Variability and Evolution in Systems of Systems

Goetz Botterweck
Lero–The Irish Software Engineering Research Centre,
Limerick, Ireland
goetz.botterweck@lero.ie

In this position paper (1) we discuss two particular aspects of Systems of Systems, i.e., *variability* and *evolution*. (2) We argue that concepts from Product Line Engineering and Software Evolution are relevant to Systems of Systems Engineering. (3) Conversely, concepts from Systems of Systems Engineering can be helpful in Product Line Engineering and Software Evolution. Hence, we argue that an exchange of concepts between the disciplines would be beneficial.

1 Introduction

In this position paper we (1) discuss two particular aspects of Systems of Systems (SoS), i.e., *variability* and *evolution*. We do this from two perspectives: (2) First, we argue that in order to address variability and evolution in the context of Systems of Systems Engineering (SoSE), concepts from Product Line Engineering (PLE) and Software Evolution are relevant and helpful. (3) Second, we observe that with increasing maturity of the disciplines the "objects of engineering" in PLE and Software Evolution become larger and larger. Consequently, such disciplines have to deal with SoS challenges. Hence, in this paper, we suggest a more lively exchange of concepts between the disciplines. We are aware that this paper mostly raises questions and does not provide a lot of answers. However, we strongly believe that such an exchange would be interesting and beneficial for both communities.

As an running example for the further discussion consider the scenario of an airport, schematically illustrated in Figure 1. Vertically, we distinguish *subsystems* like Planes, Air Traffic Control System, Baggage Handling System, Transportation Infrastructure, etc. Please note that the vertical presentation serves only to distinguish the subsystems and is *not* intended to indicate a layered model like for instance used in networking architectures.

Each subsystem consists of *elements* which can be in turn systems in their own right [23]. This hierarchical compositional structure can occur recursively over many levels. For instance, the Planes subsystem consists of planes; a plane consists of cockpit, navigation subsystem, power subsystem etc.; the power subsystem consists of engines, fuel supply, etc. Elements are connected to each other, either within the subsystems (solid lines) or across subsystem boundaries (dashed lines).

In such systems we have to deal with *variability* and *evolution*. First, in many cases it is beneficial to consider multiple elements together (e.g., when designing and implementing them) rather than addressing each element individually [26]. For instance, when creating and maintaining the communication infrastructure subsystem and its elements, it is beneficial to consider *types* or *families* of network routers rather than discussing each router individually. We then have to deal with variability and commonality among these elements. We will discuss this aspect in detail in Section 3.

Second, we have to consider how systems and their elements evolve over time. Long-term evolution is an inherent characteristic of Systems of Systems. Even though evolution in such large systems cannot be controlled by one particular party, we are nevertheless interested in systematic approaches to evolution. We will discuss this aspect of evolution in Section 4.

Larsen, Legay, Nyman (Eds.): 1st Workshop on
Advances in Systems of Systems (AiSoS 2013)
EPTCS 133, 2013, pp. 8–23, doi:10.4204/EPTCS.133.2

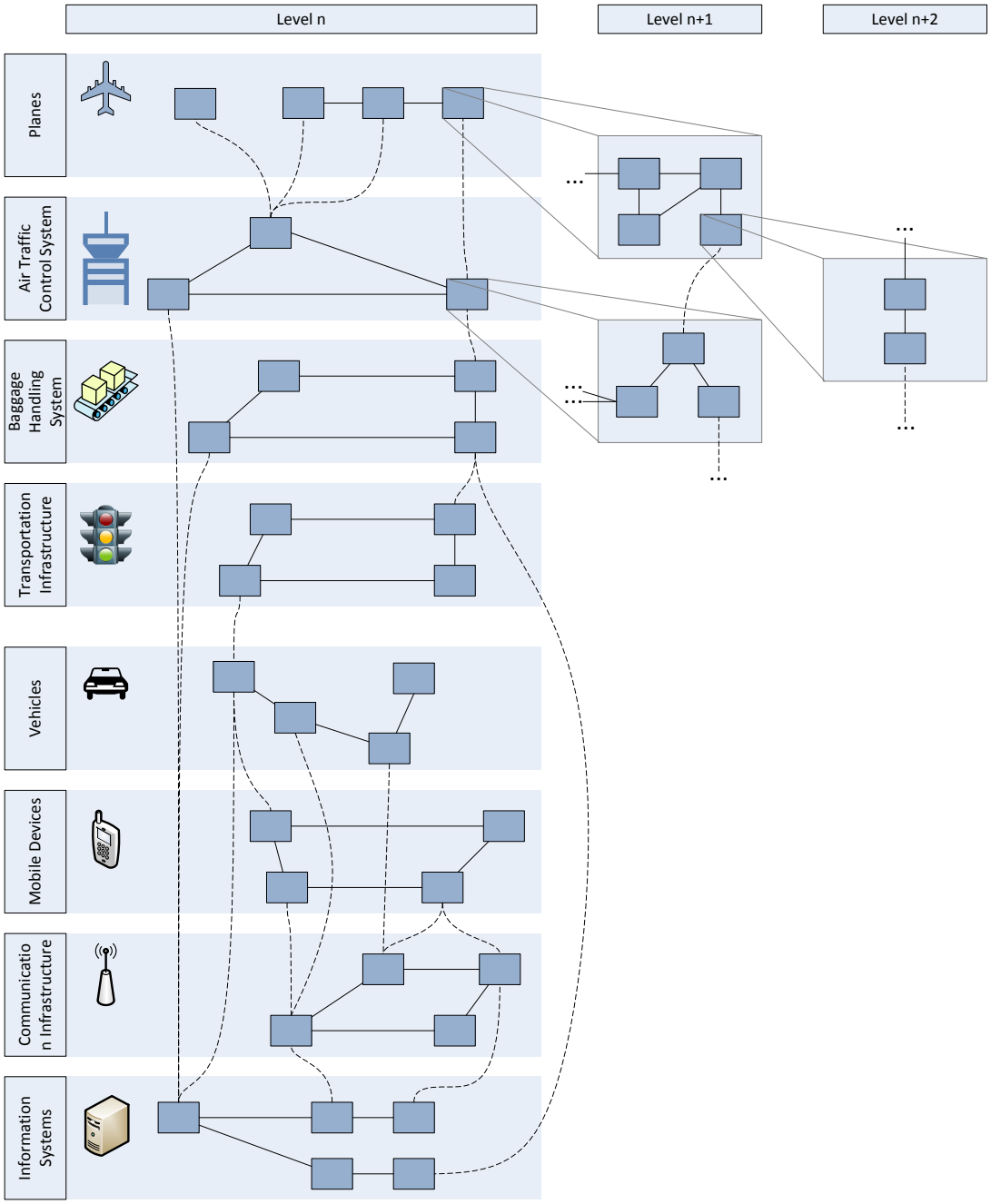

Figure 1: An airport as an example of a system of systems; including the recursive composition out of subsystems

We will now give a brief overview of background information (Section 2). Then, we will look at the aspects of variability and evolution in SoS (Sections 3 and 4). The paper concludes with final thoughts (Section 5).

2 Background

In this section we will give a brief summary of relevant background concepts from the disciplines Systems of Systems Engineering, Product Line Engineering, and Software Evolutions. Readers that are familiar with an area might skip over that particular section.

2.1 Systems of Systems Engineering

In this paper, we are discussing the relationship between Systems of Systems Engineering, Product Line Engineering, and Software Evolution. For instance, we are interested in how the inherent properties of SoS influence other engineering practices. Hence, let us first look at these defining properties of SoS.

Systems of Systems have been widely discussed in the literature (e.g., [23, 11, 32]). Even though no universally accepted definition exists, some **characteristics** have been described, e.g., by Maier [23]:

- Operational Independence of Elements
- Managerial Independence of Elements
- Evolutionary Development
- Emergent Behavior
- Geographical Distribution of Elements

In addition, we can observe the following phenomena which raise **additional challenges for engineering practices** (e.g., Systems Engineering, Software Engineering, Product Line Engineering) (based on [9, p.173–174]):

- Multiple stakeholders (for instance, related to the different subsystems) with varying, potentially conflicting interests
- High levels of technical complexity
- Large-scale, broad scope, long-term activity
- Change management and evolution management are relevant in many aspects and parts of the system
- Various constituent systems with independent life-cycles and lines of responsibility
- Requirement for adaptability, flexibility and open interfaces

Similar to Systems of Systems some authors discuss the concept of a *Federations of Systems* (e.g., [32]), which takes the idea even further, e.g., in terms of absence of a central authority. In this context, Krygiel [20] describes a hierarchical taxonomy of conventional systems, systems of systems (SoS) and federations of systems (FoS) .

2.2 Software Product Line Engineering

As a structured approach for variability management and systematic reuse, we will now look at the discipline of Software Product Line Engineering. Software Product Line Engineering [10, 29] commonly distinguishes two phases of engineering (Figure 2):

In *Domain Engineering* the product line is created (e.g., by establishing a scope and implementing shared assets). In *Domain Analysis* the scope of the product line and the allowed variations among its products are defined. To represent available variants configuration choices and to define how choosing

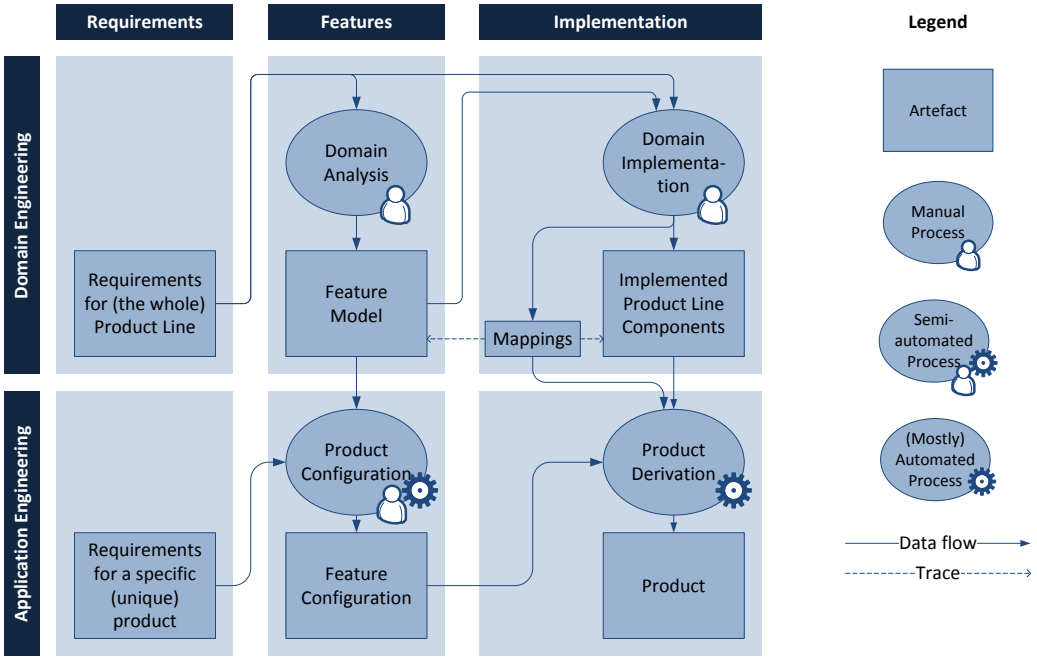

Figure 2: Software Product Line Engineering

particular options will influence the implementation-oriented artifacts, PLE approaches usually define some form of variability model. A popular group of approach are *Feature Models* (e.g., [19, 13]).[1] In *Domain Implementation* corresponding assets are created and mapped to the features to define how a particular configuration decision will influence the implementation.

In *Application Engineering* products are derived from the platform, which the product line provides. This process starts with the process of *Product Configuration* where the user is making configuration decisions based on the feature model, reducing the set of products, until exactly one product remains, represented as a *Feature Configuration*. This process can be supported by interactive tools (e.g., [3, 6]). In *Product Derivation* the corresponding implementation is created by composition and/or generative approaches. Ideally, this step can be mostly automated.

Product Line Engineering can be used on different levels of sophistication. Bosch [5] gives a taxonomy of product line approaches, including a discussion of their maturity. Of the discussed approaches, *Programme of Product Lines* goes somewhat into the direction of a System of Systems, but does not quite reach its complexity. Bosch speaks of very large systems with "a software architecture that is defined for the overall system and that specifies the components that make up the system. [..] The configuration of the components is performed [..] through product line-based product derivation".

In general, there is little discussion in the PLE literature of products as SoS, e.g., that they are composed of subsystems or part of a larger super-system. In particular, it is rarely considered how this hierarchical structure of systems should be reflected in the various PLE techniques. We will discuss this in more detail in Section 3 .

[1] Other techniques are *decision models (e.g., [33, 16])* or the *orthogonal variability model (OVM)* [29]. Surveys of variability modeling approaches can be found in [8, 14].

2.3 Software Evolution

Another relevant area related to the engineering of complex software-intensive systems is Software Evolution [25]. Traditionally, the literature in Software Evolution has mainly focused on analyzing **evolution "in hindsight"**, after it happened. We could also call this form of evolution *descriptive*, since it aims to describe how the evolution happened in the past. For instance, there are Lehman's Laws of Software Evolution [21, 22]:

- Continuing Change - a system must continually be adapted to its changing environment or it will become progressively less satisfactory
- Increasing Complexity - as a system evolves its complexity increases unless active work is undertaken to reduce it
- Self-regulation - the evolution process is self-regulating with close to normal distribution of measures of product and process attributes
- Organizational stability (invariant work rate) - the average effective global activity is invariant over the systems life time
- Conservation of Familiarity - the content of successive releases is statistically invariant
- Continuing Growth - the functional content of the system must continually be increased to maintain stakeholder satisfaction
- Declining Quality - the quality of the system will be (perceived as) declining unless rigorously maintained and adapted to a changing environment
- Feedback System - evolution processes are multi-level, multi-loop, multi-agent feedback systems and must be treated as such to be successfully improved

These "laws" have been observed and analyzed on numerous cases. Even though in the discussion of Lehman's laws, the term of "systems" is used, the SoS concept is really considered.

In contrast to the analysis of evolution "in the past", other works are focusing on **planned and managed evolution.** One could call this form of evolution *prescriptive*, since it aims to create and form a certain reality through evolution. Some authors propose apply modeling and model-driven techniques for this (e.g., [15]). In our earlier work we combine concepts from Product Line Engineering and Software Evolution to support the feature-oriented evolution of product lines [28, 34].

Again, the literature on Software Evolution rarely touches the SoS aspect. In the other direction, in Systems of Systems Engineering, however, the aspect of evolution is discussed as one of the challenges. We will look at this in more detail in Section 4.

3 Variability in Systems of Systems

In this section we will look at the interplay between Systems of Systems and Variability. Correspondingly, we will discuss relationships between concepts in Systems of Systems Engineering and Product Line Engineering.

3.1 Structures and their modeling

First of all, when dealing with Systems of Systems Engineering, we can consider each system to be potentially a product of a product line. For illustration purposes see the updated airport scenario in

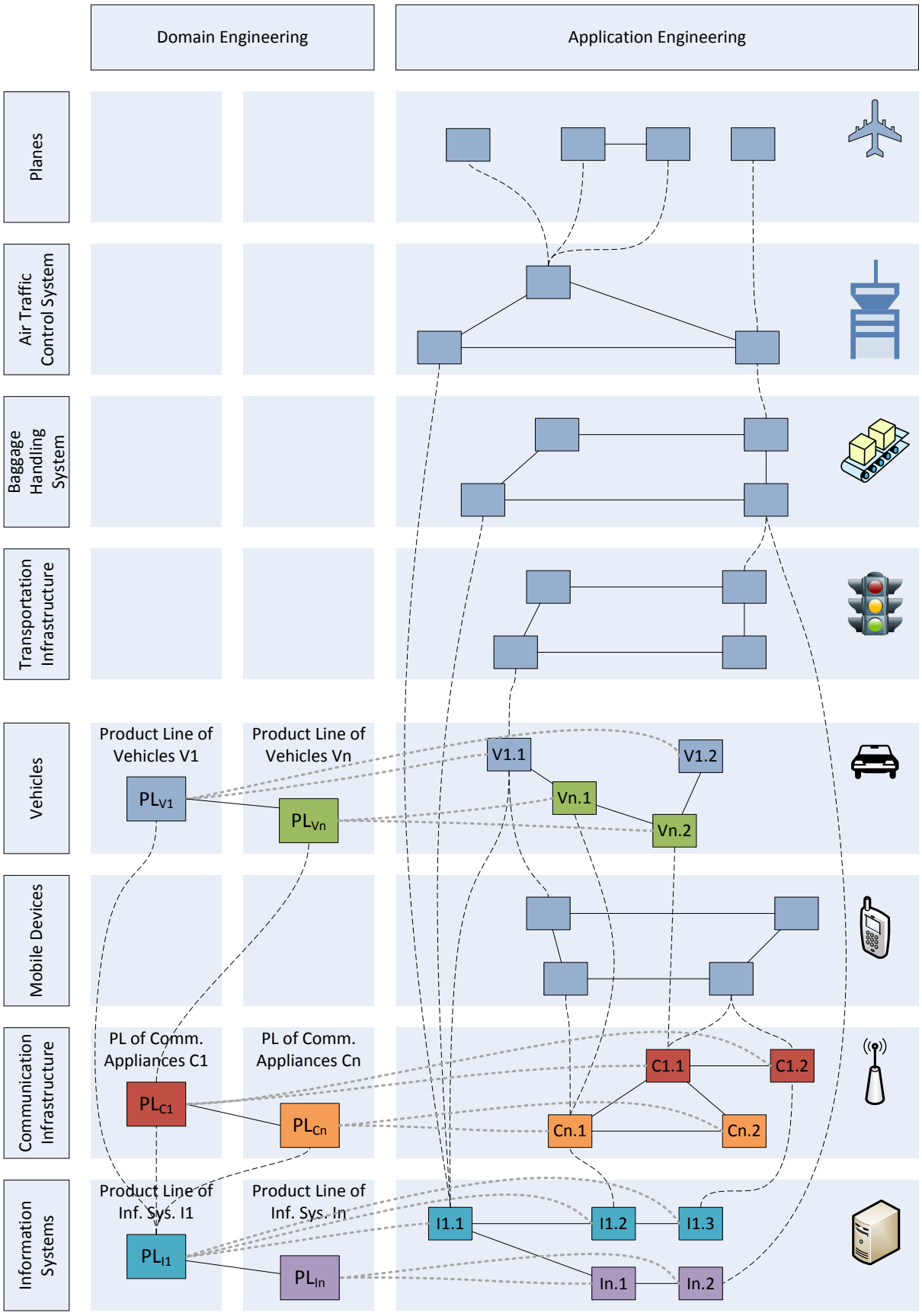

Figure 3: Again the airport as a Systems of Systems; here with concepts from Product Line Engineering and

Figure 3. It has been extended to show the distinction between product lines (Domain Engineering, left-hand side) and products (Application Engineering, right-hand side). This model is a combination of the airport scenario (Figure 1) and the PLE framework (Figure 2). Please note that we abstract from details by representing product lines as whole entities, not distinguishing various artifacts within the product line. Also here the distinction between Domain Engineering and Application Engineering is left-vs.-right, whereas in Figure 2 it was top-vs.-bottom. For simplification the diagram shows only product lines for three subsystems. We have for instance , $V1.1$ and $V1.2$ as products of the PL_{V1} product line of vehicles and $Vn.1$ and $Vn.2$ as products of the PL_{Vn} product line.

The motivation to consider systems (in a SoS context) to be products of a product line can come from multiple drivers. First, in many cases a *supplier* of systems (e.g., a manufacturer of network routers) will have families of similar systems and product line techniques promise considerable benefits in handling such families of products in a systematic fashion. So product lines can be seen as a technique to produce components in a SoS approach. Second, from the perspective of *user* of systems (e.g., the manager of the communication infrastructure of the airport) it can be beneficial to handle groups of systems together rather than addressing each system individually. For instance, the manager could use feature modeling to describe the variations of network appliances that are currently in operation in the airport.

As a first implication of the SoS context, we can observe that analogously to the co-existence and collaboration of multiple systems, we can expect a **co-existence of multiple product lines**. Also, structures between systems have to be reflected between product lines. For instance, in order to allow a communication link between the vehicle $Vn.2$ and the communication component $CI1.1$ the corresponding product lines PL_{Vn} and PL_{CI1} must be prepared to provide such products, e.g., by having corresponding implementations in their asset base.

Another aspect is that the **recursive hierarchical composition** of Systems of Systems can be applied to products as well. For instance, if we consider a plane as a product (which can be derived from a product line of planes) then the particular subsystems (e.g., cockpit displays and controls, navigation subsystem, power subsystem) can be considered products as well. We then have to consider various relationships, e.g., the plane *consisting* of cockpit subsystems etc., the cockpit subsystem *communicating with* the navigation subsystem, and so on. These relationships have to be reflected between product lines.

In the literature there are few approaches in this direction. For instance, Thompson and Heimdahl [38] discuss hierarchical product lines. A related aspect is the modularization of feature models, e.g., the multilevel feature trees suggested by Reiser and Weber [30].

3.2 Changes and challenges

We will now consider what consequences are induced by the Systems of Systems context for Product Line Engineering.

Product Line Engineering has certain assumptions. For instance, many PLE approaches assume that the domain and scope of the product line is relatively stable, such that the investment in the product line can pay off. Also, it is often implicitly assumed that there is *one* organization that controls how products are built, etc. We need to consider how these assumptions change when we move to a SoS context and what challenges arise from that.

3.2.1 Configuration.

In product configuration we can observe the following effects of a System of System context:

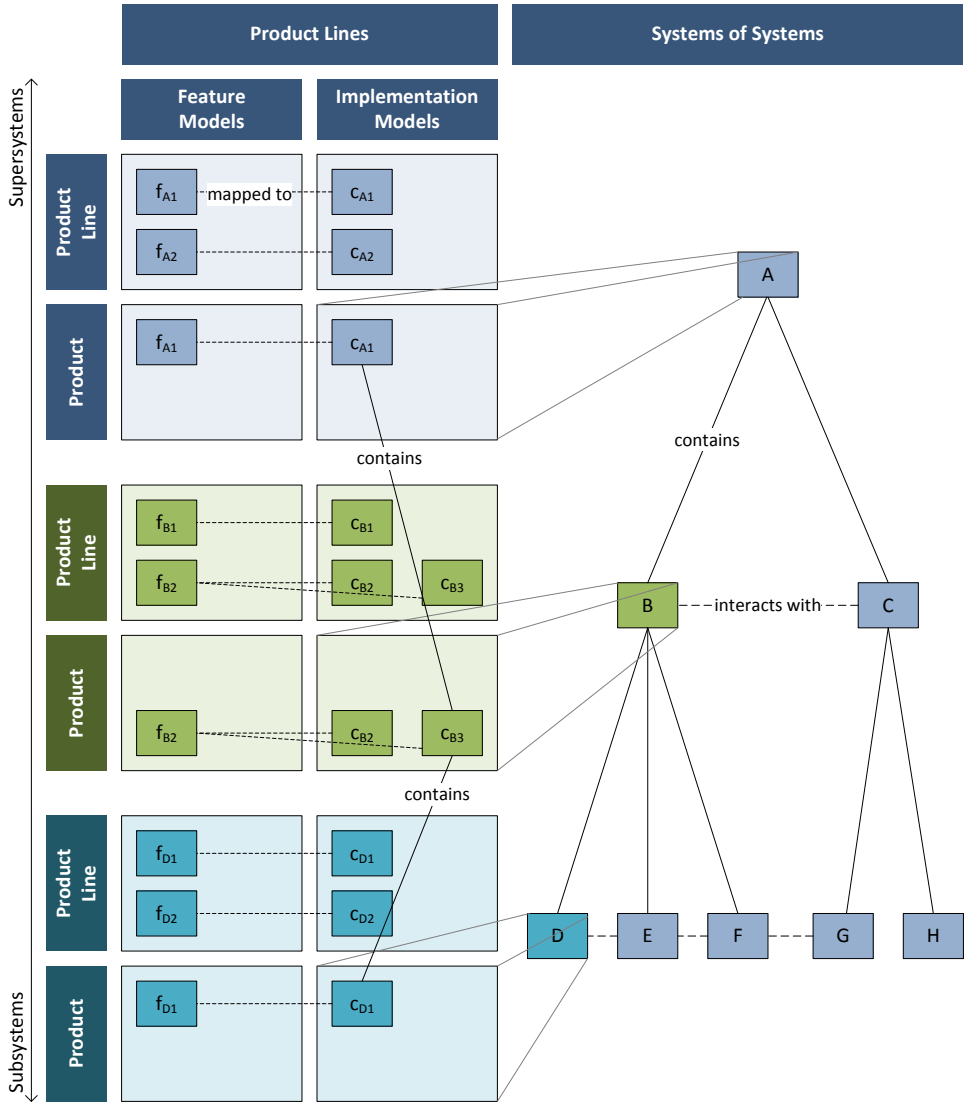

Figure 4: Modeling of product lines for systems of systems

First, the **hierarchical recursive structure** of the SoS needs to be reflected in the configuration process. For instance, when configuring a plane, we can consider that this will be part of an airport (actually multiple airports when traveling), and that it contains various subsystems. If we configure that the plane will have a certain size and wing design, then this influences how many engine we require and under which constraints (e.g., delivered thrust, fuel consumption) they must operate. Potentially, we have to propagate constraints. Depending on the scenario and the power to "dictate" conditions, this occur in various directions:

- *System to subsystem* - For instance, when the airport configuration determines the maximal size of planes that can operate on that airport.
- *System to supersystem* - For instance, when the large Airbus A380 "demands" that the airport gets extended (in its various subsystems) in order to attract the potential business.

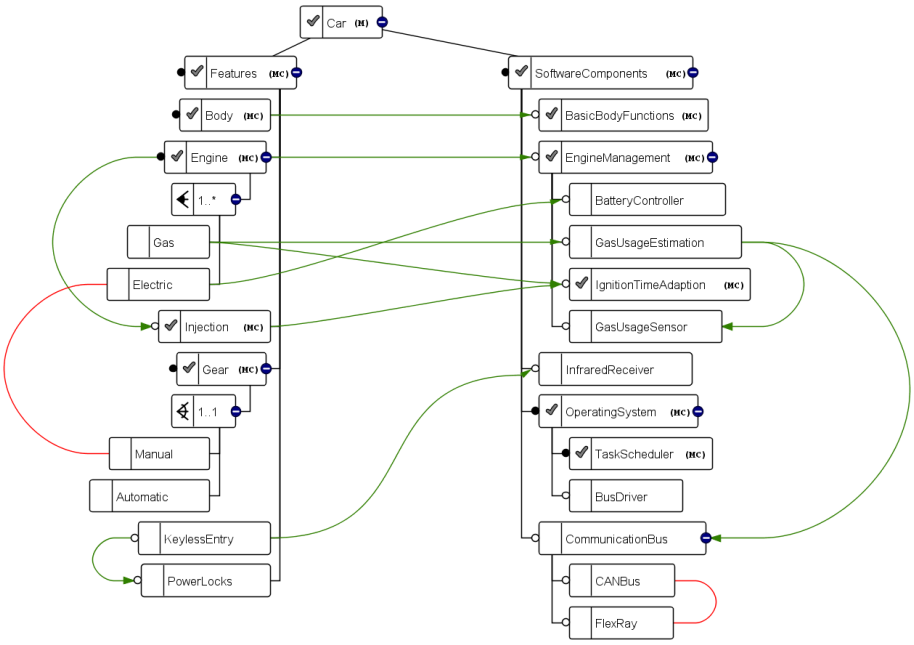

Figure 5: Configuring two related feature models side-by-side, taken from [27]

- *System to a neighboring system on a similar level* - For instance, when the Airbus A380 requires that new gates with enough passenger walkways are constructed. This can be considered as an effect of the preceding case.

Second, the **disappearance of a central control** and increased importance of multiple stakeholders with potential conflicting views needs to be reflected when configuring and deriving a product.

In the literature we find a some approaches that can be considered to help here. Czarnecki et al. [12] present their approach for Staged Configuration. This could be applied to make first major decisions (on supersystem level) and then later refine them (for subsystems). However, the approach does not provide concepts for structuring large models, e.g., by modularization.

Dhungana et al. [17] present an approach that deals large variability models that are created by multiple stakeholders, with fragmentation, the need to merge fragments, and to remove inconsistencies (e.g., variables that have been named differently by different stakeholders).

In earlier work [27], we have presented an approach for interactive configuration of feature models, including interaction techniques that allow to configure two related **feature models side-by-side**. See the example in Figure 5 with a feature model for a car on the left and the corresponding hardware components on the right. In a SoS context such a configuration tool could be used to configure two or more related systems side-by-side.

The interactive tool and its reasoning engine calculate and propagate consequences of configuration decisions. For instance, selecting the feature *KeylessEntry* (see bottom left) would cause the features *PowerLocks* and *InfraredReceiver* (in the other feature model part) to be selected as well. For the SoS discussion it should be noted that such approaches for modeling of constraints, reasoning and propagation of consequences often assume that the user who is performing the configuration (1) has precise **knowledge** about all involved systems and (2) the **power** to actually realize the chosen options. For instance, what does it help to choose a *KeylessEntry* function when we cannot ensure that the hardware will actually have an *InfraredReceiver*?

Dhungana et al. [18] present *invar,* an approach for the joint configuration of heterogeneous variability models, e.g., to configure a feature model, a decision model, and an OVM model side-by-side. Such techniques could be extended to support scenarios, where the "owners" of related systems use different variability modeling approaches.

For product configuration we remaining challenges:

- Representation and handling of very large models (both in terms of scale and complexity), including the reflection of the hierarchical, recursive compositional structure of SoS
- Dealing with incomplete information (because we might not get all information about the internals of a neighboring system to which we interface)
- Dealing with inconsistent information (because multiple stakeholders will have conflicting views and decisions)

3.2.2 Analysis.

There exist several approaches for the analysis of product lines (e.g., [1, 2]). Typical examples for available analyses is the enumeration of products, the detection of inconsistencies, or the detection of hidden dependencies (i.e., dependencies that exist but are not yet modeled explicitly).

When transitioning to an SoS context, analysis techniques face similar challenges as already discussed for configuration approaches, e.g., they have to deal with very large models, the SoS structure, and deal with incomplete as well as inconsistent information.

3.2.3 Product Properties.

Traditionally, product line approaches were often limited to Boolean concepts and decisions (e.g., a feature is selected or eliminated, f_1 requires f_2). Recently, there has been increased interest in product attributes and Non-functional Properties (NFP). This includes the prediction of properties based on a given feature configuration [35] , the configuration ("Give me a plane with top speed over 500 kph") and optimization [37, 39] ("Out of all planes that have my required features, give me the one with lowest price"). Often we have to deal with soft constraints and preferences which are often expressed as a utility function ("I want both low price and high top speed, but low prices is twice as important".)

It should be noted that here the PLE community often adapts and applies techniques that have been published earlier in other fields, e.g., in Artificial Intelligence or Constraints (e.g.,[31]).

Of particular interest in a SoS context is the article by Siegmund et al. [36], which deals with NFP in complex systems, e.g., a camera-based surveillance system, where the properties of various parts of the system influence each other (e.g., resolution of the camera vs. bandwidth of the network).

3.2.4 Software Architecture.

System of Systems engineering often deals with architectural frameworks (e.g., [7]) which are required to give structure and handle complexity, e.g., by defining how systems can be related to each other.

Product Line Engineering deals with Product Line Architectures (e.g., [4, 24]), where a greater emphasis is on variability and variability implementation. When transitioning to an SoS context, we have to reflect the structure of systems in the software architecture. For instance, corresponding to the hierarchical, recursive compositional structures we can have a nested hierarchy of product line architectures. The PLA of the supersystem is then refined by multiple PLAs of its subsystems.

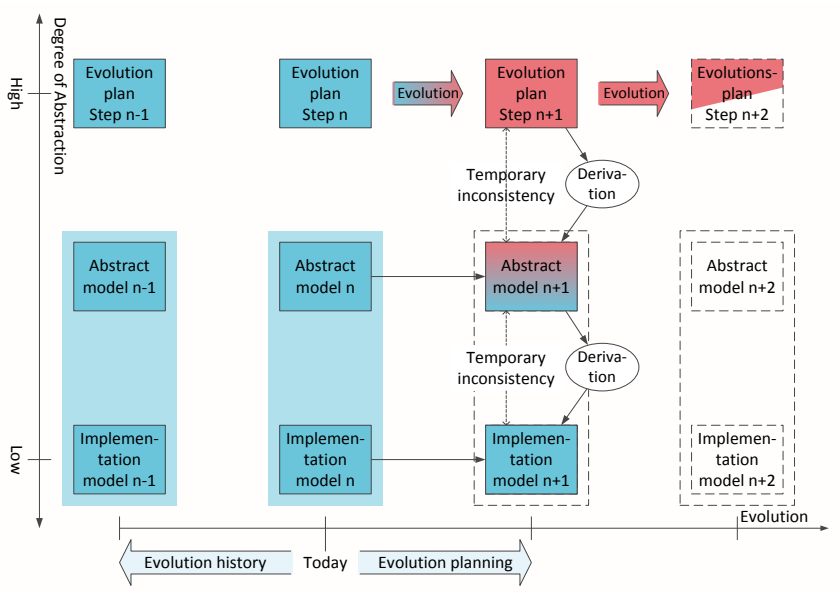

Figure 6: Planned evolution

3.2.5 Organizational structures and processes.

There are varying organizational structures and process structures for PLE approaches, e.g., we have consider which organizational units are responsible for Domain and Application Engineering [5]. When transitioning to an SoS context, these organizational structures have to be re-considered accordingly. For instance, we can have one Domain Engineering team that is responsible for multiple subsystems or the responsibility for Domain Engineering can be distributed among different teams.

4 Evolution in Systems of Systems

In this section we will look at the interplay between Systems of Systems and Evolution. Correspondingly, we will discuss relationships between concepts in Systems of Systems Engineering and Software Evolution.

The lack of strategies and techniques for a sustainable evolution of very large and complex systems is one of the main motivators towards SoS research.

4.1 Historic evolution vs. future evolution

As discussed earlier we can consider software evolution from **two perspectives**, the analysis of evolution "in the past" and the planning and management of evolution "in the future". In a SoS context, both aspects are relevant just as well: On the one hand, we are interested how and where evolution occurs in a SoS, e.g., to better understand how everything is connected, how changes were propagated, or to determine if certain "laws" hold. On the other hand, we can aim to plan and manage the evolution happening in a SoS, e.g., with techniques to plan evolution over multiple interconnected systems and techniques to monitor and influence changes while they happen.

For illustration purposes consider Figure 6 horizontally showing evolution over time from left to right, vertically showing various abstraction levels including plans for evolution on the topmost layer. To the left we can look at historic evolution that happened in the past. To the right we can see planned, future evolution, where the red annotation indicates an induced change that temporarily causes inconsistencies and is propagated through the abstraction layers when these inconsistencies get removed again.

4.2 Consequences of SoS characteristics for software evolution

What consequences do Systems of Systems characteristics have for Software Evolution?

First of all, any approach to software evolution in a SoS context has to reflect the **hierarchical, recursive structure of SoS** and must be able to handle their, scale and complexity. For instance, when planning and realizing a change in a system, this change has potentially to be propagated into all subsystems.

Systems of Systems have by their very nature a long life span and to understand, create, and manage them we have to take a **long-term perspective**. Organizations dealing with or operating within Systems of Systems often have to take this long-term perspective.

There is no obvious reason, why Lehman's laws of evolution [21, 22] should not hold for Systems of Systems. For instance, one can expect that changes in the environment induce necessary changes in the SoS in order to keep it relevant. Obviously, whether these laws actually hold needs to be evaluated before more concrete statements can be made. Moreover, it would be interesting to see how the various variables and forces in Lehman's laws (e.g., development activity, growth rate, defect rate) react in a SoS context and how the descriptive models have to be extended.

Due to the **managerial independence of elements** and the disappearance of central control an approach that aims to control evolution for the whole SoS (e.g., by demanding changes all over the place) is infeasible in practice, since there is rarely a single entity which has enough control to implement these changes. Instead, evolutionary objectives of **multiple stakeholders** have to be considered and consolidated. An approach needs to tolerate potentially incomplete and inconsistent information.

Even though there is an independence of elements, we have to consider **dependencies among elements**, when planning and implementing changes. Also, it is impossible to introduce changes without causing **inconsistencies** at least temporarily. Hence, many changes can only happen incrementally and changes are propagated through an introduction and subsequent resolution of inconsistencies.

On a related notion, System of Systems undergo **evolutionary changes**, which can be seen from two sides: First, change is inevitable and omnipresent. If systems have to evolve to adapt to changes in their environment and to stay relevant [21], the same can be argued for systems of systems. Second, because of the size and complexity of Systems of Systems change can only happen in an evolutionary fashion, not as a disruptive "flick of a switch" event. As an example in our airport scenario consider the deployment of a faster Wi-Fi network, to be used by both personnel and passengers. This causes changes in several subsystems, e.g., new authentication mechanisms, updates to the user database, mobile devices supporting the new standard, changes to software applications that using the network, etc.

On a process level the dependencies among elements and the evolutionary nature of changes has to be reflected in the corresponding **life-cycles** and processes. The overall "life-cycle" of the System of Systems is composed out of interconnected, interwoven smaller evolution steps of the individual subsystems and their elements.

4.3 More of the same?

Here, one could say that in order to handle evolution of Systems of Systems, we can just apply well known techniques for Software Evolution, just more of it. However, we would argue that above a certain level additional approaches are required.

For instance, Chen and Clothier [9] point out that the high-level engineering complexity (as discussed further by [32, 7]) raise great challenges in evolutionary development of SoS and indicates a need for considering different SE strategies at a level above individual projects.

Some part of SoS evolution can be covered by evolution on a component level, considering the component and its evolution in isolation. However, that is not sufficient. The real challenges lie in the coordinated evolution of multiple systems. Compare our earlier example of deploying a new network at an airport. Also see the discussion of "joint evolution" in [9, p. 173].

5 Conclusions

In this paper, we focused on the aspects of variability and evolution in a Systems of Systems context. We did so from two perspectives: First, we argue that Systems of Systems Engineering has to deal with variability and evolution and, hence, concepts from Product Line Engineering and Software Evolution can be helpful. Second, these disciplines, PLE and Software Evolution, more and more have to deal with very large systems where concepts from Systems of Systems Engineering can be helpful. We are aware that this paper mostly raises questions and does not provide a lot of answers. However, we strongly believe that an exchange between the fields of Systems of Systems Engineering, Product Line Engineering, and Software Evolution would be interesting and beneficial for both communities.

References

[1] D. Benavides, S. Segura, P. Trinidad & A. Ruiz-Cortés (2007): *FAMA: Tooling a Framework for the Automated Analysis of Feature Models*. In: *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, doi:10.1.1.77.8501.

[2] David Benavides, Sergio Segura & Antonio Ruiz-Cortés (2010): *Automated analysis of feature models 20 years later*. *Information Systems* 35(6), pp. 615–636, doi:10.1016/j.is.2010.01.001.

[3] Danilo Beuche (2008): *Modeling and Building Software Product Lines with Pure::Variants*. In: *12th International Software Product Line Conference (SPLC 2008)*, Limerick, Ireland, doi:10.1109/SPLC.2008.53.

[4] Jan Bosch (2000): *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, doi:10.1.1.107.7212.

[5] Jan Bosch (2002): *Maturity and Evolution in Software Product Lines: Approaches, Artefacts, and Organization*. In Garry Chastek, editor: *Proceedings of the Second Software Product Line Conference*, LNCS 2379, Springer, San Diego, CA, pp. 257–271, doi:10.1.1.92.3163.

[6] Goetz Botterweck & Andreas Pleuss (2012): *S2T2-Configurator: Interactive Support for Configuration of Large Feature Models*. In: *8th European Conference on Modelling Foundations and Applications (ECMFA 2012) Tools Track*, Kgs. Lyngby, Denmark. Available at `http://hdl.handle.net/10344/2586`.

[7] P.G. Carlock & R.E. Fenton (2001): *System of Systems (SoS) enterprise systems engineering for information-intensive organizations*. Systems Engineering 4(4), pp. 242–261, doi:10.1002/sys.1021.

[8] Lianping Chen, Muhammad Ali Babar & Nour Ali (2009): *Variability management in software product lines: a systematic review*. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09, Carnegie Mellon University, Pittsburgh, PA, USA, pp. 81–90. Available at http://dl.acm.org/citation.cfm?id=1753235.1753247.

[9] P. Chen & J. Clothier (2003): *Advancing systems engineering for systems-of-systems challenges*. Systems engineering 6(3), pp. 170–183, doi:10.1002/sys.10042.

[10] Paul Clements & Linda M. Northrop (2002): *Software Product Lines: Practices and Patterns*. The SEI series in software engineering, Addison-Wesley, Boston, MA, USA. Available at http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30731.

[11] S.C. Cook (2001): *On the acquisition of systems of systems*. In: Proceedings of the 2001 INCOSE International Symposium, Melbourne AU. ISBN: 0-9720562-0-3.

[12] K. Czarnecki, S. Helson & U.W. Eisenecker (2004): *Staged configuration using feature models*. In R. Nord, editor: 3rd International Software Product Line Conference (SPLC 2004), LNCS 3154, Springer Berlin Heidelberg, Boston, MA, USA, pp. 266–283, doi:10.1007/978-3-540-28630-1_17.

[13] Krysztof Czarnecki & Ulrich W. Eisenecker (2000): *Generative Programming*. Addison Wesley, Reading, MA, USA.

[14] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid & Andrzej Wąsowski (2012): *Cool features and tough decisions: a comparison of variability modeling approaches*. In: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '12, ACM, New York, NY, USA, pp. 173–182, doi:10.1145/2110147.2110167.

[15] A. van Deursen, E. Visser & J. Warmer (2007): *Model-Driven Software Evolution: A Research Agenda*. In Dalila Tamzalit, editor: Proceedings 1st International Workshop on Model-Driven Software Evolution (MoDSE), University of Nantes, pp. 41–49. Available at http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2007-006.pdf.

[16] D. Dhungana, P. Grünbacher & R. Rabiser (2011): *The DOPLER Meta-Tool for Decision-Oriented Variability Modeling: A Multiple Case Study*. Automated Software Engineering 18(1), pp. 77–114, doi:10.1007/s10515-010-0076-6.

[17] Deepak Dhungana, Thomas Neumayer, Paul Grünbacher & Rick Rabiser (2008): *Supporting Evolution in Model-Based Product Line Engineering*. In: 12th International Software Product Line Conference (SPLC 2008), pp. 319–328, doi:10.1109/SPLC.2008.26.

[18] Deepak Dhungana, Dominik Seichter, Goetz Botterweck, Rick Rabiser, Paul Gruenbacher, David Benavides & Jose A. Galindo (2011): *Configuration of Multi Product Lines by Bridging Heterogeneous Variability Modeling Approaches*. In: Proceedings of the 15th International Software Product Line Conference (SPLC 2011), Munich, Germany, doi:10.1109/SPLC.2011.22.

[19] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak & A. Spencer Peterson (1990): *Feature Oriented Domain Analysis (FODA) Feasibility Study*. SEI Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute. Available at http://www.sei.cmu.edu/reports/90tr021.pdf.

[20] A.J. Krygiel (1999): *Behind the Wizard's Curtain. An Integration Environment for a System of Systems*. Technical Report, DTIC Document. Available at http://www.dodccrp.org/files/Krygiel_Wizards.pdf.

[21] M. M. Lehman (1996): *Laws of Software Evolution Revisited*. In Carlo Montangero, editor: *EWSPT*, *Lecture Notes in Computer Science* 1149, Springer, pp. 108–124, doi:10.1007/BFb0017737.

[22] M. M. Lehman & Juan F. Ramil (2001): *Rules and Tools for Software Evolution Planning and Management*. *Ann. Software Eng.* 11(1), pp. 15–44, doi:10.1023/A:1012535017876.

[23] M.W. Maier (1998): *Architecting principles for systems-of-systems*. *Systems Engineering* 1(4), pp. 267–284, doi:10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

[24] Mari Matinlassi (2004): *Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA*. In: *ICSE*, pp. 127–136. Available at http://csdl.computer.org/comp/proceedings/icse/2004/2163/00/21630127abs.htm.

[25] Tom Mens & Serge Demeyer, editors (2008): *Software Evolution*. Springer, doi:10.1007/978-3-540-76440-3.

[26] D. Parnas (1976): *On the Design and Development of Program Families*. *IEEE Transactions on Software Engineering* SE-2(1), pp. 1–9, doi:10.1109/TSE.1976.233797.

[27] Andreas Pleuss & Goetz Botterweck (2012): *Visualization of variability and configuration options*. *International Journal on Software Tools for Technology Transfer (STTT)*, pp. 1–14, doi:10.1007/s10009-012-0252-z.

[28] Andreas Pleuss, Goetz Botterweck, Deepak Dhungana, Andreas Polzer & Stefan Kowalewski (2012): *Model-driven Support for Product Line Evolution on Feature Level*. *Journal of Systems and Software (JSS) - Special Issue on Automated Software Evolution* 85(10), pp. 2261–2274, doi:10.1016/j.jss.2011.08.008.

[29] Klaus Pohl, Günter Böckle & Frank van der Linden (2005): *Software Product Line Engineering : Foundations, Principles, and Techniques*. Springer, New York, NY. Available at http://www.springer.com/computer/swe/book/978-3-540-24372-4.

[30] M.-O. Reiser & M. Weber (2006): *Managing Highly Complex Product Families with Multi-Level Feature Trees*. In: *Requirements Engineering, 14th IEEE International Conference*, pp. 149 –158, doi:10.1109/RE.2006.39.

[31] Daniel Sabin & Rainer Weigel (1998): *Product Configuration Frameworks - A Survey*. *IEEE Intelligent Systems and Applications* 13(4), pp. 42–49, doi:10.1109/5254.708432.

[32] A.P. Sage & C.D. Cuppan (2001): *On the systems engineering and management of systems of systems and federations of systems*. *Information Knowledge Systems Management* 2(4), pp. 325–345. Available at http://iospress.metapress.com/content/wx6b5wft80k8p9a2/.

[33] K. Schmid & M. Schank (2000): *PuLSE-BEAT - A Decision Support Tool for Scoping Product Lines*. In: *Third International Workshop on Software Architectures for Product Families*, pp. 64–74, doi:10.1007/978-3-540-44542-5_8.

[34] Mathias Schubanz, Andreas Pleuss, Goetz Botterweck & Claus Lewerentz (2012): *Modeling rationale over time to support product line evolution planning*. In: *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, ACM, New York, NY, USA, pp. 193–199, doi:10.1145/2110147.2110169.

[35] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller & Gunter Saake (2012): *Predicting Performance via Automated Feature-Interaction Detection*. In: *Proceedings of International Conference on Software Engineering (ICSE)*, IEEE, pp. 167–177, doi:10.1109/ICSE.2012.6227196. Available at `http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/SKK+12.pdf`.

[36] Norbert Siegmund, Maik Mory, Janet Feigenspan, Gunter Saake, Mykhaylo Nykolaychuk & Marco Schumann (2012): *Interoperability of Non-functional Requirements in Complex Systems*. In: *ICSE2012: International Workshop on Software Engineering for Embedded Systems*, IEEE, pp. 2–8, doi:10.1109/SEES.2012.6225487. Available at `http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/SMF+12.pdf`.

[37] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel & Gunter Saake (2011): *SPL Conqueror: Toward Optimization of Non-functional Properties in Software Product Lines*. *Software Quality Journal* to appear, doi:10.1007/s11219-011-9152-9.

[38] Jeffrey M. Thompson & Mats Per Erik Heimdahl (2003): *Structuring product family requirements for n-dimensional and hierarchical product lines*. *Requir. Eng.* 8(1), pp. 42–54, doi:10.1007/s00766-003-0166-0.

[39] Jules White, Brian Dougherty & Douglas C. Schmidt (2009): *Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening*. *Journal of Systems and Software* 82(8), pp. 1268–1284, doi:10.1016/j.jss.2009.02.011.