Challenges in Managing Behavior Variability of Production Control Software

Miao Fang
Siemens AG
Erlangen, Germany
miao.fang@siemens.com

Georg Leyh
Siemens AG
Erlangen, Germany
georg.leyh@siemens.com

Christoph Elsner
Siemens AG
Erlangen, Germany
christoph.elsner@siemens.com

Jörg Dörr
Fraunhofer Institute IESE
Kaiserslautern, Germany
joerg.doerr@iese.fraunhofer.de

***Abstract*—Software product line engineering helps organizations to achieve systematic software reuse by taking advantage of commonalities and predicted variability. Variability management has been considered as one important issue in product line development. In this paper, a variability analysis in production control systems reveals that the variability in such systems lays in the dynamic behavior and interaction of configured components, which we consider as behavior variability.**

This paper identifies the three main challenges to be solved for applying a product line approach to the domain of production control systems: (1) the selection or design of a proper variability language for describing the flexible behavior variability, (2) the need to model variability of behavior at different levels of granularity, as well as to map the elements among different levels, (3) the binding of behavioral variation points and variants into the various involved systems in a manageable way.

***Index Terms*—Behavior Variability, Variability Management, Product Line Infrastructure**

I. Introduction

Software product line (SPL) engineering has been widely recognized as a successful approach to help development teams to reduce development cost, shorten the time-to-market, and achieve better software quality [2]. Variability is a key concept in product line engineering, to represent how products in one product family can differ and be derived [4], [5]. Managing variability concerns the representation, the dependencies, and the instantiation of variability during the complete variability life cycle [11].

Many approaches for modeling and managing variability have been developed and proved as effective ways to help on software product line design and development, such as FORM, KobrA, or SPLIT, just to name a few [4]. Feature modeling is a primary approach, which is widely used for capturing variation points and exposing variability [4], [6], mainly for domain analysis. Based on feature modeling, a number of approaches have been suggested to enhance the expressiveness of variability in a more comprehensive manner [3], [14] in order to improve on its capabilities for product configuration. However, commonly, the modeling and management of structural variability remain to be in the main focus.

In the context of this paper, we take an example from production control systems to investigate the influence of behavior variability both for the *representation* and *implementation* of variability. Practitioners commonly use behavioral models, such as use cases, activity models, or state machines for specifying system behavior and functions [8] in requirement elicitation and design phases to ensure that stakeholders share the same understanding of system functions [1]. For representing the behavior of a product family, the behavior variability turns out to be as important as the other variability, especially for production control systems, which we analyze in this paper.

The reason is that the functions of such kind of systems comprise the flexible combination and interaction of multiple instances of components, whereas each single component already contains an enormous amount of variation points. This increases the complexity of managing their behavior variability. The example illustrates the difficulty of representing the variability of such systems using existing structural and behavior modeling approaches. We also present the open issues regarding the representation of behavior variability in different abstraction layers, and the binding of the behavioral variants for implementation.

This paper consists of three remaining sections. Section II introduces behavior variability by presenting the different behavior of components and their interaction in an exemplary production process. Section III presents the challenges of representing and resolving the behavior variability. Section IV summarizes the challenges and concludes this paper.

II. An Exemplary Case

Production control systems are the software systems that manage the manufacturing process in factories. These systems work in different domains, such as food, chemical, or automotive production. The size of manufacturing plants ranges from small to very large-scale. The production processes for each domain can be totally different, involving possibly mechanical or electronic engineering. However, from a software perspective, the software, running on servers, controllers, electronic devices, or sensors, is highly reusable crossing different domains. This is because the individual components normally have repeated and limited behavior, but the combination of them actually forms the different functions to support the manufacturing activities. For organizations who hold a broad product portfolio of such systems, SPL approaches provide the potential to significantly shorten the development cycle. Based on our observation, the current reuse of software components and their configurations is following a copy-and-modify style.

978-1-4673-6449-2/13 © 2013 IEEE

PLEASE 2013, San Francisco, CA, USA

Without a systematic SPL approach, the reuse can turn out to be very tedious and error-prone.

Figure 1 illustrates a typical production process in the automobile domain, controlled and coordinated by production control software. It shows a high-abstraction-level process for car production, without any details in each activity. As can be seen, the production systems have a process-oriented nature. The first step is to transport the required materials and components of the expected car for preparation. The second step, the prepare step, already has various sub-processes. For instance, the required numbers of some components are selected from the storage containers; some components are necessary to have pre-assembly steps, to be ready for further integration. The third step is to transport the prepared components to different assembly benches. The last assembly step can be performed both by automatic work and manual work. At the end of this process, a car is produced.

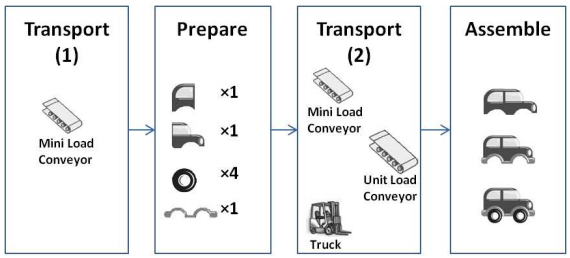

Fig. 1. A High-Level Production Process

We analyze the *Transport* activity within this car production example to understand the behavior variability. There are already two explicit *Transport* activities shown in Figure 1. If we further analyze them, we see the *Transport* behaves differently depending on different situations:

- In the first *Transport* step, a *Mini Load Conveyor* automatically brings small materials from storage locations to different workstations, where *Prepare* activities are performed.
- In the second *Transport* step, a *Mini Load Conveyor* automatically transports pieces that are smaller than 50X50X25(cm); a *Unit Load Conveyor* automatically transports pieces that are bigger than 50X50X25(cm); human workers manually transport specific components to certain locations for assembly purposes with a *Counterbalance Truck* (e.g., car tires are manually transported to the end of whole assembly line).
- In the *Assemble* step, a *Transport* activity is also required. The car frame as the biggest piece of the production process is carried by a *Overhead Conveyor* to support other assembly tasks.

Figure 2 illustrates a more detailed analysis of the *Transport* activity. It presents four behavioral variants of the *Transport* activity, that we identified from this car production example:

1) Within the second transport step, components are possibly transported with three different kinds of transporters, depending on their types.
2) At several points, different transport tasks need to be synchronized to gather all the necessary components for further integration.
3) To fulfill one transport task, several different sub-transport activities have to be connected subsequently.
4) The transport task by the overhead conveyor has to be stopped every certain period of time, to let other assembly activities finish.

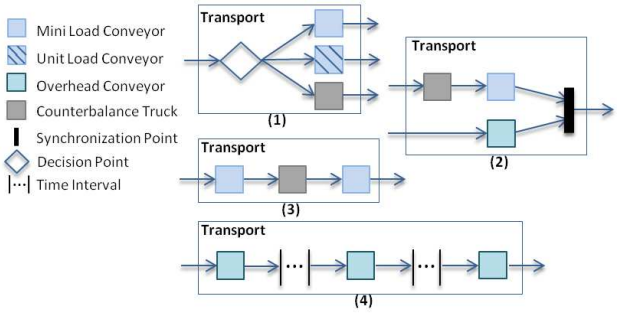

Fig. 2. Variants of Transport Activity

These four variants shown in Figure 2 are identified from our example, but it does not mean that the behavioral variants are limited to only these four. It is also possible to generalize them and find other behavioral variants in other sub-processes and activities, where several repeated functions can be selected, configured, and combined to perform more sophisticated tasks.

On the one hand, traditional variability modeling approaches, such as feature modeling, are helpful for understanding the structure of variation points and variants, as illustrated in Figure 3. This figure shows a generic feature model of transporters, which is not limited to car production. The three kinds of automatic transporters are: *Mini Load Conveyor*, *Unit Load Conveyor*, and *Overhead Conveyor*. The two kinds of manual transporters are *Counterbalance Truck* and *Side load Truck*. They are driven by human workers in the manufacturing factory, and are used for taking and moving materials from one location to another. The *Side load Truck*, for example, is used to reach some special locations. The software-related elements, such as controllers and user interfaces (UI) are in blue color, to differentiate them from the controlled devices.

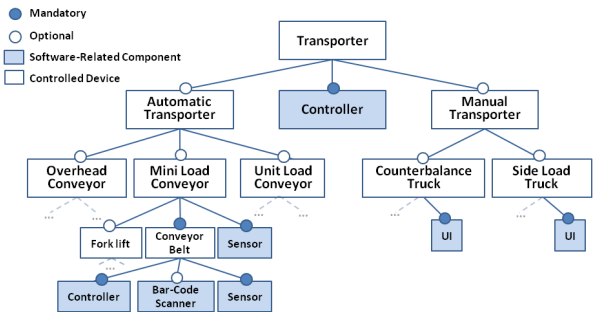

Fig. 3. A Transporter Feature Model

On the other hand, however, the behavioral variants identified in Figure 2 are not easy to be presented with feature modeling, neither with other variability approaches sharing the similar form. Figure 3 also gives an indication of the hierarchy of the different elements, which the production control software requires to coordinate and communicate with. The instances of the same elements from feature modeling behave differently and participate in different activities. Thus, we expect to involve behavior variability to help on enhancing the expressiveness in this respect. There are existing works that provide candidate solutions to address behavior variability. To apply them, we identify some unsolved issues. We present them in section III.

III. Challenge Analysis for Managing Behavior Variability

The ultimate aim for modeling and managing behavior variability is to use the models as a "vehicle" to facilitate the development of production control software. This section analyzes the challenges of managing behavior variability in two aspects: 1) representation and 2) implementation. The representation of variability concerns the identification of behavioral commonalities and variability, as well as explicitly representing them with models. The variability implementation concerns the mechanisms to resolve variation points based on behavior variability. It requires the consideration in the product line infrastructure and influences to software architecture.

A. Representation of Behavior Variability

To the best of our knowledge, there are mainly three groups of existing approaches that address behavior variability. We categorize them as follows: UML-based approaches, business-process-based approaches, and Domain Specific Languages (DSLs):

- Halmans and Pohl propose extensions to use case diagrams to explicitly mark functional variants for communicating variability to customers [5]. Brown *et al.* propose an approach in the embedded software system domain, that links behavior to features in bi-directional feature modeling. In this approach, they use Use Case Maps (UCM) path notation to capture the behavioral commonality and variability in both structural and behavioral aspects [3]. Another extension based on the activity diagram is suggested by Razavian and Khosravi [9] in the domain of E-Business processes. In this approach, several stereotypes are defined and added to UML activity diagrams to express behavior variability.
- The second group of approaches is originally from business process management. In this group there are approaches based on Business Process Modeling Notations (BPMN) [12], or Event-driven Process Chain (EPC) [7].
- Völter classifies three kinds of variability: *configuration* variability refers to the selection from several variant alternatives; *construction* variability requires a well-defined language which can be designed to be able to define an unlimited number of variants; *combination* variability combines the former two kinds [15]. In the context of production control software, we understand that proper designed DSLs might be a solution to model the unlimited combinations of behavioral variants [14].

The challenge of modeling flexible combinations of activities: Most of these behavioral variability modeling approaches come up with new models by making decisions on pre-identified variation points and variants, which means that the configured models have the same or less execution traces than the original model. For production processes, the combinations of activities are not fixed until the application engineering phase. In the existing literature, the approach of Schnieders and Puhlmann [12] provides a chance to enrich the execution traces. The work flow patterns summarized by Russel *et al.* [10] might also help on finding the possible patterns of different combinations. Since both of them are proposed in the business process domain, whether they can be applied in production domain still requires more investigation. The DSLs approach proposed by Völter is a promising approach to express the flexibility of behavior variability. But the effort and difficulty of designing such a language or several DSLs to describe behavior variability in production control software still needs to be investigated and requires further study.

The challenge of granularity: Behavioral models can be used to model the software system in different abstraction levels, from a global business layer as maybe the top most layer, a product layer, an application layer, an individual component layer, and even code as the most detailed layer. Consequently, the variability exists in all these different abstraction layers. Due to the flexibility in each layer, it may not be necessary to cover all behavior variability in all layers. Thus, the question raises as how to scope a correct set of variability to model and support the product derivation, in order to help developers on their development work.

B. Implementation of Behavior Variability

In [5], Halmans and Pohl classify variability into essential and technical variability. Whereas essential variability adopts a customers' usage perspective, technical variability is related to the implementation of the variability and the influence of the IT-infrastructure. However, behavior variability is one type of essential variability that can significantly influence the IT-infrastructure. The reason is that configurable behavior modeling demands more flexible variability mechanisms in the product line infrastructure. To decide on the required variability mechanisms, it is important to determine when and how to bind variants for each variation point, and decide an appropriate variability strategy for each of them [13].

Two characteristics of the domain add to the difficulty of addressing behavior variability with a product line approach. Firstly, the control of manufacturing processes requires to communicate with the real-world material flow. For example, there are tasks done by human workers, which do not involve any software or hardware, but need to be synchronized by the control systems. Secondly, production control software has to process a huge amount of messages from different sensors,

automation controllers and electronic devices, as indicated in the hierarchy of Figure 3. Thousands of configuration options all need to be configured and aligned to each other to perform production tasks. There is a high potential of benefits by reducing the effort for configuration by adopting a product line engineering approach in this domain.

The challenge to resolve binding: The decisions of variants in production control systems are distributed in different components and artifacts. Some of them may be stored in a database, some others are in configuration files or even in the code of individual components. This is because of the highly distributed structure (hardware, sensors, electronics) of the manufacturing plants. The behavior variability models are relatively centralized, as is shown at top of Figure 4. To resolve the binding of behavioral variants, the configured behavioral variants need to establish the mappings to databases, configuration files, and some feature models in lower layers. This creates the difficulty to choose a single strategy and single binding time to resolve different variation points. Considering this difficulty, a combined variability implementation strategy might be necessary to be applied [15].

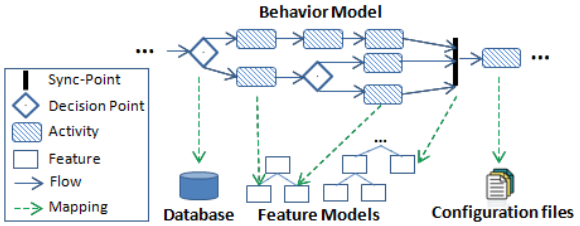

Fig. 4. The Challenge to Resolve Binding

IV. Conclusion

In this paper, we have analyzed the behavior variability in production control systems by means of an example from car production processes. The example shows the insufficiency of using traditional structure-based variability modeling to express behavior variability. We have identified three main challenges in representing and implementing behavior variability, for which we are looking for possible solutions:

- Existing approaches, such as DSLs, are able to model complex behavior variability. We are looking for approaches that ease the design of such languages to model behavior variability, for example, by reuse of behavioral patterns or by leveraging experience in applying DSLs in similar domains.
- Since behavioral variability exists in different levels of abstraction, we seek criteria that help us to define and scope the proper granularity for modeling the behavior variability at different abstraction levels, and methods to further link them together to support product derivation.
- It is difficult to bind the variants configured in behavioral models to the distributed components with a single strategy, due to the highly distributed structure of production systems. We search for possible solutions that can guide us to establish the binding between the behavior models and the lower-level structural configurations of the involved software-related components in a systematic and maintainable way.

Managing behavior variability is essential for organizations in our target domain to achieve more systematic reuse and adopt a software product line approach. For the representation aspect of behavior variability, a key measure of candidate approaches would be to evaluate whether they can help developers to reduce the time on specifying the behavior of systems. For the implementation aspect, we plan to evaluate the improvement on the design of the variability configurator and the code generator within SPL infrastructure to support the flexible requirement of behavior variability. Finally, as the challenges of managing behavior variability also appear in other process-oriented and message-driven systems, we see the potential to generalize our aspired solutions to further domains.

References

[1] S. Adam, J. Doerr, M. Eisenbarth, and A. Gross. Using task-oriented requirements engineering in different domains–experiences with application in research and industry. In *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*, pages 267–272. IEEE, 2009.

[2] J. Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. *Software Product Lines*, pages 247–262, 2002.

[3] T. Brown, R. Gawley, R. Bashroush, I. Spence, P. Kilpatrick, and C. Gillan. Weaving behavior into feature models for embedded system families. In *Software Product Line Conference, 2006 10th International*, pages 52–61. IEEE, 2006.

[4] L. Chen, M. Ali Babar, and N. Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference*, pages 81–90. Carnegie Mellon University, 2009.

[5] G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Software and Systems Modeling*, 2(1):15–36, 2003.

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990.

[7] M. La Rosa, F. Gottschalk, M. Dumas, and W. van der Aalst. Linking domain models and process models for reference model configuration. In *Business Process Management Workshops*, pages 417–430. Springer, 2008.

[8] OMG. OMG Unified Modeling Language TM (OMG UML), superstructure, 2011.

[9] M. Razavian and R. Khosravi. Modeling variability in business process models using uml. In *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*, pages 82–87. IEEE, 2008.

[10] N. Russell, A. Ter Hofstede, and N. Mulyar. Workflow controlflow patterns: A revised view. 2006.

[11] K. Schmid and I. John. A customizable approach to full lifecycle variability management. *Science of Computer Programming*, 53(3):259–284, 2004.

[12] A. Schnieders and F. Puhlmann. Variability mechanisms in e-business process families. In *9th International Conference on Business Information Systems (BIS 2006)*, volume 85, pages 583–601, 2006.

[13] J. Van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, pages 45–54. IEEE, 2001.

[14] M. Voelter and E. Visser. Product line engineering using domain-specific languages. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 70–79. IEEE, 2011.

[15] M. Völter. Handling variability. Hillside Europe for inclusion in the CEUR archive of conference proceedings and for Hillside Europe website, 2009.