Application area for multiple software product lines in automotive development

Uwe Beher, Guenter Boenisch, Mike Heidrich

Application area for multiple software product lines in automotive development

Uwe Beher[1], Guenter Boenisch[2], Mike Heidrich[3]

1: ESG Elektroniksystem- und Logistik-GmbH, uwe.beher@esg.de
2: Continental AG, Guenter.Boenisch@continental-corporation.com
3: Fraunhofer Einrichtung Systeme der Kommunikationstechnik (ESK), mike.heidrich@esk.fraunhofer.de

Abstract:
Since today's well known software product lines (SPL) approaches [1] [2] [4] [5] [6] [7] [8] [9] [12] focus on one single SPL, a methodology for connection and adjustment of multiple product lines is needed. The paper will shortly survey existing processes and methods e.g. FODA [4] for product lines and show adaptations to automotive industry. The focus of the paper will be an adapted process for automotive functional development, which is based on multiple software product lines (MSPL) and particularly regards customer-supplier relationship. It will propose a general SPL interface to manage using MSPL. The interface consists of a SPL Interface Methodology which defines steps for the adjustment of two or more different SPLs as well as a data interface (SPL Interface Data) which defines a data format for the exchange between different SPL tools. The SPL adjustment is demonstrated with a case study of an imaginary advanced driver assistance system called mobilSoft Adaptive Cruise Control (MCC), which consists of three product lines, one for the whole MCC and two for the subsystems linear tracking and traverse control.

Keywords: software product lines, multiple software product lines, SPL Interface Methodology, SPL Interface Data

1. Introduction

In automotive functional development aspects like software reliability and productivity draw more and more attention. As a consequence automotive manufacturers and suppliers address improved software engineering processes; introducing software product lines (SPL) increases reuse of software elements and supports accomplishing higher quality at less effort. Implementing improved SPL adaptations for automotive industry was the aim of the research program mobilSoft [14]. With the help of latest scientific methods and the experiences of the partners in industry, which were automotive manufacturers and suppliers, purposive solutions for manifold requirements to existing development processes especially for automotive application were found. Among these requirements were short development timelines, less development effort and high quality for each single software element. At the same time development processes are facing increasing variability and complexity of the final products.
Four steps were stated as necessary for Automotive OEMs or suppliers to obtain optimized SPLs and to get to a functional demonstrator, which verifies the feasibility of the planned adaptations for the SPL. The first step was an analysis at automotive manufactures and suppliers which lead to global requirements and characteristics for automotive SPLs and supporting processes. The second step was the evaluation of existing methods and tools, which define the technical state of the art, with the requirements found. The third step was evolving a specification of an automotive specific SPL, because existing approaches for SPLs were not particularly made for automotive application. The last step was the setup of a demonstrator, where the appropriate processes and methods were verified. Aim of this last step is not only proofing the concept of an adapted process, but also testing acceptance of comprehensive tool landscapes which are often combined with the introduction of integrated approaches.
Among these four steps the analysis of existing methods in the academic environment and available tools on the market turned out to be an appropriate basis for introducing SPLs improvements. In order to adapt existing tools to fit into existing processes, systematic and purposive investigations and determination of efforts are necessary. Especially introducing new methods in existing complex development structures is a critical task and needs well substantiated decision points.
Additionally, existing methods and processes were investigated for coverage of the interface between automotive OEMs and suppliers. Because of missing approaches for this important task in automotive development, additional methods were evolved which fit into to existing SPL as an extension of the process framework. The paper will have a main focus on this aspect and will propose an SPL Interface Methodology for adjustment of two or more different SPLs and SPL Interface Data as data interface during parallel development in different organizations. The result of applying these interfaces is a connection of single SPLs to a Multiple Software Product Line (MSPL).

2. State of the art software product lines

2.1 Existing SPL approaches

The arrangement of software development as a software product line is an effective method for increasing reuse in software development and well known in branches with high software share.

Short	Name	Short description
FODA	Feature Oriented Domain Analysis	Domain analysis method based on feature trees [4]
FORM	Feature Oriented Reuse Method	Extension of FODA for domain design [5]
COPA	Component Oriented Platform Architecting Method	Approach for setup of SPLs based on architecture and component assets [8]
FAST	Family-Oriented Abstraction Specification Translation	Total SPL framework [12]
QADA	Quality Driven Architecture Design	Quality oriented software design [7]
PuLSE	Product Line Software Engineering	Generic and comprehensive framework for SPL [2]
KobrA	Komponen-tenbasierte Anwendungs-entwicklung	Component based software development, application of PuLSE [1]
EAST-ADL	EAST - Architecture Description Language	Usage of variants in automotive software design [6]
pure:: variants	pure:: variants	SPL modeling tool [9] [pure::systems]

Table 1: Overview of SPL approaches

It was initially applied e.g. in telecommunication, medical equipment or consumer products, but recently automotive industry shows increasing interest as well. In table 1, a short overview of existing SPL approaches is given. The listed SPL approaches mainly differ in the focus on software engineering tasks. The approaches can be classified in terms of two dominating characteristics.

- The phases of software engineering including scoping, analysis, design or implementation of software components
- Main process areas of a SPL, which are domain engineering and application engineering and which have their focus on designing and applying the SPL.

Figure 1 shows the coverage of the different approaches in terms of the different software engineering phases and SPL process areas. The range of the coverage extends from very specialized tasks for SPLs up to integrated and comprehensive process models.

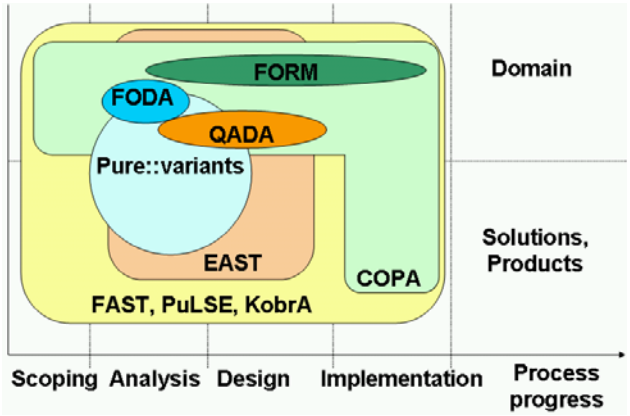

Figure 1: Coverage of SPL approaches

The application of one of these approaches as whole or partial as individual interpretation of the SPL methodology is, in some instances, realized at automotive manufacturers or suppliers. However, the adaptation of the given approaches to the needs of automotive product development remains being a very complex task and only few parts of the listed approaches fit in the special automotive environment. Thus the following chapter proposes characteristics needed for a SPL to be suitable as an (automotive) MSPL.

2.2 Characteristics of an automotive SPL

Regarding [13] the SPL is divided into the process areas domain engineering and application engineering. Domain engineering leads to the setup of the software product family and the design of the assets, application engineering deploys the assets of the software product family in order to generate a final product. As preface for a generative approach [3] the automotive SPL is divided into

requirements space and solution space. This results in the following alignments of the SPL:

- Domain analysis is the requirements space of the domain engineering
- Domain design is the solution space for the domain engineering
- Application analysis is the requirements space of the application engineering
- Application design is the solution space for the application engineering

A common method for organizing and illustrating in the requirements space is the usage of feature models for variant requirements, which include features and the relations between features. Generally a feature model e.g. FODA represents all possible requirements which can be met by the product family.

- A Feature describes the requested product properties from the point of view of a stakeholder, which in the context of a SPL is a customer, developer, manager, investor or supplier.
- A feature reflects single requirements or a set of aggregated requirements.
- The feature relation describes the relationship between one or more features, which is mandatory, optional or excluding.

The selections of features in the feature model during application engineering builds up single products of solution space.

The solution space contains a solution model for the domain and the resulting solutions by specific selections in the solution model. The solution selection itself is the result of decisions made along the solution path. The solution model consists of selected assets and their relationship. An asset is the smallest unit of a solution model and can't be divided into smaller units. Assets can be categorized into three types.

- A basic asset is a solution, which can be used in many products without any change.
- A customized asset is a basic asset, which can be used in many products and which includes complete calibration possibilities.
- A specific asset is a solution for only one product, which is initially introduced into the product family and which aims at utilization for further products.

For example an innovation is a specific asset that is implemented in a product for the first time and which, in the case of a successful introduction in the market, is planned to be extended to other products as customized or basic asset. Aligning the design of an asset with an existing product family eases the implementation of the project specific solution into the SPL.

The feature-asset relation expresses the relationship between an asset and a feature. Because of a feature being a set of one or more requirements, an assignment between requirements and solution is achieved. The FA Linker (feature asset linker) includes the relationship between chosen features and related assets and thus regards only the relevant subset from the total set of all feature asset relations. For the interface between SPLs, which is described in the following chapter, the FA linker plays a crucial role.

With the characteristics and main tasks proposed for SPL software engineering, an appropriate fundament is formed for the specification of an automotive SPL. This leads to connecting different SPLs to a Multiple Software Product Line (MSPL), which is also an automotive requirement.

3. Connection of software product lines

3.1 Fundamentals of connecting product lines

The automotive final product "vehicle" of the OEM is not the result of only one integrated product line. It rather consists of different subsystems with their own product lines and additional single solutions e.g. innovations, which are initially implemented only in few car models. Additionally, each subsystem may be delivered from its own organization, which may be internal or an external supplier. As a result the structure of each product line and the stakeholders mapped to it may differ.

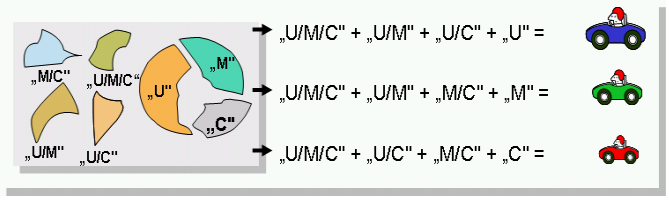

Figure 2: Example: Structure of an OEM SPL

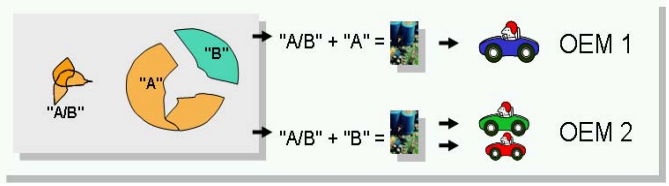

Figure 3: Example: Structure of a supplier SPL

Examples for possible product structures with assets at OEM and supplier are depicted in figure 2 and figure 3. Figure 2 shows an OEM with its three platforms "Upper", "Middle" and "Compact". The "Upper" platform itself consists of four assets. "U/M/C" is used in "Upper", "Middle" and "Compact"

platform, “U/M” in “Upper” and “Middle” platform, “U/C” in “Upper” and “Compact” platform and finally “U” only in “Upper”, which is a single solution. The structure of the supplier in figure 3 is similar in methodology but different for the solution. Its platform “OEM1” consists of “A/B”, a common asset for its platform “OEM1” and “OEM2”, and a single solution “A”. It is used in the “Upper” platform of the OEM in figure 2, but not restricted to it. The supplier has actually no influence on the OEM decision, to put it into “U/M/C”, “U/M”, “U/C” or “U”. And finally, supplier may sell a similar product to another OEM, which will not be regarded in the SPL of the first OEM. As a consequence, even if products from the supplier are used in the platform of the OEM, the configuration of assets which build up the solution space is independent.

Furthermore a simple one-to-one match in the solution space where a product “A/B”+”A” may fit into “upper class” vehicles is hardly applicable for automotive industry. In general, separate SPLs exist at different levels of functional hierarchy and depend on their own field of activity. In this case, all individual SPLs need to be synchronized in order to deliver the correct product at the right time.

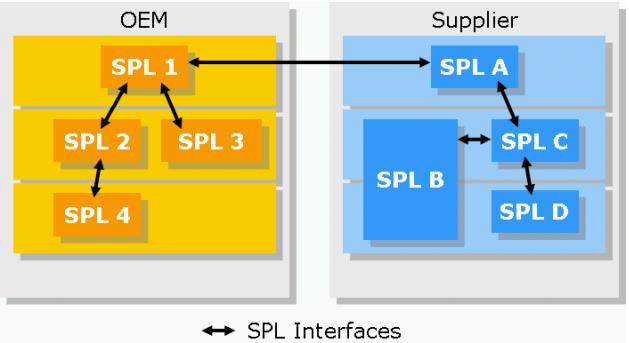

Figure 4: Connected SPLs

After synchronization, the setup being composed of connected SPLs is stated as a Multiple Software Product Line (MSPL). Figure 4 shows an example for the structure of connected SPLs. The interfaces in this example occur between SPLs internally in one company or externally in many companies. For a functional connection of SPL it is crucial, to align each single SPL in a correct hierarchical order.

A layered architecture e.g. EAST-ADL [6] is able to structure SPLs hierarchically. In the given example, “SPL A” at supplier may be at the highest level “user” as interface to the customer. But also internal product lines “SPL C” at “cluster” level or “SPL D” at “platform” level need correct alignment to facilitate “SPL A” providing the requested product.

3.2 Interfaces of multiple software product lines

As stated in the last chapter an important prerequisite for a MSPL is the synchronization of each SPL to each other in order to align development of the SPLs assets. This paper proposes two main tasks for synchronization:

- Exchange of requirements and design data via a standardized data model “SPL Interface Data”
- Adjustment of SPL specific tasks via a general methodology “SPL Interface Methodology”

Figure 5 shows the extension of existing data models for general automotive SPLs to implement interface data (“SPL Interface Data”) for adaptation to other SPLs.

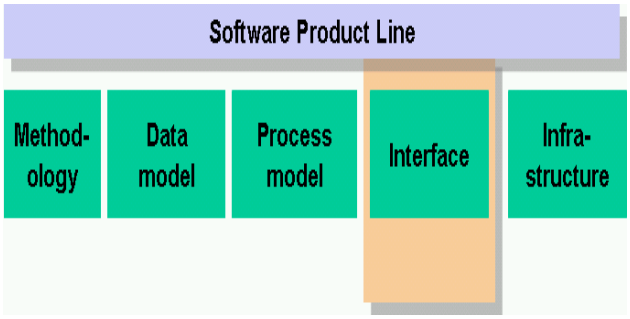

Figure 5: Extension of existing SPL data models

The SPL Interface Data generally contains design data like

- Product requirements
- Features of the product as set of requirements
- Architecture patterns as reference solutions

The data exchange for synchronization mainly takes part at an early stage of the development process in a SPL. An appropriate phase is after domain analysis, where each single SPL defines its own reuse concept and asset structure for the product development. Before reaching domain design phase, inputs from all other SPLs complete own basic product requirements by aligning own product application to the requirements of the total product. The SPL interface based on the data models and synchronization tasks needs to cope with a contradiction. On the one hand it shall be as flexible as needed to react on changed constraints after domain design, which at this point typically come from functional implementation. On the other hand the SPL interface shall serve as a consistent backbone for development data. Figure 6 illustrates the role of SPL Interface Data and SPL Interface Methodology in the context of single SPLs implemented into the total product development.

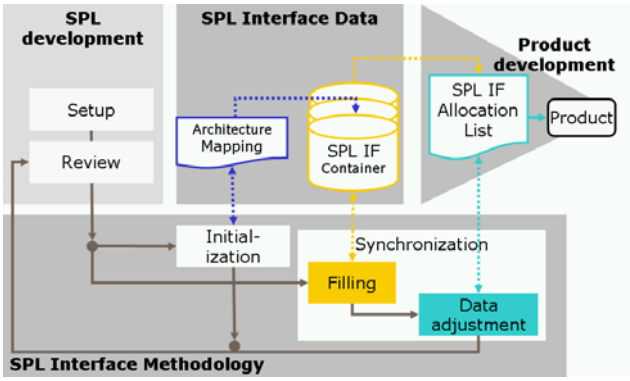

Figure 6: Concept of connecting SPLs

The part SPL development represents the individual SPL process of an organization like a company or a profit centre in a company. The product development covers the development of the final product as the result of the combination of the individual SPL processes. The SPL interface in between consist of two main elements, the central data container SPL Interface Data and the SPL Interface Methodology in order to synchronize the individual SPL processes. The SPL Interface Data serve as exchange mechanism for design data. It consists of

- an architecture mapping
- an SPL IF Allocation List
- the data container itself

The reason for an architecture mapping is founded in the different SPL approaches, where the connection into the total product architecture is not provided. The architecture mapping verifies the architecture patterns of the single SPLs and assigns each component to its correct place in the product, thus a common understanding of the product is established throughout all SPLs.

The allocation list synchronizes the product features and assets and supports the compliance of the requirements to a consistent product design. It deals as configuration support and is a main document for the total product development.

The data container is the physical container for design data for combined SPL development which are mainly requirements and assets. It is recommended to structure the container in terms of connected SPLs, this eases each SPL updating their own content during change management.

The SPL Interface Methodology describes the process for connecting, adapting and aligning two or more SPLs. It provides initialization and synchronization, which realize a strong connection of the individual development processes to the final product. The SPL Interface Data and the SPL Interface Methodology will be explained in more detail in the following chapters.

3.3 Architecture mapping and initialization

A key element of a SPL is a reference architecture, where single products are derived from. In order to connect software product lines, the elements of the reference architectures have to match together to generate a functional total architecture. Reference architectures may be EAST-ADL [6] or Car-DL [11]. The common base of most reference architectures is a layered system model, where abstract system elements at a higher level are divided into smaller elements at a lower lever which provide more details of the internal structure.

The architecture mapping coordinates the different reference architectures at initialization of SPL alignment. It contains methods for the combination of requirements and solution spaces in the correct level of abstraction. Provided, that each individual SPL has a reference architecture, which fits into the general architecture frame for the architecture mapping, the architecture mapping merges all SPLs into one central architecture of the MSPL. As a consequence of all reference architectures being abstract and generic, no SPL has to open its internal detailed architecture to the central architecture of the MSPL for the final product. That is e.g. strategic product plans and customer structures, which are often modeled in internal architectures, stay closed to other competing SPLs.

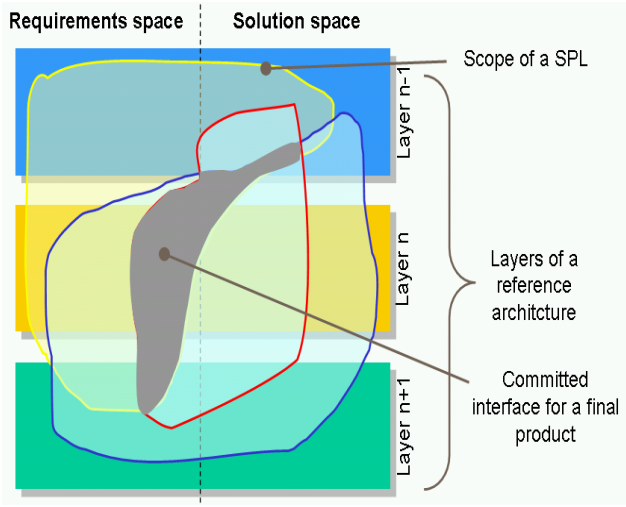

Figure 7: Alignment of different reference architectures at initialization

Figure 7 shows the result of architecture mapping of individual SPLs. In this figure, three possible SPLs are drawn as bordered shapes. The main objective of the architecture mapping in the SPL Interface Data is to assign each element of the single SPLs to the correct layer of the reference architecture and to requirements or solution space. By overlaying all SPLs valid intersections of all SPLs can be determined. The dark grey area in the middle of all

shapes is the common part of all SPLs, which needs a common problem and solution descriptions. Otherwise areas without any interaction have no significance for the SPL Interface Methodology.

3.4 Synchronization and allocation

In a MSPL individual SPLs exchange data mainly during synchronization phase. The data base for the data exchanged is the SPL IF container. Content of the container are requirements, features as sets of requirements, assets, relations of container elements and additional documents, which support version control and asset history. Synchronization is separated into two steps, filling and data adjustment. During filling phase, all interface data of the other product lines are collected and structured. Basis of the structure is the architecture mapping described previously, which is elaborated during initialization. After all intersections of the product line are identified, all features and assets are put together in the data adjustment phase. As a result, a comprehensive configuration for all features, assets and relations is found, which is valid for realizing the total product. This configuration is a global SPL IF allocation list which contains the relationship between all features and assets involved in the product development.

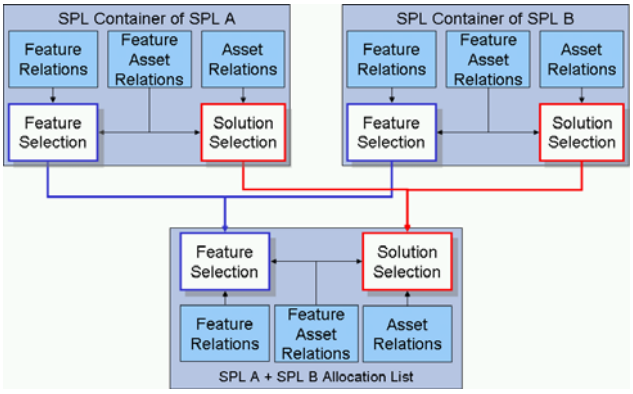

Figure 8: Allocation list

Figure 8 shows the structure of the allocation list, if two SPLs “SPL A” and “SPL B” are connected. Only if the allocation list as a combination of both SPLs contains only viable product solutions, data adjustment phase is finished. Product application design and domain design follow after the adjustment phase. In case of changes in single SPLs are required due to the result of a review, the overall SPL adjustment is triggered again.

Figure 9 shows the resulting package model for a MSPL which contains all data elaborated for connected SPLs. These data are SPL specific data like own reuse, SPL model or Scope or data especially defined for connection like layered architecture or SPL interface.

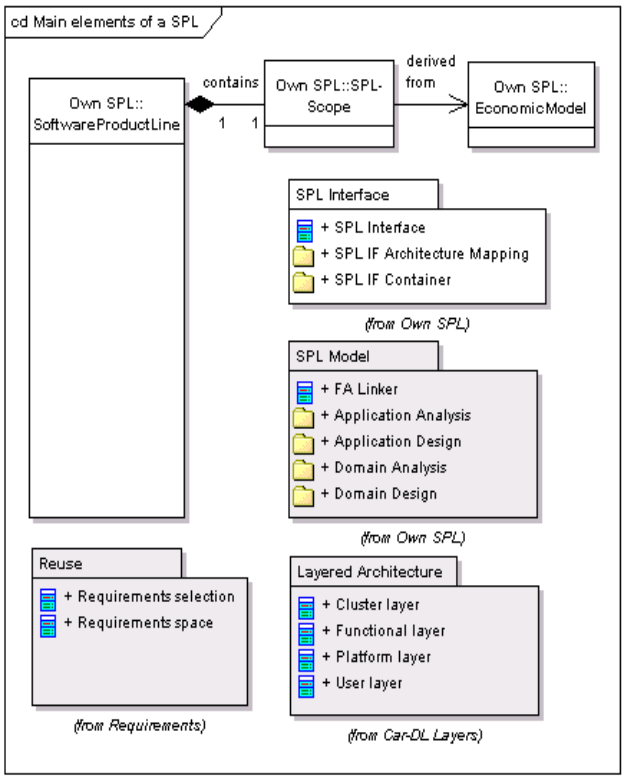

Figure 9: Package model of a MSPL

4. Demonstrator for Multiple Software Product Lines

4.1 Main task of the demonstrator

The demonstrator supports the verification of an adapted SPL regarding implementation into a MSPL. A second effect, which development organizations should not neglect, is the acceptance test for introducing integrated software tool landscapes which comes along with the SPL adaptation.

An option for a MSPL demonstration is building up the individual SPLs as hardware independent function, which eases realization of the demonstration. The software platform to be used may be a standardized platform like AUTOSAR, but AUTOSAR is not a prerequisite for the demonstrator. The concept of the stakeholder need for performing the interface for the SPLs does not reflect existing personnel but describes abstract roles and tasks that need to be performed. The mapping of the requested roles to an organizational chart has to be found for each organization individually. The process shown in figure 10 can serve as a template for all automotive organizations in order to perform their own SPL adaptation for a MSPL.

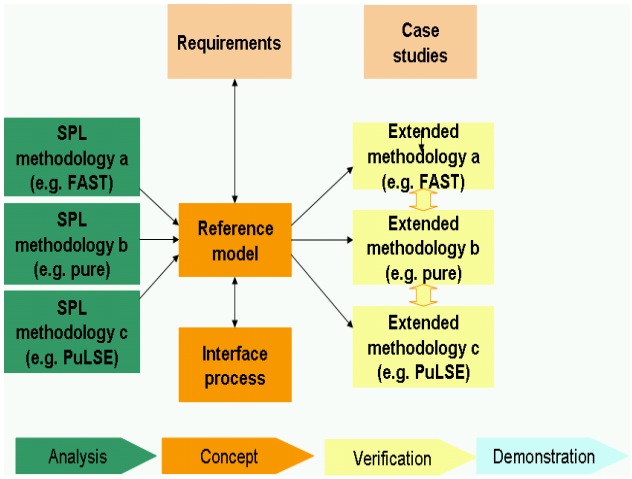

Figure 10: Steps for realizing a MSPL

During analysis phase of SPL the own standard SPL process or individual interpretation of SPL has to be investigated for adaptations for the MSPL. The result is a general reference model, which supports connection to other SPLs and is able to accomplish own requirements. The verification is achieved by setting up a case study, which combines several methods and SPLs together. The demonstration helps out for analyzing the results of the total process.

4.2 Results of connection of software product lines

Figure 11 shows the setup of the function mobilSoft Cruise Control (MCC), which is composed of three product lines

- The overall MCC function
- Linear tracking for MCC function
- Traverse control for MCC function

In this case study tracking and traverse control are SPLs of two automotive suppliers and the total MCC function a SPL at one OEM. The modeling of all feature models for the problem and solution space was realized with pure::variants [9]. Architecture mapping was evolved within the research project mobilSoft [14] by the use of Car-DL[11] as a reference architecture. However the shown approach can be used with other architectures, e.g. EAST-EEA-ADL [6] as well. The appropriate files for the SPL IF container and the SPL IF allocation list were realized as *.xml-documents according to XMS scheme definitions (XMD) as output of UML based data models. Finally an iterative refinement of the MSPL approach was accomplished.

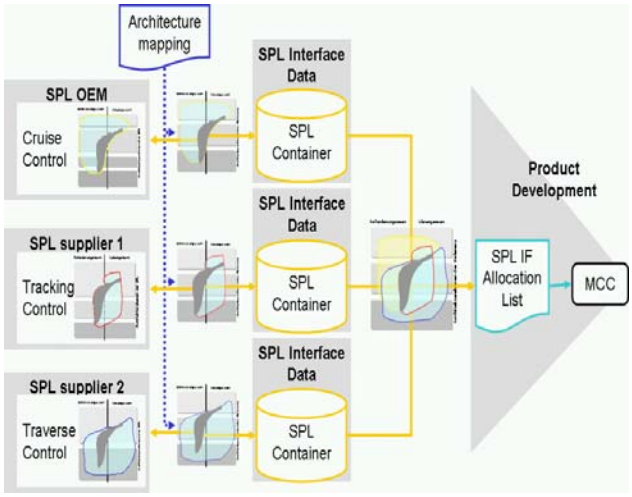

Figure 11: mobilSoft Cruise Control as a MSPL

5. Conclusion

This paper has provided the structure of SPL Interface Methodology and SPL Interface Data in order to adjust many individual SPLs into one Multiple Software Product Line. The result is a global adjustment list for requirements and solution space of all involved SPLs. The adjustment list is the base of product application in each SPL and is an additional specification at the early stage of a product development process. In automotive applications, after implementing additional requirements emerging from the adjustment list, existing SPL process for product application shall be able to perform SPL product development. The extension of the MSPL has no necessity for special SPL tools and shall be ready to be integrated in most tool supported SPLs.

6. Acknowledgement

The authors acknowledge the contribution all partners of mobilSoft TP6 to this work which are - in alphabetical order - Audi AG, ESG Elektroniksystem- und Logistik GmbH, Siemens VDO Automotive AG and Fraunhofer Einrichtung Systeme der Kommunikationstechnik.

7. References

[1] C. Atkinson, J. Bayer, O. Laitenberger, and J. Zettel, "Component-Based Software Engineering: The KobrA Approach," 2001

[2] PuLSE™: A Methodology to Develop Software Product Lines, Authors: J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), (Los Angeles, CA, USA), May 1999, pp. 122-131.

[3] K. Czarnecki, U. W. Eisenecker, Generative Programming, Addison-Wesley, Reading, MA, 2000.

[4] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and P. A. Spencer, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," 1990.

[5] K. C. Kang, S. Kim, K. Kim, G. J. Kim, and E. Shin, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," 1998.

[6] H. Lönn (ed.), “Definition of language for automotive embedded electronic architecture”, EAST-EEA Deliverable D3.6, public report, Version 1.02, 2004.

[7] M. Matinlassi, E. Niemelä, and L. Dobrica, "Quality driven architecture design and quality analysis method," 2002.

[8] H. Obbink, J. Müller, P. America, and R. v. Ommering, "COPA - A Component-Oriented Platform Architecting Method for Families of Software-Intensive Electronic Products," 2000.

[9] Pure-Systems GmbH, Tool pure::variants, http://www.pure-systems.com.

[10] S. Voget (ed.), “Embedded Electronic Architecture - Glossary”, EAST-EEA public report, Version 7.3, 2004.

[11] D. Wild, A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, and S. Rittmann, “An architecture-centric approach towards the construction of dependable automotive software”, In S. of Automotive Engineers, editor, Proceedings of of the SAE 2006 World Congress, Detroit, 2006.

[12] D. Weiss, C. Lai, and R. Tau, Software product-line engineering: a family-based software development process. Addison-Wesley, Reading, MA, 1999.

[13] G. Böckle, P. Knauber, K. Pohl, and K. Schmid, "Software-Produktlinien," 2004.

[14] Softwaretechnik für das Automobil der Zukunft, http://www.itm.tum.de/mobilsoft/Startseite.htm.

8. Glossary

Artifact: A physical piece of information that is used or produced by a software development process. Examples of Artifacts include models, source files, scripts, and binary executable files.

Asset: Artifacts in the solution space of a SPL. Examples of assets includes models, code, documentation or test cases.

Domain Engineering: Software engineering part of a SPL which covers the commonalties and variants of all solutions of a SPL.

Feature: A feature describes a product property from the point of view of a stakeholder. A feature is a set of requirements or a single requirement. The relationship between features may be optional or mandatory and which may have constraints. The relationship is graphically described in a feature model.

Layered Architecture: Base structure of a software product line architecture e.g. Car-DL or EAST-ADL. In a layered architecture a system is structured into different levels of details.

MCC: Mobilsoft Cruise Control, the realized demonstrator for a multiple software product line evolved by the project “Teilprojekt 6” of the research program “mobilSoft” under the auspices of the German state of Bavaria.

MSPL: Multiple Software product line, connection of many individual and independent software product lines

Requirements space: Focus of all requirements in a software development process. Artifacts of the problem space in a SPL are features.

Scoping: Software engineering part of the domain engineering, which consists of product line scoping (identification and description of products), domain scoping (determination of common and different application domains) and asset scoping (planning of the assets for the SPL).

SPL: Software product line, a mechanism for software development with focus on reusability, productivity and quality.

Solution space: Solving of all requirements in a software development process. Artifacts of the solution space in a SPL are assets