Chapter 16
Dynamic Software Product Lines

Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid

What you will learn in this chapter
- *The importance of dynamic software product lines.*
- *The role of software product lines in adaptive systems.*
- *The underpinnings of dynamic software product lines.*

1 Introduction[1]

In emerging domains such as ubiquitous computing, service robotics, unmanned space and water exploration, and medical and life-support devices, software is becoming increasingly complex with extensive variation in both requirements and resource constraints throughout its lifetime. This is partly due to the dynamic nature of modern computing and network environments and the way they are used and partly due to the need to survive evolution of the same. For example, computing

[1] This chapter is partially based on a previous article by the authors (cf. [4]).

S. Hallsteinsen
SINTEF ICT, Trondheim, Norway
e-mail: svein.hallsteinsen@sintef.no

M. Hinchey (✉)
Lero-The Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland
e-mail: mike.hinchey@lero.ie

S. Park
Sogang University, Mapo-gu, Seoul, South Republic of Korea
e-mail: sypark@sogang.ac.kr

K. Schmid
University of Hildesheim, Hildesheim, Germany
e-mail: schmid@sse.uni-hildesheim.de

and communication resources, user requirements, and interface mechanisms between software and hardware devices such as sensors can change dynamically at runtime. As a result, a higher degree of adaptability is demanded from software systems. At the same time, developers face growing pressure to deliver high-quality software with extensive functionality, on tight deadlines, more economically.

This challenge is by now widely recognized and has led to a significant number of different efforts to achieve (self-)adaptation properties in software systems. Examples for this are self-adapting systems [1], autonomous systems [2], agent-based systems [3], reflective middleware, emergent systems, etc. All these approaches share a common underlying goal: to make software systems more flexible than ever before. In this chapter aim at a discussing a particular subclass of approaches addressing this goal, which borrows essential ideas from product line engineering to achieve flexibility at runtime and, hence, is called *dynamic software product lines* [4].

Software product lines (SPL) have been successful in coping with varying requirements by allowing the derivation of product variants from a common asset base. However, to cope with the challenges discussed above, *dynamic* SPLs (DSPL) aim at producing software capable of adapting both to fluctuations and evolution in user needs and resource constraints at runtime. DSPLs may bind variation points initially when software is launched to adapt to the current environment as well as during operation to adapt to changes in the environment.

Although traditional SPL engineering recognizes that variation points are bound at different stages of development, and possibly also at runtime, the main body of research focuses on binding variation points at the latest at system startup. Traditional approaches to software product line engineering focus in their methods and techniques on this scenario. In contrast, DSPL engineers typically aren't concerned with pre-runtime variation points. However, they recognize that in practice mixed approaches might be viable, where some variation points related to the environment's static properties are bound before runtime and others related to the dynamic properties are bound at runtime [4].

1.1 Product Lines

Henry Ford (1863–1947), founder of the Ford Motor Company, is often viewed as the father of factory automation and the use of assembly lines, which he introduced and expanded in his factories between 1908 and 1913 in building his Model T line of motorcars.

Ford is famously quoted as saying that "Any customer can have a car painted any colour [sic] that he wants so long as it is black" [5]. He is noted for his introduction of mass production and the assembly line. What is less known is that this was achieved through the use of interchangeable parts, based on earlier ideas by Honor Blanc and Eli Whitney, which significantly streamlined the process over previous approaches where parts were often incompatible and one difference in a product meant that the entire development process had to be restarted. The result was economies of scale and a line of motor cars that were affordable, built quickly, and to a high quality standard, even if the choice of colors, etc., was extremely limited.

Ford's ideas have been influential in the development of the idea of using product lines sometimes called Product Family Engineering or Product Line Engineering (PLE), which affords economies of scope. As Greenfield et al. point out: "Economies of scale arise when multiple identical instances of a single design are produced collectively, rather than individually. Economies of scope arise when multiple similar but distinct designs and prototypes are produced collectively, rather than individually" [6]. Economies of scope imply mass customization. Mass customization is defined as "producing goods and services to meet individual customer's needs with near mass production efficiency" [7].

PLE provides an alternative to mass production in order to create customized products as similar variants of the mass produced products; that is, reusing core assets (a "platform") to develop similar products for a market segment. Its key aim is to create an underlying architecture of an organization's product platform, based on commonality and variation. Product variants can then be derived from the basic product family, reusing common components and using various combinations to create a variety of products.

The use of product lines involves engineering new products in such a way that it is possible to predictably reuse product components and offer variability and choice while simultaneously decreasing costs and development lead time. The software development community has caught on to the usefulness of the approach with the idea of Software Product Lines.

1.2 Software Product Lines

The Software Engineering Institute defines a Software Product Line (SPL) as "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [8]. The approach has been successfully used to develop a wide variety of product lines in a number of different domains, ranging from avionics over medical equipment to information systems, in a wide variety of organizations, ranging from five developers to more than a thousand.

Consistently, strong achievements in terms of time-to-market, cost reduction, and quality improvement have been achieved. The interested reader is directed to the Product Line Hall of Fame [9]. In-depth discussions of product lines case studies are given in many literature sources like [10].

Fundamental to product line engineering is a shift from a single system point-of-view to an integrated understanding of a set of products. As a result, the differences or variations among products become a primary concern. Thus management of variation—so-called variability management—is a core capacity of PLE. The idea

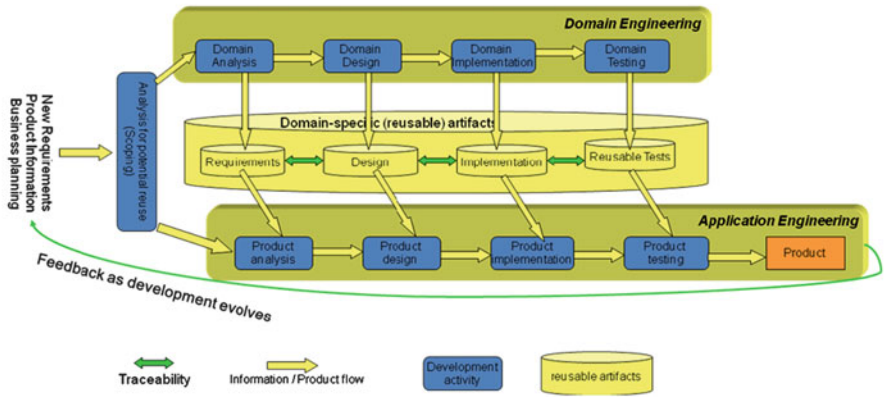

Fig. 16.1 The product line engineering process

is to separate all products in the product line into three parts and to manage them throughout development:

- *Commonality*: artifacts, which are common to all products in the product line. On the requirements side these are common requirements; in the implementation this results in common components.
- *Reusable variation*: aspects that are common to some, but not all, products in the product line. This is the powerhouse of product line engineering; by providing a low-effort mechanism with predictable properties, it is easy to assemble new products by reusing existing assets.
- *Product specifics*: no matter how well designed a product line is, there will always be requirements that are specific to individual products. Here, it is key not to waste any effort on generic development for aspects that will be used only once.

In addition to variability management, a second key principle of product line engineering is the use of a twin-lifecycle approach. We separate development into domain engineering and application engineering (cf. Fig. 16.1). Domain engineering is responsible for an analysis of the product line as a whole and for producing any common and (reusable) variable parts. Application engineering is then responsible for, for all aspects peculiar to individual products, such as the derivation of product requirements, the selection and integration of reusable parts, and possibly the production and integration of product-specific parts. Both life cycles can rely on fundamentally different processes (e.g., agile application engineering combined with plan-driven domain engineering).

The Software Product Line approach has received increased attention as a means of coping with product diversity, especially as software engineers and developers are faced with increasing pressure to deliver high-quality software with more functionality on strict release deadlines ever more economically.

Traditional product line engineering supports the binding of variability (i.e., the selection of the variations to be exhibited in a specific product) at different points in time: the so-called *binding time*. However, the focus was always on development time binding, ranging from the design stage to the moment of initialization of the system, with a strong focus on preprocessing, compiling, and linking. In the light of the need for a higher degree of adaptability, the need arose to provide variation also at runtime. This is the main characteristic of a system that qualifies as a DSPL, as we will further discuss in the next section.

2 DSPL

Dynamic Software Product Lines (DSPL) may be viewed as an area of research where we try to apply ideas developed in the SPL community like variability modeling, common assets shared by product variants, and automated product derivation to build software that adapts dynamically to fluctuations in user needs, environmental conditions, and resource constraints at runtime [4].

Relying on product line ideas and transferring them to the realm of runtime adaptation is one approach to build self-managed systems, i.e., software systems, which can modify their own behavior with respect to changes in their operating environment and thus adapt at runtime to the changing environment. Such systems are becoming more essential for complex network management, for use in unmanned space and underwater exploration, for complex medical and life support devices.

The central shift from the traditional view of software product lines to dynamic software product lines is that variation points are bound at runtime. First, when software is launched to adapt it to the current situation and subsequently to re-bind variation during operation to adapt the software to changes in the situation.

Thus, DSPL is basically not concerned with pre-runtime variation points. However, it recognizes that in practice mixed approaches may be viable, where some variation points related to static properties of the environment of use are bound before runtime, while others related to the dynamic properties are bound at runtime. Examples of approaches that seek to unify pre-runtime and runtime variation points into the same development framework are EASy-Producer [11] or CAPucine [12]. Both use aspect weaving as their primary mechanism for variability.

If a dynamic software product line is only variable at runtime, it will be perceived as a single adaptive system. On the other hand, if also development time binding is involved, it will still be perceived as a product line, where some or all products are adaptive systems. Some approaches support such a combination of binding times, even for the same variability, thus avoiding the need to duplicate the implementation of features [11, 13, 14].

Transferring product line concepts to the dynamic situation mostly focuses on the aspect of variability management. Variability management is on the one hand responsible to model the variability that is supported by the product line and on the other hand responsible for mapping this to the level of implementation, i.e., to determine what implementation impact a certain variability has. This can be readily transferred to the level of runtime binding by introducing a model that makes the potential variation explicit and supports the mapping to implementation consequences *at runtime*. Along the lines of traditional product lines, this model is a discrete model, mapping out certain alternatives and optional characteristics, and relating them to implementation consequences, just as existing approaches to product line variability do. Due to its importance, we regard the existence of such an explicit variability model as a defining piece for a DSPL. Actually, we will refer to any approach that relies on variation management at runtime for dynamic adaptation as a DSPL.

In dynamic software product lines, monitoring the current situation and controlling the adaptation are central tasks. This may be performed manually by an operator or by the user, or automatically performed by the application or by generic middleware. In this aspect, the DSPL approach is less restrictive than other approaches to adaptive systems like autonomous computing [2] or self-adaptive systems [1], etc., which demand that the system is able to autonomously perform the data gathering and decision making. Another difference is that the models that guide these variabilities are akin to product line variability models, although evaluated at runtime. This excludes certain ways of dealing with adaptation. As a DSPL we explicitly do not take into account cases where the whole system needs to be halted and restarted in order to perform an adaptation, no matter how flexible it seems (like [15]).

Evolution has been a concern in traditional SPL research. It is recognized that it is in general impossible to foresee all functionality or variability that will be needed in an SPL up-front, so evolution must be expected. In application areas such as those mentioned above it becomes more and more important to be able to evolve both the functionality and the adaptation capabilities of deployed systems. Here, DSPL can help. On a first level a DSPL is able to adapt to varying circumstances at runtime. Thus, it can, for example, address evolution needs that are created by changes in the environment. However, despite these capabilities the basic range of variability, which is supported typically remains fixed. However, approaches that combine DSPL with open variability (i.e., variability is identified, but dynamically more alternatives of realizing a variability can be added) may have an answer, which enables to dynamically extend the scope of the DSPL at runtime. The details of how to do this while providing certain correctness guarantees are still to date a challenging research problem.

It is worth noting that although Dynamic Software Product Lines are built on central ideas of SPL, there are also differences. For example, the focus on understanding the market and let the market drive the variability analysis may not be so relevant to DSPL, where concern is about adaptation to variations in individual needs and situations rather than market forces. Also reconfiguring a running product has additional challenges compared to configuring a new product instance, like pausing the execution and transferring state.

Dynamic Software Product Lines is an emerging and promising area of research with clear overlaps to other areas of research besides SPL, notably:

- Research related to self-* (adapting/managing/healing...) systems and autonomic computing is also concerned with situation monitoring and adaptation decision making [1], and DSPL may be seen as one among several approaches to building such systems
- Agent-based software engineering [3] and multi-agent systems, which focus on the use of agents and organizations (communities) of agents as the main abstractions

DSPL brings to this arena a number of techniques, which have proven very successful in a different, but strongly related field. This supports the expectation that these techniques can also contribute to the general research direction of adaptive and runtime-flexible systems.

3 Conclusion

We have presented the notion of *dynamic software product line (DSPL)*. In summary, to qualify as a DSPL we envision a system, which was developed based on (some) SPL principles and has the following properties:

- Dynamic (runtime) variability: (re-)configuration and binding happen at runtime, changing the binding several times during the lifetime of the system
- Dealing with changes in the environment or triggered by users (e.g., changes in functional and/or quality requirements)
- The system can be seen as a product line where the running system switches at runtime from one variant to another
- It uses a variability management approach for identifying the different possible runtime variants
- An integrated model of the various runtime variations of the system exists (comparable to a domain model for a design-time product line)
- An architecture exists that describes the architectural variation that may happen (comparable to a reference architecture in classical PLE)

DSPL is suited to develop systems, which are adaptable at runtime by manual intervention, as well as autonomic or self-adaptive systems. In the latter case it is also necessary to address context or situation awareness and automatic adaptation reasoning and decision-making.

At this point there exists a small, but growing community, which aims to take the DSPL way to transfer existing knowledge and solutions to the realm of adaptive systems. First surveys exist that capture the state of the field [16, 17], which is at this point still rather fragmented. One of the most important challenges at this point is probably to address open runtime variability, i.e., variations that are not foreseen at development time, but need to be integrated at runtime. This might provide a pathway for dynamic evolution of product lines.

References

1. Cheng, B., Giese, H., Inverardi, P., Magee, J., de Lemos, R.: Software engineering for self-adaptive systems: a research road map. In: Software Engineering for Self-Adaptive Systems, Dagstuhl Seminar Proceedings, 08031, available at: http://drops.dagstuhl.de/opus/volltexte/2008/1500 (2008)
2. Kephart, J., Chess, D.: The vision of autonomic computing. IEEE Comput. **36**(1), 41–50 (2003)
3. Jennings, N.: On agent-based software engineering. Artif. Intell. **177**(2), 277–296 (2000)
4. Hallsteinsen, S., Hinchey, M., Park, S.Y., Schmid, K.: Dynamic software product lines. Computer **41**(4), 93–95 (2008)
5. Ford, H: My Life and Work – An Autobiography of Henry Ford. Heinemann, London (1923)
6. Greenfield, J., et al.: Software Factories, Assembling Applications with Patterns, Models Framework, and Tools. Wiley, Indianapolis, IN (2004)
7. Tseng, M.M., Jiao, J.: Mass customization, In: Handbook of Industrial Engineering, Technology and Operation Management. Wiley, New York (2001)
8. Software Engineering Institute. Software product lines – overview. Online available at: http://www.sei.cmu.edu/productlines (last checked: 7.4.12)
9. The product line hall of fame. Online available at: http://splc.net/fame.html (checked: 7.4.12)
10. van der Linden, F., Schmid, K., Rommes, E.: Software Product Lines in Action – The Best Industrial Practice in Product Line Engineering. Springer, Heidelberg (2007)
11. Schmid, K., Eichelberger, H.: Model-based implementation of meta-variability constructs: a case study using aspects. In: Second International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS), ICB-Research Report No. 22, ISSN 1860-2770, pp. 63–71 (2008)
12. Parra, C., Blanc, X., Cleve, A., Duchien, L.: Unifying design and runtime software adaptation using aspect models. Sci. Comput. Program. **76**(12), 1247–1260 (2011)
13. van der Hoek, A.: Design-time product line architectures for any-time variability. Sci. Comput. Program. **53**(30), 285–304 (2004)
14. Dolstra, E., Florijn, G., de Jonge, M., Visser, E.: Capturing timeline variability with transparent configuration environments. In: International Workshop on Software Variability Management. ICSE Workshop (2003)
15. White, J., Schmidt, D.C., Wuchner, E., Nechypurenko, A.: Automating product-line variant selection for mobile devices. In: Software Product Line Conference (SPLC), pp. 129–140 (2007)
16. Bencomo, N., Lee, J., Hallsteinsen, S.: How dynamic is your dynamic software product line? In: Workshop on Dynamic Software Product Lines, pp. 61–68 (2010)
17. Burégio, V., de Lemos Meira, S., de Almeida, E.: Characterizing dynamic software product lines – a preliminary mapping study. In: Workshop on Dynamic Software Product Lines, pp. 53–60 (2010)