication.	---
The ***MainframeInterface*** presents the card information and PIN-code to the ***Bank Mainframe*** and request customer authentication.	---
The ***MainframeInterface*** accepts the customer authentication from the ***Bank Mainframe***, and presents the information to the ***TransactionControler***.	---
The ***TransactionControler*** accepts the information and request that the ***OutputDevice*** presents available services.	---
The ***OutputDevice*** presents available services to the ***Customer***.	---

Figure 2. An example of the notation used for describing (a): use case scenarios (system level), and (b): use case realizations (sub-system level).

	Questionnaires	Interviews
Pros	+ Provides a better understanding of the modeled domain. + Provides a better overview of the product line. + Provides stronger means for short- and long-term product planning.	+ Provides a better overview of the product line and its requirements space. + Improves the overall quality of models. + Ease maintenance of models. + Notations easy to use and understand. + Resulting models are a better means of communication with non-software stakeholders in the organization. + Provides a better overview of dependencies within the model. + More coherent models, makes it easier to find information. + Reduces the effort for requirements analysis. + Reduces the effort needed for specification work. + Central source of information about a domain. + Change cases provide a good overview of what is new and what has been done before (the current delta in the model).
Cons	– DOORS and Rose not integrated well enough, leads to time consuming manual synchronization between the tools.	– Requires a stronger configuration management function. – Requires a stronger product planning function and smart scoping decisions to avoid a feature explosion in the model.

Table 2. Summary of initial experience with PLUSS at BAE Systems Hägglunds AB (see [2] for details).

product line change impact analysis. However, once accepted for implementation in a product within the product line, these change cases are transformed to use cases as illustrated in Figure 3.

Experience

Initially, PLUSS was used to reengineer a subsystem product line for an onboard vehicle information system resulting in a product line model containing approximately 200 features and 40 (parameterized) use cases. A case study after the first two delivered products revealed very positive results (see [2], summarized in Table 2). Since then several new products have been added to this model, and PLUSS is now also used to model other product lines within the organization.

PLUSS is a relatively simple extension to "standard" use case modeling and fits very well within the Unified Process framework. This implied low adoption costs, since use case modeling and a RUP-based process were already in place at our company. It was furthermore relatively easy to extend our single-system CASE tool environment, consisting of Telelogic DOORS and IBM-Rational Rose, to support some parts of PLUSS (see [3]).

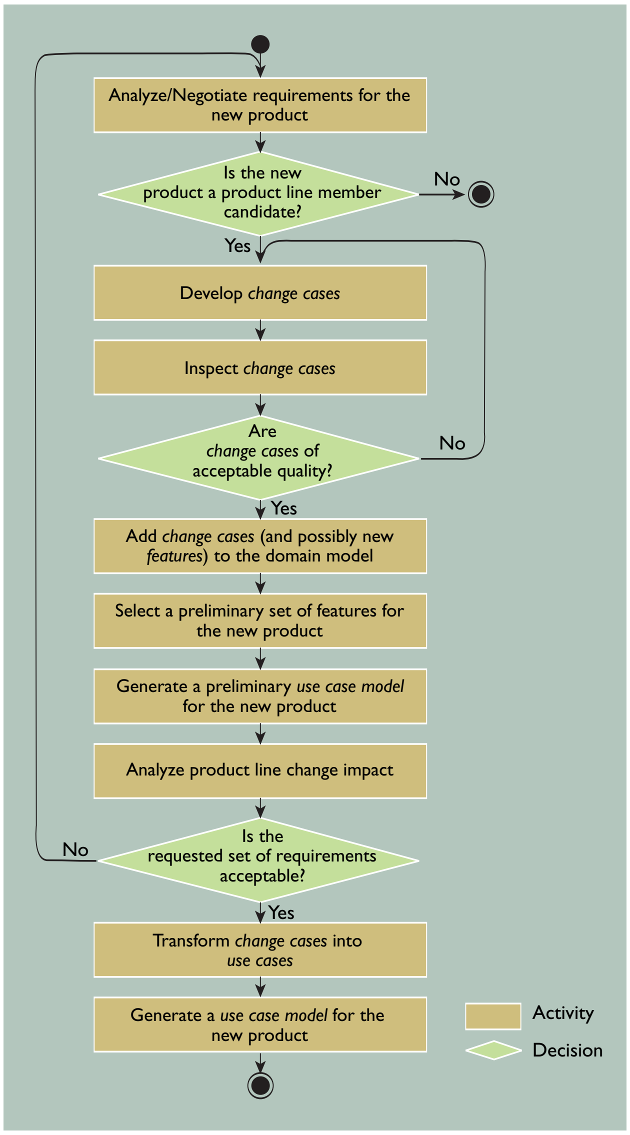
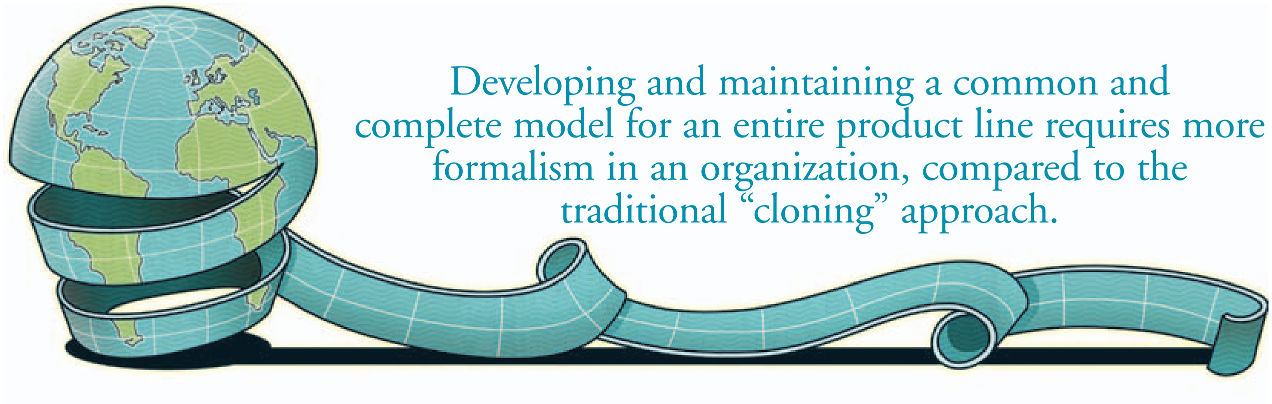

Analyze/Negotiate requirements for the new product
Is the new product a product line member candidate?
No
Yes
Develop change cases
Inspect change cases
Are change cases of acceptable quality?
No
Yes
Add change cases (and possibly new features) to the domain model
Select a preliminary set of features for the new product
Generate a preliminary use case model for the new product
Analyze product line change impact
Is the requested set of requirements acceptable?
No
Yes
Transform change cases into use cases
Generate a use case model for the new product
Activity
Decision

Figure 3. Adding a new product to a product line model.

We would like to stress two other lessons learned from this work. First, developing and maintaining a common and complete model for an entire product



Developing and maintaining a common and complete model for an entire product line requires more formalism in an organization, compared to the traditional "cloning" approach.

line requires more formalism in an organization, compared to the traditional "cloning" approach. The simple reason for this is that several products are managed at the same time instead of separately. Secondly, adopting such an approach may lead to resistance from some product development teams. They no longer "own" the complete product requirements documentation, since they are a part of the product line core assets base. This reduces the degrees of freedom for the product development teams, which, however, is a desirable effect from a management point of view to support effective reuse. C

References

1. Ecklund, E., Delcambre, L., and Freiling, M. Change cases—Use cases that identify future requirements. In *Proceedings of OOPSLA'96* (San Jose, CA, 1996), 342–358.
2. Eriksson M., Börstler, J., and Borg, K. The PLUSS approach—Domain modeling with features, use cases and use case realizations. In *Proceedings of the International Conference on Software Product Lines*, LNCS, Vol. 3714, Springer-Verlag, 2005, 33–44.
3. Eriksson M., Morast, H., Börstler, J., and Borg, K. The PLUSS toolkit—Extending telelogic DOORS and IBM-Rational Rose to support product line use case modeling. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering* (Long Beach, CA, 2005), ACM Press, 2005, 300–304.
4. Fantechi, A., Gnesi, S., Lambi, G., and Nesti, E. A methodology for the derivation and verification of use cases for product lines. In *Proceedings of the International Conference on Software Product Lines*, LNCS, Vol. 3154, Springer-Verlag, 2004, 255–265.
5. Gomaa, H. *Designing Software Product Lines with UML—From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley, 2004.
6. Griss, M., Favaro, J., and d'Alessandro, M. Integrating feature modeling with the RSEB. In *Proceedings of the Fifth International Conference on Software Reuse* (Vancouver, BC, Canada, June 1998), 76–85.
7. Halmans, G. and Pohl, K. Communicating the variability of a software-product family to customers. *Software and Systems Modeling 2*, 1 (Mar. 2003), 16–36.
8. Jacobson, I., Griss, M., and Jonsson, P. *Software Reuse—Architecture, Process and Organization for Business Success*. Addison-Wesley, 1997.
9. John, I. and Muthig, D. Product line modeling with generic use cases. In *Proceedings of SPLC-2 Workshop on Techniques for Exploiting Commonality Through Variability Management, Second Software Product Line Conference* (San Diego, Aug. 2002).
10. Kang, K., Cohen, S., Hess, J., Novak, W. and Peterson, A. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.
11. Moon, M., Yeom, K., and Chae, H. An approach to developing domain requirements as a core asset-based on commonality and variability analysis in a product line. *IEEE Transactions on Software Engineering 31*, 7 (July 2005), 551–569.

Magnus Eriksson (magnus.eriksson@baesystems.se) is a systems engineer with BAE Systems Hägglunds AB and a doctoral candidate in Software Engineering in the Department of Computing Science at Umeå University in Sweden.
Jürgen Börstler (jubo@cs.umu.se) is an associate professor in Software Engineering in the Department of Computing Science at Umeå University in Sweden.
Kjell Borg (kjell.borg@baesystems.se) is a Software Quality Manager with BAE Systems Hägglunds AB, in Örnsköldsvik, Sweden.