Dipl.-Ing. Andrea Leitner Bakk.techn.

Multi-paradigm variability modeling for Software Product Lines

---

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr.techn.)

durchgeführt am Institut für Technische Informatik
Technische Universität Graz
Gutachter: Em. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Graz, im Oktober 2012

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....
date (signature)

Kurzfassung

Die zunehmende Bedeutung und Komplexität eingebetteter Software macht moderne Entwicklungsmethoden immer wichtiger. Besonders im Automobilbereich wird mehr und mehr Funktionalität und Innovation mittels Elektronik anstelle von Mechanik realisiert. Die große Variantenvielfalt (Fahrzeugtypen, in Software realisierte Funtionalität, etc.) und immer kürzer werdende Entwicklungszyklen erfordern zukunftsweisende Entwicklungs- und Wiederverwendungsstrategien, wie z.B. generische Softwarearchitekturen. Eine weitere Herausforderung stellt die Heterogenität der Domäne dar. Unterschiedliche Disziplinen mit spezifischen Systemsichten sind an der Entwicklung von Fahrzeugen beteiligt.
Software Produktlinien sind eine vielversprechende Methode systematische Wiederverwendung für die Entwicklung einer definierten Menge ähnlicher Produkte zu nutzen. Die Aufteilung in eine abstrakte Problembeschreibung (inkl. Variabilität) und beliebig viele technische Realisierungen für das beschriebene Problem verringert nicht nur die Komplexität der Beschreibung, sondern ermöglicht auch eine konsistente Variabilitätskontrolle. Dadurch wird die Problembeschreibung zu einem integralen Bestandteil der Produktlinie. Die Art der Repräsentation muss daher so gewählt werden, dass Produkte effizient und möglichst fehlerfrei erzeugt werden können. Auf Basis von bewertbaren Kriterien, die aus der Untersuchung verschiedener Domänen resultieren, wurde eine systematische Unterstützung im Entscheidungsfindungsprozess zur Auswahl eines Modellierungsparadigmas entwickelt.
Oft sind unterschiedliche Systemsichten durch verschiedenste Charakteristika geprägt, was eine eindeutige Auswahl des Modellierungsparadigmas unmöglich macht. Um eine bessere Gesamtrepräsentation zu erhalten, ist es daher notwendig verschiedene Paradigmen gemeinsam zu verwenden. Diese Arbeit beschreibt eine Methode um domänenspezifische Sprachen und feature-orierentierte Variabilitätsmodellierungsparadigmen, basierend auf bestehenden Technologien, zu kombinieren.
Die vorgestellten Strategien werden am Beispiel von eingebetteter Software für Hybridelektrofahrzeuge demonstriert. Aufgabe dieser Software ist es, verschiedene Antriebe zu koordineren, den Fahrkomfort zu verbessern und den Fahrer zu unterstützen. Wesentliche Beiträge dieser Arbeit sind eine Methode zur Unterstützung der Auswahl eines Modellierungsparadigmas, ein Modellierungsansatz zur Darstellung von heterogenen Domänen und ein Konzept um die Konsistenz zwischen verschiedenen Artifakten im Entwicklungsprozess zu gewährleisten.

Abstract

The importance and complexity of embedded software is growing evermore leading to the demand for more advanced development strategies. The automotive domain for example, changed and still changes its focus from mechanics towards electronics. Due to the high diversity of variants (vehicle types, software supported functionality, etc.) in this domain, along with the need for shorter development cycles, modern development, and reuse strategies like generic software architectures and mass-customization are getting vital. Another challenge is the heterogeneous nature of the domain. Various disciplines are involved in the development of automotive systems resulting in a need for interrelated stakeholder views.

Software product lines are a viable approach to significantly support systematic reuse of a series of similar products. This corresponds to the understanding of "generic" as a set of predefined products. The separation between a high-level variability description and various coexisting, but consistently instantiated technical realizations raises the level of abstraction. Additionally, it ensures consistency throughout all artifacts of the development process. The central role assigned to the problem description requires an improved representation to increase efficiency and reduce errors. Investigating various domains results in assessable criteria, that serve as basis for the selection of an appropriate modeling paradigm for a certain domain or domain view. Nevertheless, stakeholder views often imply diverse characteristics, which favor different paradigms. In order to achieve a better overall representation of the entire domain, this work proposes a combined representation of domain-specific languages and feature-oriented modeling approaches to describe variability based on existing technology.Implementations of variability mechanisms for key development environments (e.g. Simulink) of automotive software can be used to consistently control variability.
The proposed strategies are demonstrated using the example of automotive control software for hybrid electric vehicles. This control software coordinates different energy sources, improves the driveability, and supports the driver. Variability is introduced by various drivetrain configurations, various vehicle types, various markets with different legal constraints, and so on. Significant contributions include a method to support modeling paradigm selection in order to improve the representation of the problem domain, a multi-paradigm variability modeling approach to represent heterogeneous domains, and a concept to ensure consistency over various artifacts in the development process.

Acknowledgements

In the beginning, I would like to thank my supervisor Prof. Dr. Reinhold Weiß for his helpful advice during the conduction of my dissertation at the Institute for Technical Informatics at Graz University of Technology. I especially want to thank Dr. Christian Kreiner and Dr. Christian Steger for their support and mentoring during the last years. Special thanks to Timo Käkölä for his valuable input and fruitful discussions.

I would like to thank all people and companies involved within the HybConS project. My special thanks go to the Virtual Vehicle Competence Center for providing the framework for the HybConS project, and especially to Wolfgang Ebner, Petr Micek, Bernhard Knauder, Simon Waltenberger and Dr. Evgeny Korsunsky for their support. Additionally, I would like to thank Nermin Kajtazovic, Wolfgang Raschke, Nicolas Pavlidis, and Philipp Töglhofer for their valuable contribution.

Furthermore, my thanks go to the entire staff at the Institute for Technical Informatics for the organizational support during my work.

My greatest gratitude belongs to my partner and my family for their support, understanding and guidance throughout my life.

Graz, October 2012 Andrea Leitner

Extended Summary

The importance of automotive software and the number of embedded software functions is growing continuously. More and more functionality is provided by software. As a result, complexity raises and more advanced development strategies are required. An important attribute is the potential for mass-customization in order to deliver customer-specific cars. This also influences software since it requires a generic software solution which supports the development of customer-specific control software variants. Generic in common sense means to provide all possible variants. In a realistic scenario this is not feasible since it would result in unmanageable complexity. The first step to reduce complexity is to restrict the domain to a certain set of variants. Obviously, for complex systems as for example automotive control software there is still a lot of variability even for a small number of variants.

Software Product Line Engineering (SPLE) allows to systematically describe variability in software and, as a result, to systematically reuse software components or modules. The major concept is the distinction between two different development processes. In *Domain Engineering*, the generic platform containing variability is built. Variability is made explicit in terms of variation points, which indicate the existence of alternative realizations. In *Application Engineering*, concrete products are derived from the platform by taking decisions for each variation point. Besides systematic reuse, one major advantage of SPLE is the possibility to control variability consistently over all artifacts in the development process (*single point of variability control*). This is realized by the separation of the problem description and different technical solutions (requirements, design, implementation, tests, documentation, etc.) as shown in Figure 1 and Section 6.5. This work mainly focuses on domain engineering, and there in particular on the problem space. Technical realizations from the automotive domain are used to show the applicability of the single point of variability control concept (see Section 6.8 and Section 6.9).

This work is part of a national funded project called HybConS (Hybrid Control Systems). The proposed concepts are independent of a specific problem domain, but are subsequently described for automotive control software. The objective of the overall project is the development of a generic software architecture for control units of hybrid electric vehicles (HEV). Hybrid electric vehicles basically consist of at least one electric motor and some other kind of energy source, usually a combustion engine. The aim of the software is to control the hybrid electric system. Due to the complexity of the overall system there are many influencing factors for the software. Some examples are illustrated in Figure 2. Different strategies for an improved representation of the domain are required in order to handle the complexity.

Within this thesis, different domains have been analyzed (see Section 6.1 and Section 6.2)

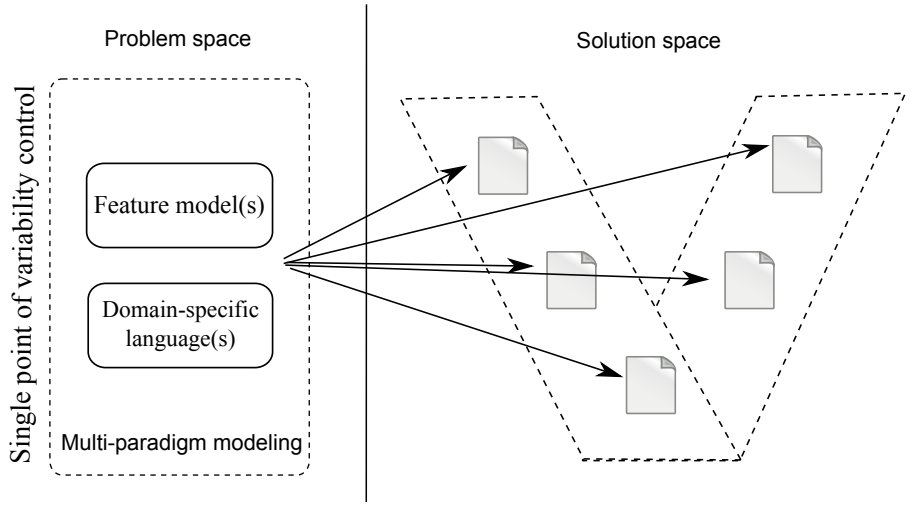


Figure 1: Improved representation of the problem space through the use of multi-paradigm modeling together with a separation of concerns in order to consistently control variability in various development artifacts.

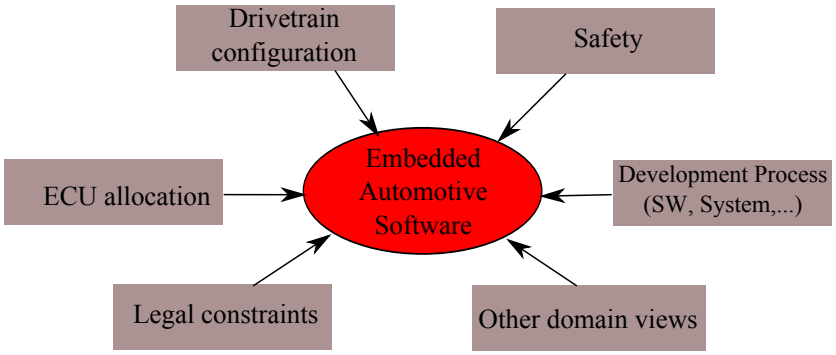


Figure 2: Various interrelated domain views for embedded automotive software

in order to find a modeling paradigm resulting in a domain model with low complexity. Two modeling paradigms have been identified to be of practical relevance: Domain-Specific Modeling (DSM) and Feature-Oriented Domain Modeling (FODM). The investigation of domains revealed different domain characteristics favoring different modeling paradigms. Analyzing the influence of the modeling paradigm on the representation complexity depending on domain characteristics resulted in a method to support the decision making process for an appropriate modeling paradigm (see Section 6.3). The evaluation of different domains further shows that complex domains often can be decomposed to several views with diverse characteristics. These domains are here referred to as *heterogeneous*. The main problem resulting from heterogeneous domains is the inability to find an appropriate modeling paradigm for the entire domain. Despite the fact that there are proposals towards new languages supporting or combining domain characteristics, there is no satisfying solution. The proposed multi-paradigm variability modeling approach (see Section 6.6) uses existing and proven technology wherever possible. The concept is based on the

fact that modeling approaches consist of three main building parts: elements, connections between elements and properties of elements. The most fundamental problem when combining different variability modeling paradigms is the definition of inter-model constraints (e.g. element in a model depends on element in another model). In the proposed solution, the constraints are restricted to the main building parts and are represented in a *Common Domain Model*. Figure 3 illustrates the basic multi-paradigm modeling concept.

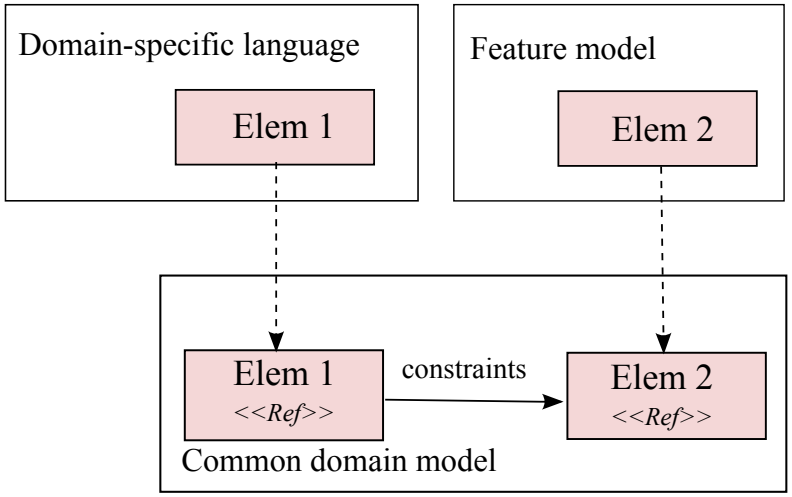


Figure 3: Multi-paradigm modeling concept to define inter-model constraints in a Common Domain Model. The Common Domain Model only consists of references to original model elements.

As a metric to evaluate the usefulness of the proposed approach, the domain representation complexity has been assessed. Representation complexity can be seen as an important quality metric since the domain model is used for the derivation of many products. Furthermore, the domain model has a long life cycle. Using a model with lower complexity helps application engineers to work more efficient and reduces the probability of errors in the product derivation process. In order to measure the representation complexity, three simple metrics have been defined (see Section 6.4). The metrics are again based on the main building parts. Element complexity measures complexity on element level, interface complexity the number of varying connections and, finally, property complexity counts the number of non-fixed properties. The sum of these single values results in an overall representation complexity value. This value can be used to compare different representations. An evaluation of different domain representations shows that the multi-paradigm variability modeling approach results in lower complexity, because the representation of single views can be improved. Inter-model constraints in this case do not cause significant overhead.

In conclusion, the main objective of this thesis is an efficient strategy for systematic reuse. Improved domain representations, in particular of heterogeneous domains, are used to reduce complexity. Significant contributions include the complexity reduction of domain representations, an improved approach to represent heterogeneous domains and a concept to ensure consistency over the entire development process (see Figure 1).

Contents

List of Figures

List of Abbreviations

AE	Application Engineering
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitecture
CVL	Common Variability Language
DE	Domain Engineering
DSM	Domain-Specific Modeling
DSL	Domain-Specific Language
EAST-ADL	Electronics Architecture and Software Technology - Architecture Description Language
ECU	Electronic Control Unit
EMF	Eclipse Modeling Framework
ERP	Enterprise Resource Planning
FODA	Feature-oriented Domain Analysis
FODM	Feature-oriented Domain Modeling
HEV	Hybrid Electric Vehicle
IDE	Integrated Development Environment
InCoME	Integrated Cost Model for Product Line Engineering
MADMAPS	Multi-Attribute Domain Modeling Approach for Paradigm Selection
MBD	Model-based Development
MDA	Model-driven Architecture
MDD	Model-driven Development
PIM	Platform Independent Model
PL	Product Line
PLE	Product Line Engineering
PSM	Platform Specific Model
OEM	Original Equipment Manufacturer
OMG	Object Management Group
RTE	Runtime Environment
SEI	Software Engineering Institute
SOA	Service-Oriented Architecture
SPL	Software Product Line
SPLE	Software Product Line Engineering
UML	Unified Modeling Language
VDM	Variant Description Model
VRM	Variant Result Model
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

Glossary

Application engineering *is the process of software product line engineering in which the applications of the product line are built by reusing domain artifacts and exploiting the product line variability.*

Binding time *is the point in time when the decision upon selection of a variant must be made.*

Domain engineering *is the process of software product line engineering in which the commonality and the variability of the product line are defined and realized.*

Domain model: *definition of the functions, objects, data, and relationships in a domain.*

Domain-specific language *is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.*

Domain-specific modeling *use domain-specific languages for the specification of domain concepts.*

Feature: *prominent and distinctive user visible characteristic of a system.*

Feature-oriented domain modeling *uses features to describe commonalities and variabilities of a domain usually in a feature diagram.*

Generic architecture: *Architecture supporting a defined set of solutions.*

Heterogeneous domain: *Different dependent stakeholder views without one-to-one mapping between views.*

Model configuration binding time: *bind variation points when a variant-rich model is instantiated/configured to represent a single variant.*

Multi-paradigm modeling *enables the combined representation of a domain model by using different variability modeling paradigms.*

Problem space *is a more abstract representation of the problem domain (i.e. the terminology).*

Single point of variability control: *one point which consistently controls variability throughout various artifacts.*

Software Product Line: *a set of software-intensive systems sharing a common, managed set of features that satisfy 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.

Solution space *describes all possible configurations of implementation components.*

Variability *describes differences between similar products.*

Variation point: *A variation point identifies one or more locations at which the variation will occur.*

Chapter 1

Introduction

The focus of automotive systems changes from mechanics to embedded electronic systems as illustrated in Figure 1.1. Hardware is not the main cost driver in automotive development anymore, instead software and electronics are growing in importance. They are the main source of innovations [7]. An enormous number of safety critical, real-time functionality has to be developed for a complex, distributed environment. This requires more efficient and systematic development methods like reusability, generic architectures, code generation, etc.

1.1 Motivation

Reduced time to market, high quality requirements, complexity of domains, the reduction of development costs and obeying different safety and quality standards are challenging for many industries nowadays. Although applicable to various domains this work has a strong focus on development of automotive control software. Especially the automotive domain as a multi-disciplinary domain covering mechanics, E/E (electrical and electronic), embedded software, thermodynamics, and many more has strong requirements on flexibility in the system development process. The surrounding system has many dependencies on the embedded control software. This increases complexity.
Different development strategies are applied in practice. Although often used in literature, single system development (developing each product from scratch) seems to have no practical relevance. Usually, a so called clone & own approach is used, which simply means that an existing project is copied and adapted to the requirements of a new project [8]. This can be reasonable for a small number of projects. With a growing number of projects this strategy is getting very error-prone and inefficient. Another major drawback is the fact that the correction of bugs or other adaptation have to be done independently in different software projects often leading to inconsistencies.
In order to be competitive, strategies for systematic reuse and generic architecture design should be included in the development process. One development strategy promoting these characteristics is called Software Product Line Engineering (SPLE). The main idea is the definition of a set of supported products (scoping) and the explicit description of variability (differences between these products). Variability comes in two shapes: variability in time, which describes the existence of different versions of an artifact that are valid at different

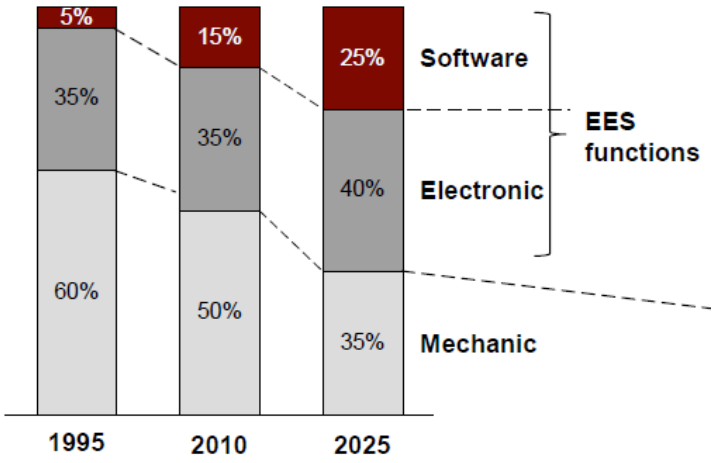


Figure 1.1: Trend towards embedded software from 1995 to 2015 [1] showing the growing share of embedded electronic systems (EES) in automotive systems development.

times and variability in space, which describes the existence of an artifact in different shapes at the same time [2]. This work focuses on variability in space. Current literature on software product line engineering often considers software requirements as the main drivers for platform development [2, 9, 10]. Therefore, software requirements are often used as basis for the specification of the problem description. This is legitimate for the development of COTS software. Nevertheless, in recent years the requirements on software have changed. Often software is embedded in a more complex technical device or in a business or development process. In these cases, software requirements can not be used as a single source to describe the problem domain, since the surrounding system or process, respectively, has a huge influence on the resulting software system.
This work will be applied using the example of an embedded automotive control software. The software is responsible for the control of the peripheral system. Thus, the system influences software and vice versa. Following from this, different domain views are part of the problem. This results in a paradigm shift of thinking about basic activities like domain analysis and domain modeling, which is illustrated in Figure 1.2. What exactly is part of the problem description depends on the specific stakeholder structure. This means that different views as well as different levels of abstraction may constitute the problem space.

1.2 Problem description

Section 1.1 reveals several existing challenges in automotive software development processes:

- Software has to be mass-customizable in order to stay competitive.
  Building each product from scratch is economically unreasonable. A clone & own approach performs better for a very small number of products, but is very error-prone and also inefficient in the long term. The introduction of explicit variability in an

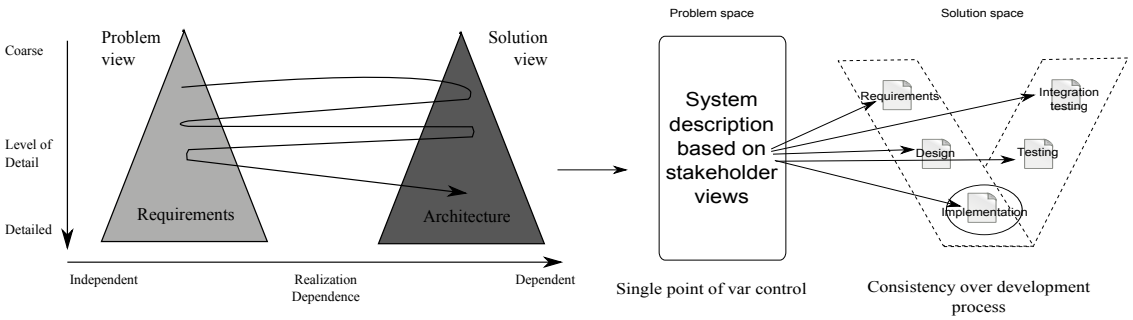


Figure 1.2: Shift from current predominant view in literature [2] showing software requirements as problem description (left) vs. system requirements as problem description for complex embedded systems (right).

inherently complex domain further increases complexity. This complexity has to be handled in some way.

- Heterogeneous domain.
  Various periphery views influence the software. This results in a series of domain views with different characteristics and different stakeholders.

- A diverse tool landscape makes the consistent control of variability difficult.
  Common development processes use a series of different tools. Often there is no integration between the tools in order to exchange information.

The main objective of this work is to provide strategies and means for the development process of generic system architectures. The explicit representation of variability increases complexity, requiring strategies to keep complexity on a manageable level. Another challenge is the diversity of the tool landscape used across the entire development process. Variability has to be handled consistently throughout various tools and artifacts.

1.3 Contribution and significance

This section summarizes the main contributions:

1. **Problem space description using multi-paradigm variability modeling for heterogeneous domains.**
   Coupling of feature-oriented domain models and graphical domain-specific models.

   (a) **Reuse of existing technology.**
   Existing tools provide advanced constraint-checking mechanisms, that can be used in this context.

2. **Reduction of single-model paradigm representation complexity**
   The representation complexity depends on the selection of the modeling paradigm.

   (a) **Modeling paradigm selection**
   Approach to support the selection of an appropriate modeling paradigm.

3. **Ensure consistency over the entire development process.**
   Implementation of a single point of variability control.

1.4 Organization of the thesis

The remainder of this thesis is organized as follows. Chapter 2 describes relevant related work and basic terminology from software product line engineering, existing strategies for the improvement of domain representations, and a comparision with similar projects. Chapter 3 proposes an advanced development strategy for generic software development. Chapter 4 applies the proposed concepts using the example of automotive control software. Chapter 5 concludes this thesis and provides an outlook on possible future work.

Chapter 2

Related work

This section first introduces important terminology and provides definitions for concepts which are used throughout this work. It gives a basic understanding on software product line engineering and related concepts. The second part of this section summarizes some related projects and identifies the differences to the presented work.
Note: Discussing all relevant and related work from literature would go beyond the scope of this work. This section makes no claim of being complete, but discusses a selection of important work.

2.1 Software Product Lines

Modern software demands higher quality and shorter time-to-market cycles at lower costs, making reuse more important than ever before. The idea of reuse is not new and has evolved over time. The latest development attempts to make reuse of software more systematic. One approach propagating systematic software reuse are software product lines (SPL). The core idea is to build multiple products from a single infrastructure in a way that is aligned to stated business goals. An often used definition from Northrop and Clements [3] describes a software product line as *"a set of software-intensive systems sharing a common, managed set of features that satisfy 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."*

Basically, software product line development consists of two fundamentals:

- The differentiation of domain and application engineering and
- the separation of commonalities and variabilities in domain engineering.

Mass customization (the ability to efficiently customize a product for specific customer needs) is one main aim of software product lines. To realize this, variability has to be made explicit and managed in an effective manner.

Product line adoption is still a challenge in practice. Existing strategies can be classified into three groups [11]:

Proactive
The product line is built from scratch with all possible products considered before startup. This strategy causes high upfront investments.

Reactive
The product line grows incrementally with each new product. This approach causes less upfront costs.

Extractive
The organization uses existing software systems and extracts the common and variable parts. This approach is useful if the organization already offers products with many similarities.

2.1.1 Essential activities

Software product line engineering can be summarized to 3 essential activities: *Core Asset Development (domain engineering), Product Development (application engineering)* and *Management*. Figure 2.1 illustrates these activities and the relationships between them. The interdependent arrows indicate that they are all highly iterative. Core assets are used to develop products and feedback from product development affects core asset development.

Figure 2.1: Essential activities in Software Product Line Engineering [3]

All of these activities are equally important. Overall success depends on the development processes as well as on management support, which builds the organizational and strategic frame [3] for the product line. For that purpose, the management has to develop a Marketing and Product Plan [12], which requires the analysis of the market to thoroughly scope the product line.

In the following, core asset development is referred to as domain engineering and product development as application engineering.

Domain engineering - Development for reuse

"Domain engineering is the process of software product line engineering in which the commonality and the variability of the product line are defined and realized" [2].
This definition distinguishes between domain definition and domain realization. One important task of the domain definition phase is *domain scoping*. Scoping identifies and defines the focus of development for reuse, which is of great importance from an economical point of view. A too narrow scope leads to inefficient reuse and a wide scope may result in a waste of resources [13]. In distinction to other reuse approaches, software assets themselves contain explicit variability. Traceability links between these artifacts can be used to facilitate systematic and consequent reuse. Ideally, they enable one to take e.g. a requirement and identify all related implementation code and test cases [14].

Application engineering - development with reuse

"Application Engineering is the process of software product line engineering in which the applications of the product line are built by reusing domain artifacts and exploiting the product line variability" [2].

2.1.2 Domain modeling in detail

A domain model defines *"the functions, objects, data, and relationships in a domain"* [15].

Separation of problem and solution space

Czarnecki and Eisenecker [16] propose to differentiate between a problem and a solution space. The problem space specifies the problem from a more abstract point of view and the solution space describes various technical realizations.

Variability

Variability is defined in terms of features, variation points and variants. Features are described as "*end-user-visible characteristics of a software system*" [15]. *"A variation point identifies one or more locations at which the variation will occur"* [17]. Variants are the possible alternatives defined for each variation point.
Variation points can be regarded as delayed design decisions [18] which may have different properties. First, a variation point can be implicit. This means, that the design decision is not identified, but is accidentally left open [19]. In other words, the decision has not been deliberately left open [20]. If the design decision is identified and intentionally left open, it is said to be *explicit*. Other authors refer to explicit variation points as available [16]. This work always refers to explicit variation points.

Binding A variation point provides several possible variants which can be chosen for a concrete product. At the moment a specific variant is selected, the variation point is said to be bound. The binding time is defined as *"the point in time when the decision upon selection of a variant must be made"* [14].

The binding time has an important influence on the flexibility of a system. If a variation point is bound too early, flexibility of the product line artifacts is lost. Late binding, on the other hand, is costly.

Binding time There are many classifications for variability binding times in current literature. The simplest is the classification in compile-time, link-time, and start-up time [14]. A similar approach is the division of the system configuration into 3 main steps: Compiling, linking, loading. Before, during, and after each of these steps variants can be bound. Examples for binding mechanisms before compilation time are code generation, aspect-oriented programming, and model-driven approaches. For configuration at compile time, precompiler macros and conditional compilation may be distinguished. Precompiler macros are actually evaluated before compilation. In the case of conditional compilation, commands are given via parameters.

Link time binding can, for example, be implemented by the use of a Makefile. Depending on the given parameter, certain compilations and linkages are performed. A configuration file can represent all files that have to be loaded together and, thus, realize different variants at load time. At runtime, components may register their interfaces and access points in a central registry [2].
Krueger [21] gives a good summary and overview of different binding times and their corresponding mechanisms.
Another classification, especially for automotive embedded systems, has been introduced by Fritsch et al. [22]. The authors distinguish 4 different binding times: Programming, Integration, Assembly, and Run Time. Beuche and Weiland [23] introduce a binding time for model-based development. Binding at *ModelConfigurationTime* means to *"bind variation points when a variant-rich model is instantiated/configured to represent a single variant."* This work mainly focuses on *model configuration binding time* (*ModelConfigurationTime*).

Domain modeling paradigms

Several paradigms have been proposed for domain modeling in practice. The most important ones are described below.

Feature-oriented domain modeling In feature-oriented domain modeling, features describe common and variable parts of a problem domain. One big advantage is the fact that it can be understood by both, customers and developers [24].
Kang [15] first proposed the use of features to represent the problem domain with the concept of *Feature-Oriented Domain Analysis* (FODA). A feature model consists of a hierarchical representation called feature diagram and composition rules, such as mutual exclusion (excludes) and mutual dependency (requires). Commonality can be described in terms of *mandatory* features and variability in terms of *optional* and *variant* features [10].
Over the years, several extensions to this original approach have emerged. Basically, they are used to simulate the modeling capabilities of domain-specific modeling. Cardinality-based feature modeling [25], for example, supports the instantiation of features. In this extension, a feature can be annotated with cardinality, which indicates the number of possible clones of this feature in a product. It is also possible to combine features into

feature groups and define cardinalities for entire groups. Extensions are not considered in this work.

Domain-specific modeling Domain-specific modeling uses domain-specific languages for the specification of domain concepts. *"A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain"* [26].

Basically, two types of computer languages can be distinguished: *domain-specific languages* and *general-purpose languages*. Domain-specific languages provide better solutions for a smaller set of problems. Because of their narrower problem space, they provide higher expressiveness [27] and can be used to generate products directly from these high level specifications [28]. The problem is described using a language with domain-specific notation. This notation is an important factor to improve productivity [29]. The idea of domain-specific languages is not new. In fact, the first programming languages have been application-specific. Research in this field has been intensified in the last years [27].

Ontology-based domain engineering An ontology is generally defined as *"an explicit specification of a conceptualization"* [30]. The use of an ontology for knowledge representation has many advantages compared to prior approaches. First of all, it uses a formalized representation of the common terminology of a domain. This overcomes the problem of feature models, which mostly lack of formalism due to the absence of a strictly defined meta-model [31].
Falbo et al. [32] state that an ontology can promote a common understanding among developers. In their approach, they use the ontology in the domain analysis process. This results in a domain model representing the problem space. Czarnecki et al. [33] investigate the relationship between feature models and ontologies. In their opinion, feature models are a semantical subset of ontologies, because ontologies have a higher descriptive power. But again, these observations are restricted to the problem space. Matcha et al. [34] claim the lack of semantics in current feature modeling approaches. They propose an ontological approach by using Jena[1] to build the ontology programmatically.
Ontologies are not covered in this work, because there seem to be little practical relevance.

Common variability language (CVL) Haugen et al. [35] describe a separated language approach for specifying variability in domain-specific language models. They propose a Common Variability Language (CVL) and corresponding variability resolution mechanisms embedded in the OMG[2] meta-model stack. This allows for the description of variability in potentially all MOF[3](MetaObject Facility)-based languages, including UML, as well as MOF- and UML profile-based domain-specific languages. Although this represents a general-purpose, clean approach for handling variability, it is only applicable for MOF-based artifacts. Contrary to this, the current work aims to develop an approach which is applicable to or at least extensible to all kinds of artifacts.

[1] http://jena.sourceforge.net/ontology/ index.html
[2] http://omg.org/
[3] http://www.omg.org/mof/

Decision modeling Following Czarnecki et al. [36], decision models can be traced back to Synthesis, where they are defined as *"a set of decisions that are adequate to distinguish among members of an application engineering product family and to guide adaptation of application engineering work products"*.
Decision models can be represented as *"a directed, connected graph, where each node represents a decision and an edge represents (one of) the next decisions to be made. The direction of the edge defines the ordering of the decisions"* [37]. To generate products, the decision model must be mapped to the product line architecture [37].
One example for this approach is called DoplerVML and has been described in Dhungana et al. [38]. Czarnecki et al. [36] compare decision modeling and feature-oriented approaches. Decision models are an interesting concept, but are out of scope for this work.

2.2 Advanced domain modeling strategies

Recent literature includes a series of investigations towards improved domain representations. This section discusses the most important ones and the main differences to the present thesis.
Elsner et al. [39] propose an approach for constraint checking across arbitrary configuration file types. A constraint checking framework ensures consistency between various types of models and files. This work shows that consistency between different development artifacts is an important issue. Contrary to the present work, the consistency is defined on the level of technical realization and not on a higher level of abstraction.
Holl et al. [40] describe an approach to define configuration dependencies for multi product lines. Multi product lines are defined as *"collections of self-contained and individually developed product lines (PLs)"*. The proposed approach and tool support can be used to define dependencies between heterogeneous systems. In contrast to the current work, the term heterogeneous here refers to different systems instead of different stakeholder views. Holl et al. further introduce a classification of different dependency types. They distinguish between emerging (proposed by end-users) and established (defined in advance) dependencies. Dependencies in this thesis are abstracted to a higher level in order to be tool independent. Another distinguishing factor is the fact that the work by Holl et al. is not based on existing and established concepts and technologies.
Rosenmüller et al. [41] propose a language for multi-dimensional variability modeling. It enables the representation of variability at different levels of detail from domain variability to technical variability and allows the composition of separate variability dimensions. In another work, Rosenmüller et al. [42] describe a method for the configuration of multi software product lines by the use of composition models. In [43], they define a set of dependency types between software product lines. In this case, they are used to describe dependencies between differently configured instances of product lines whenever several small software product lines are composed to a big one. Contrary to the current work, the dependencies are more fine-grained.
Friess et al. [44] describe a concept for model composition of various feature models. The feature models can be created in different tools using different notations. The translation between the different notations is achieved by using an intermediate format. Contrary to the current work, it is restricted to the use of different feature models, but no other

modeling paradigms.
Another approach, describing the connection between feature models and Ecore[4]-based models, has been suggested by Heidenreich et al. [45]. The Ecore metamodel is based on the Essential Meta-Object Facility standard and similar to OMG's MetaObject Facility standard. In their work, they describe the mapping between problem and solution space. The solution space consists of Ecore-based models which are used to represent the implementation in terms of model-based development.
In a follow-up work, Heidenreich et al. [46] compare the aforementioned approach to a second mapping approach called VRL*. Both approaches describe a mapping which could be used as an extension of the current work. The focus of both approaches is on the mapping from problem to solution space, whereas the approach proposed in this thesis focuses on the combination of different representations mainly in the problem space. This is why Ecore-based models in this thesis are mainly used for the definition of a domain-specific language.

2.3 Related projects from the automotive domain

This section gives an overview of some important related projects from the automotive domain which claim to improve reusability.

AUTOSAR The AUTOSAR[5] (AUTomotive Open System ARchitecture) consortium has developed a standardized automotive software architecture to handle various aspects of complexity in the development of automotive systems. The main advantages of the AUTOSAR approach are the separation of hardware dependent and hardware independent software modules and as a result an improved reusability of software components. This means, that it should be possible to exchange software components with OEMs and other suppliers without revealing any implementation details [4]. The AUTOSAR architecture (illustrated in Figure 2.2) is divided into 3 main parts:

- **Basic Software layer:**
  Basic software is used to decouple Hardware and Application Software.

- **AUTOSAR RunTime Environment (RTE):**
  The Runtime Environment provides interfaces and communication mechanisms to decouple applications from the underlying hardware and Basic Software.

- **Application Software layer:**
  Software components are independent from each other. They are also independent from the allocation to different electronic control units. Ports are used for the communication with hardware and other software components. A direct communication is not allowed.

[4] http://www.eclipse.org/modeling/emf/
[5] www.autosar.org/

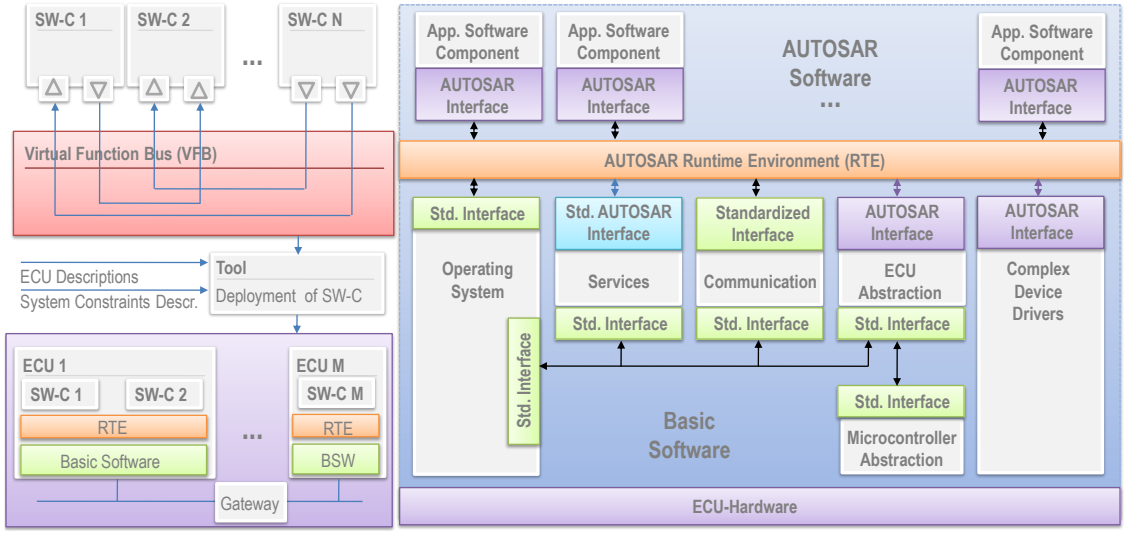


Figure 2.2: AUTOSAR architecture [4]

ATESST2 The Advanced Traffic Efficiency and Safety through Software Technology, Phase2 (ATESST2)[6] project tries to find systematic approaches for information management, architecture, variability, requirements and verification. The result is EAST-ADL2, an Architecture Description Language aligned to the needs of the automotive domain. In further projects it has also been aligned to the AUTOSAR automotive standard.
EAST-ADL2 is organized in 4 abstraction layers as illustrated in Figure 2.3. The most abstract layer is the *VehicleLevel*, represented by a *TechnicalFeatureModel*. This model is the top-level view on the properties of a vehicle. The second layer is called *AnalysisLevel*. This layer represents electronic functionality in an abstract manner. It captures the principal interfaces and behavior of the vehicles' subsystems. This layer is represented in the *FunctionalAnalysisArchitecture*. The *DesignLevel* is divided in a *FunctionalDesignArchitecture* and a *HardwareDesignArchitecture*. These models include the implementation-oriented aspects of software and hardware. On the lowest level, the *ImplementationLevel*, the software architecture is defined by AUTOSAR (see Section 2.3) components.
An UML2 profile and a workbench for EAST-ADL2 are available. Special attention has been given to safety, requirements, timing, and variability. Tool support for these features has been implemented.

CESAR CESAR (Cost-efficient methods and processes for safety relevant embedded systems)[7] is a European funded project from ARTEMIS JOINT UNDERTAKING (JU) with about 60 partners from industry as well as scientific partners. It brings together partners from Automotive, Rail, Avionics & Space, and Industrial Automation.
One goal was to provide a customizable systems engineering platform for industrial use based on a common meta-model called MaxSysML. Three tools are used for variability management, but they are not fully integrated until now. Dubois et al. [47] describes the

[6] www.atesst.org/
[7] http://www.cesarproject.eu/

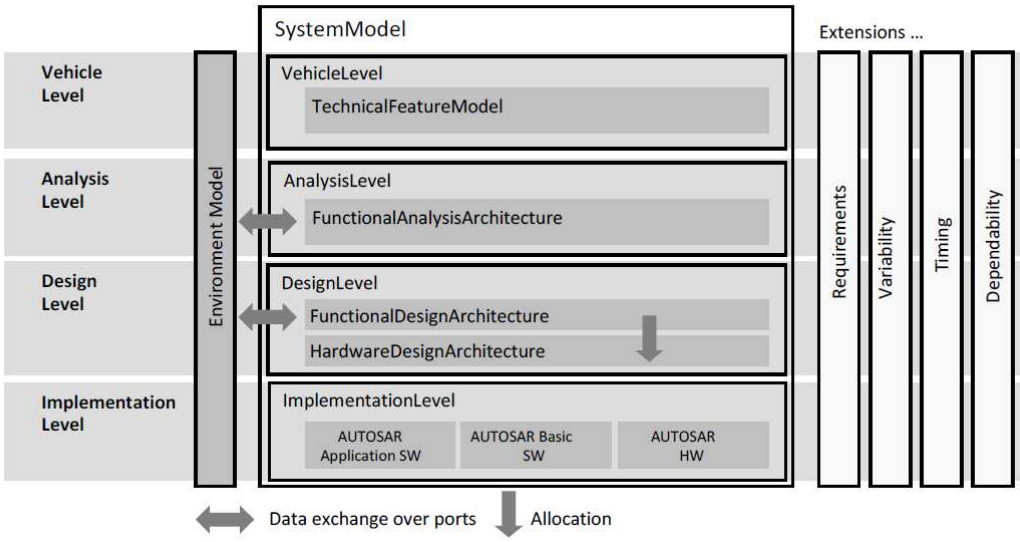


Figure 2.3: EAST-ADL abstraction levels [5]

use of the PLUM tool[8], the CVL tool[9] and pure::variants[10] using the example of radio management for avionics.

Moses & VEIA Two major studies at the Fraunhofer ISST by order of the BMW group are engaged with the development of embedded automotive software. MOSES (German acronym: Modellbasierte Systementwicklung) [48], the first project, aims to develop a methodology for integrated model-based development of E/E systems in vehicles. There is little support for variability in this project.
The follow-up project named VEIA[11] (German acronym: Verteilte Entwicklung und Integration von Automotive-Produktlinien) focuses completely on product lines. The aim of this project is the development of a methodology to support variability in the system development process. First, a reference process [49] has been developed to show the requirements of the industry for such a methodology. The following work packages propose modeling approaches for requirements and architecture and resulted in a tool called aXBench[12]. The major drawback of the VEIA project is the focus solely on the requirements and architecture phase.

2.3.1 Comparing projects

Table 2.2 compares the aforementioned related projects with the current thesis based on the following criteria. It has to be kept in mind that the focus is on variability for all

[8] http://www.tecnalia.com/plum
[9] http://www.omgwiki.org/variability/doku.php/doku.php?id=cvl_tool_from_sintef
[10] http://www.pure-systems.com/pure_variants.49.0.html
[11] http://veia.isst.fraunhofer.de
[12] http://www.axbench.de/

criteria:

1. **Variability management support:**
   In order to support the development of generic software architectures it is essential to explicitly describe variability.

2. **Tool support:**
   Projects often result in theoretical concepts or prototype implementations, but for an industrial application mature tool support is essential. Furthermore, there are well established tools available in many domains. Instead of replacing these tools it is often more advisable to extend the existing tool landscape.

3. **Methodical process integration:**
   Various development processes are used in practice. Is should be easily possible to integrate the proposed concepts in this development process.

4. **System focus/Various views:**
   Various stakeholder views influence the development of embedded control software. All these views need to be part of the problem description.

5. **Scalability:**
   The approach should work with a realistic number of variants in a complex domain.

6. **Multi-domain approach:**
   An advantage of the current work is the independence of a specific problem domain. Although it is described using the example of automotive control software, it can easily be transferred to other domains. Many current solutions are very domain-specific (e.g. ATESST2, VEIA, etc.).

7. **Single point of variability control:**
   One central point (e.g. feature model) which is used to control variability consistently across various development artifacts and tools.

2.4 Summary

The aim of this thesis is the introduction of advanced development strategies for generic software architectures. Generic is here defined as a set of preplanned products. This implies that variability between these products has to be described explicitly. This section first introduced important concepts and terms from the field of software product line engineering. This includes several domain modeling paradigms, which can be used for an efficient representation of variability. The remainder of this section discusses various related projects and compares them to the current work following defined criteria. As mentioned before, variability management support is essential for the development of a generic architecture. Therefore, it is a mandatory requirement in this context. Mature tool support will be provided by the use of existing and proven technology whenever possible to ensure practical applicability with little development effort. Software development follows

	AUTOSAR	ATESST2	VEIA	CESAR	HybConS
Variability management support	yes	yes	yes	partly	yes
Tool support	partly	partly	partly	partly	yes
Process integration	partly	partly	partly	partly	partly (but extensible)
System focus/Various views	no	yes	no	yes	yes
Scalability	yes	yes	partly	partly	yes
Multi-domain approach	no	no	no	partly	yes
Single point of variability control	no	partly	no	partly	yes

Table 2.2: Results of the project comparison (yes - supported, partly - partly supported, no - not supported)

a predefined process. Variability should be supported in various process steps. Furthermore, different stakeholder views influence embedded software and need to be included in a holistic domain model in order to ensure consistency. Many current approaches are very domain specific. Especially in the context of variability management this does not need to be the case, because it can be seen as an orthogonal issue.
Table 2.2 illustrates that none of the existing projects covers all of this criteria. The use of existing approaches and ideas enhanced with new concepts results in an approach which covers the defined criteria at least partly.

2.5 Contribution beyond state of the art

In order to face the current challenges, this work contributes beyond the state-of-the-art as follows:

1. **Problem space description using multi-paradigm variability modeling for heterogeneous domains.**
   Especially for complex, heterogeneous domains the characteristics of the domain are not uniquely fitting to one existing modeling paradigm. The main contribution of this work is the combined representation (further referred to as multi-paradigm modeling) of the two mainly used domain modeling paradigms in order to improve the overall representation. The reduction of domain model complexity is important in many ways. The domain model can be seen as a platform, which is used for the development of many products. A complex domain model often results in error-prone products or inefficient product derivation. Another important aspect is the evolution of the domain model. Typically, domain models grow over time (reactive product line strategy), since not all possible or required variants are obvious from the beginning. This can be caused by the development strategy or simply due to innovations in technology. More details can be found in Section 6.6.

   (a) **Reuse of existing technology.**
   One requirement of this work is the reuse of existing technologies with mature tool support.

2. **Reduction of single-model paradigm representation complexity**
   In order to enhance the representation of heterogeneous domains it is first required to improve single-paradigm modeling. As a prerequisite, the characteristics of different domain modeling paradigms (described in Sections 6.1 and 6.2) have been investigated.

   (a) **Modeling paradigm selection**
   The identification of modeling paradigm characteristics resulted in a method to support modeling paradigm selection. The proposal, described in Section 6.3, is a first practical approach towards a more systematic decision making process.

3. **Ensure consistency over the entire development process.**
   Variability occurs at different stages of development and artifacts all over the development process. As a consequence, various tools have to be capable of representing and resolving variability. This is even more challenging since there are no standardized variability mechanisms (see Sections 6.5, 6.8 and 6.9).

Chapter 3

Advanced development strategies for generic software

The main objectives of this work are strategies to support the development of generic software architectures. In a realistic scenario it is not feasible to provide all possible solutions for a certain problem domain. Instead, the scope has to be restricted to a defined set of solutions. This idea of "genericity" is supported by software product lines.
Software product line engineering comes with a lot of advantages often mentioned in literature (e.g. [50]), but has a lot of disadvantages as well. One major challenge is the additional complexity due to the introduction of variability. This demands for strategies to handle and possibly reduce this complexity. One fundamental concept is the separation between problem and solution space. Following Czarnecki et al. [16], the problem space is a more abstract representation of the problem domain (i.e. the terminology), whereas the solution space describes possible technical realizations (i.e. possible configuration of implementation components). The concrete content of the two spaces is highly dependent on the stakeholders. Depending on who the main stakeholders are, the problem description can either be very abstract or very technical.
Raising abstraction from components in the solution space to a more high-level problem description can be compared to raising the abstraction from Assembler to C, or later from code to model-based system descriptions. Kelly et al. [28], for example, use this comparison for domain-specific modeling. Figure 3.1 compares software product line engineering and Model Driven Architecture[1] (MDA) concepts. The problem space (either single-paradigm or multi-paradigm) can be compared to a Platform-Independent Model (PIM). Both make no assumptions on the underlying technical realization. The middle layer corresponds to a Platform-Specific Model (PSM) in MDA and to a description of one or more technical realizations (e.g. code generators) in software product line engineering. It can be seen as an intermediate layer between the problem description and the concrete solution. This layer is optional in domain engineering. The lowest layer corresponds to the implementation, in this case the variability mechanisms in the specific tools or artifacts.
In contrast to the MDA layers, software product lines consist of another dimension in order to reflect the second development process - application engineering follows the same layers. Each product is first specified by a concrete problem description derived from the domain

[1] http://www.omg.org/mda/

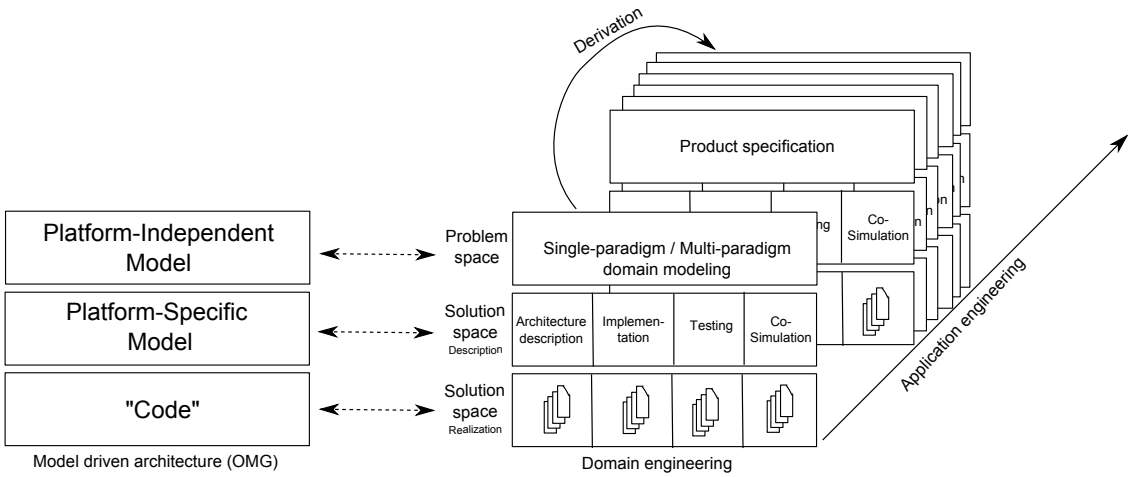


Figure 3.1: Comparison of the different layers of model-driven architectures as defined by the OMG (left) and the SPLE concept on the example of the HybConS project (see Section 6.7)

model. This description is then used to derive concrete products.
The higher abstraction of Model Driven Architectures improves productivity by decreasing complexity. The same can be achieved with software product lines.
Another advantage of the separation between problem and solution space is the possibility to implement a *single-point-of-variability-control* as shown in Section 6.5. This is especially helpful for complex domains with a lot of cross-dependencies. Various development artifacts have to provide variability mechanisms, which can be connected to the *single-point-of-variability-control*. This makes variability consistently controllable.
A third advantage is the support of different binding times. Section 2.1 mentions the importance of binding times to increase flexibility. Separating variability control and variability representation facilitates the implementation of binding times.

Summarized, this leads to the following requirements:

- Raise level of abstraction,
- improve the high level representation, and
- ensure consistent variability control throughout the development process.

Figure 3.2 shows this thesis in the context of the Adoption Factory Pattern [6]. This pattern has been chosen, because it is a convenient, high-level roadmap to illustrate product line adoption from different views and perspectives and it builds a valuable frame to categorize different parts of the current thesis. The focus here is on the "Establish context" part, discussed in Section 3.2, and "Establish production capability" described in more detail in Section 3.3. The "Operation of PL" is out of scope for this work.

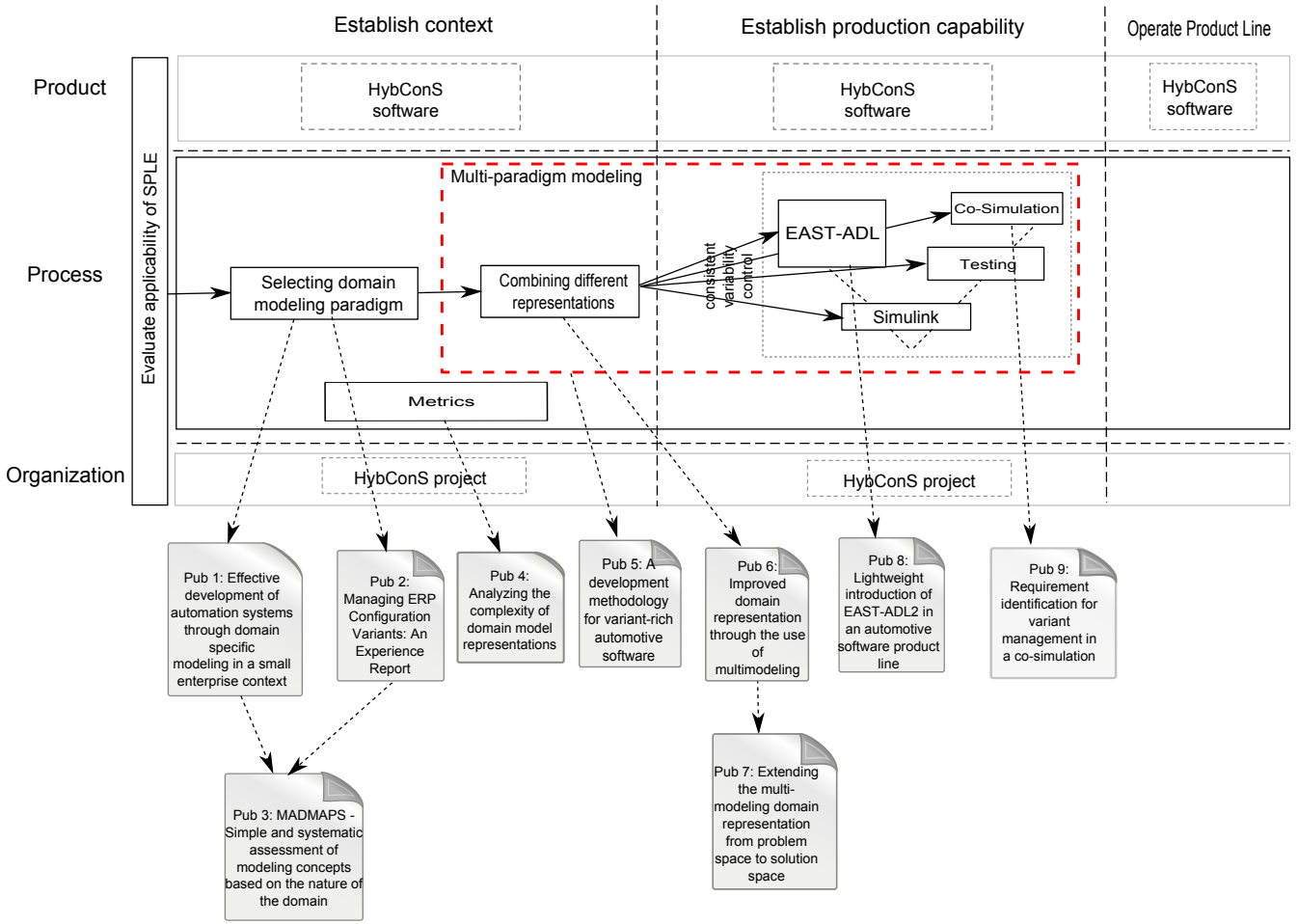


Figure 3.2: Embedding the proposed development strategies and corresponding publications (Section 6.1 - Section 6.9) using the example of the automotive domain in the Adoption Factory Pattern [6], which describes different views and perspectives of a roadmap towards successful product line adoption.

3.1 Hypotheses

Hypothesis 1: Choosing a "wrong" modeling paradigm results in high representation complexity.

A variety of modeling paradigms has been developed in order to solve different problems. From this it follows that they are tailored to specific application scenarios. Nevertheless, there are no explicit suggestions when to use which modeling paradigm. If one chooses a paradigm which does not perfectly fit to the current problem, a higher representation complexity can be assumed.

Hypothesis 2: The overall representation complexity of heterogeneous domains can be reduced by a combined representation in a multi-paradigm model enabling an improved description of various, independent domain views.

Usually, various stakeholders are concerned with the development of systems resulting in multiple views which make up the problem description. It can be assumed that, especially for complex domains, characteristics of these views are diverse justifying a combined representation with different, appropriate modeling paradigms. If the representations of single views are improved, it can be assumed that the overall representation will be improved as well.

Hypothesis 2.1: It is possible to use existing technology for the implementation of the multi-paradigm framework.

There has been a lot of development effort in the field of software product line engineering so far. Various tools and methods have been proposed in order to solve different problems (e.g. domain modeling paradigms, constraint definition and checking frameworks, etc.). As a consequence, it should be possible to reuse and combine existing technologies in order to get a solution for multi-paradigm variability modeling without inventing a completely new language.

Hypothesis 3: Consistency, application domain independence and various binding times can be supported by an abstraction of variability descriptions to a more high-level single point of variability control.

Variability can either be included directly in the language or be described orthogonal to the realization. One advantage of an orthogonal description is the possibility to configure various technical realizations consistently. Furthermore, integrated variability solutions are often very domain-specific. Providing a high-level, general variability description ensures independence of a specific problem domain or tool. Binding times can be described easier, if variability descriptions and technical realizations are separated.

3.2 Establish context

This phase *"paves the way for the product line adoption by determining the scope and associated business case, ensuring the necessary process capability, and performing the necessary organizational management tasks"* [3]. Two different aspects are covered here. First, the selection of a modeling paradigm and second, the representation of heterogeneous domains.

3.2.1 Selecting a variability modeling paradigm

Hypothesis 1: Choosing a "wrong" modeling paradigm results in high representation complexity.

One output of the "Establish Context"-phase is a well-scoped domain model. Domain models are an integral part of a software product line, which makes the choice of a domain modeling paradigm an important decision (see Section 6.3). This decision is often taken implicitly without a systematic decision making process. Nevertheless, domain characteristics are diverse by nature. Modeling paradigms support different domain characteristics and might be completely inapplicable for others. Problem descriptions of various product lines have been investigated in the scope of this thesis resulting in distinguishing characteristics. The product lines describe diverse domains (ERP system configuration, fish farm automation, logistics automation) and are implemented with different modeling paradigms. The identified characteristics are used to put the decision making on a more systematic basis. Section 6.1 and Section 6.2 describe the investigated domains in more detail and Section 6.3 introduces the resulting decision making support.

Identified domain characteristics

The investigation of domains resulted in four distinguishing characteristics:

- *Proportion of fixed and variable connections*
  In general, domains are described by elements and connections between these elements. Some connections are fixed, which means that they are part of each product. More interesting are connections that vary between different products. The proportion of fixed and variable connections, as defined in Formula 3.1, can be used as one distinguishing criteria.

$$|connections_{fixed}| \geq |connections_{variable}| \tag{3.1}$$

- *Several instances of elements*
  Instantiation of elements means that any number of instances from an element (e.g. feature in a feature model with cardinality > 1) can be part of a concrete product. This is trivial if all instances are equal. More difficult is the description of different configurations for each element instance, i.e. if a feature has a cardinality greater than 1 and each feature instance has to be configured independently. This is not supported by feature-oriented modeling without the use of special extensions. Czarnecki et al. [25] describe an extension to represent instantiation in feature models. This

extension has several disadvantages including the resulting high complexity. Moreover, the extension has only been implemented in a prototype tool. Instantiation in domain-specific modeling is very natural, since language elements can be used in different contexts.

- *Different binding times/views*
  It seems to be easier to realize different binding times in a feature-oriented approach, because it is defined on a much higher level of abstraction (i.e. almost everything can be defined as a feature). Different views can easily be represented in one feature model. As a consequence, dependencies between different views can be described as simple requires/excludes constraints in the feature model. Domain-specific languages support multiple views as well, but usually this makes the language much more complex than necessary.

- *Target groups*
  The selection of a domain modeling paradigm also depends on the user of the domain model. The expertise of the target group for the domain model has to be evaluated in order to select the appropriate modeling paradigm.

Section 6.3 gives a more detailed description of these characteristics.

Selecting a modeling paradigm

Based on the identified characteristics, the current work proposes a method to support the modeling paradigm selection process. Multi-Attribute Domain Modeling Approach for Paradigm Selection (MADMAPS) is an adaptation of the Multi-attribute utility theory (MAUT), which allows decision finding based on multiple attributes and goals. The original MAUT approach consists of 5 steps: Identification of alternatives, establishment of assessment criteria, determination of criteria weighting factors, assessment of alternatives and the calculation of utility values. These steps have been adapted for the MADMAPS approach as described below. Step 1-3 are one-time tasks during MADMAPS development, Step 4 and 5 are recurring tasks for each paradigm selection process.

Step 1: Identification of alternatives. The two alternatives in this case are feature-oriented domain modeling and domain-specific modeling.

Step 2: Establishment of assessment criteria. The domain characteristics identified above constitute the assessment criteria.

Step 3: Assessment of alternatives. Contrarily to the original MAUT approach, the assessment values are predefined in the MADMAPS approach (see Section 6.3). They give a rating for each criteria on how well it is supported by the respective alternative.

Step 4: Determination of weighting factors. The weighting factors are based on a questionnaire which has to be answered following a Likert scale. This is the only step actually performed by the MADMAPS user.

Step 5: Calculate utility values. The last step describes the calculation of utility values. These values may give a recommendation for one modeling paradigm. If they

do not, this is an indicator that the domain probably is heterogeneous. In this case, the domain has to be split into various stakeholder views.

3.2.2 Coupling variability representations - Multi-paradigm modeling

Hypothesis 2: The overall representation complexity of heterogeneous domains can be reduced by a combined representation in a multi-paradigm model enabling an improved description of various, independent domain views.

Hypothesis 2.1: It is possible to use existing technology for the implementation of the multi-paradigm framework.

Especially in embedded systems engineering, the problem domain incorporates various views (*"a view addresses one or more of the system concerns held by the system's stakeholders"*)[2]. Some examples are software, mechanics, ECU allocation, safety, supply chain, legal constraints, and others. They are not only interrelated, some of them also have different development (e.g. software can usually be developed faster than mechanics) and life cycles (e.g. mechanical parts as for example engines have a longer average durability than electronic control units). A domain model covering various views helps to synchronize cycle times.
It is also very likely that different views have diverse characteristics. This is called a heterogeneous domain here (see Section 6.6). Not all domains consisting of multiple views are heterogeneous by nature. In some cases, views directly correspond to each other. An investigation of two different automation system projects showed that software (one view) can be mapped to specific hardware elements (second view), i.e. if a specific sensor is part of a product the corresponding software needs to be part as well. This means that the software architecture is driven by the hardware architecture. In this case, one modeling paradigm can be used to represent the entire domain, thus the domain is not heterogeneous. For other domains, e.g. hybrid electric vehicles (see Chapter 4), there is no such correspondence between software and mechanics. In order to improve the representation of heterogeneous domains it is often advisable to split the domain and combine different modeling paradigms. This thesis introduces multi-paradigm modeling as an approach to represent heterogeneous domains. As mentioned before, two modeling paradigms are considered here: Domain-specific modeling (graphical) and feature-oriented domain modeling. The multi-paradigm modeling concept abstracts the two modeling paradigms to their most essential parts: elements, connections between elements and properties of these elements. From an abstract point of view, these are the main building parts of domain-specific modeling and feature-oriented domain modeling (and all other modeling concepts as well). Each language or modeling paradigm following this abstract structural rule can be integrated in the multi-paradigm modeling framework. The major challenge is the definition of constraints between different modeling paradigms. Section 6.6 identifies three types of inter-model constraints:

hasElement constraint describes the dependence of an element in Model A on the existence of an element in Model B as illustrated in Figure 3.3.

[2] http://www.iso-architecture.org/ieee-1471/docs/ISO-IEC-FDIS-42010.pdf

hasConnection constraint describes the dependence of an element in Model A on the existence of a connection between two specific elements in Model B.

hasProperty constraint describes the dependence of an element in Model A on a specific property value in Model B.

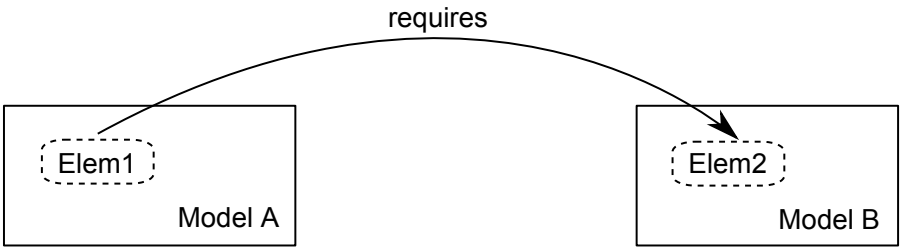


Figure 3.3: Example for a *hasElement* constraint to describe the dependence of elements in different models (see also Section 6.6).

Constraints are defined in a so called Common Domain Model (CDM). The Common Domain Model (see Section 6.6) contains references to elements and connections, which are part of an inter-model constraint. This enables the use of an existing tool providing the required functionality (e.g. constraint checking) for the implementation of the multi-paradigm modeling framework. The concept does not rely on a specific tool, but requires the ability to describe elements, connections and properties and requires means to describe and solve constraints. Pure::variants[3] has been selected as the base tool for the multi-paradigm modeling framework. It provides a powerful Prolog-based language to describe constraints and elements and connections can easily be described as features. The tool is shipped as an Eclipse plugin and thus, is easily extensible. Figure 3.4 illustrates the basic concept, which is described in Section 6.6.

Applying multi-paradigm modeling

Below, the most important steps of the domain and application engineering process are described. The domain engineering process consists of five basic steps:

Step 1: Domain analysis and scoping following approaches described in SPLE literature, e.g. [2].

Step 2: Paradigm selection for each view follows the MADMAPS approach described in Section 3.2.1 and Section 6.3.

Step 3: Representation of each view with appropriate modeling paradigm. Each identified view is modeled with the appropriate single-paradigm modeling approach.

Step 4: Adding references of elements, connections and properties to the Common Domain Model.

[3] http://www.pure-systems.com/pure_variants.49.0.html

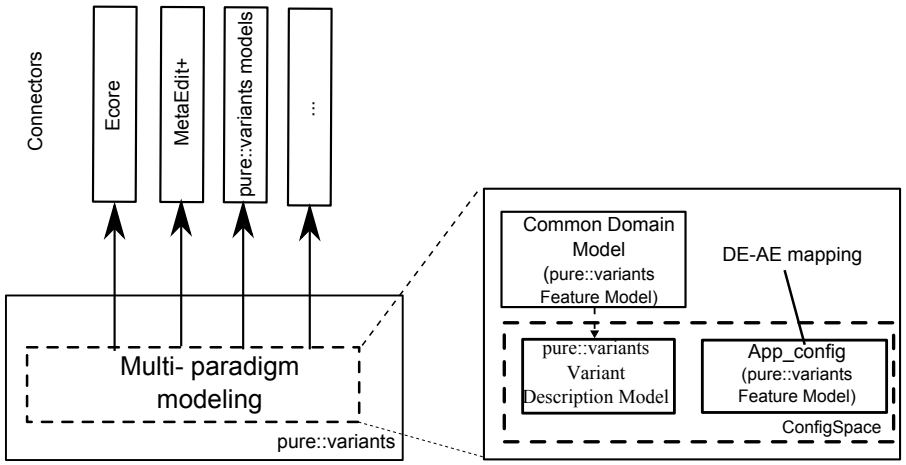


Figure 3.4: Technical realization of the multi-paradigm modeling concept consisting of the multi-paradigm modeling framework implemented in pure::variants and various connectors for the communication with various domain modeling environments (see also Section 6.6).

Step 5: Definition of inter-model constraints as illustrated in Figure 3.5. As mentioned before, the Common Domain Model contains references to the original model elements. Therefore, there is no need for an explicit common meta-model which would probably lead to a loss of information in the transformation step.

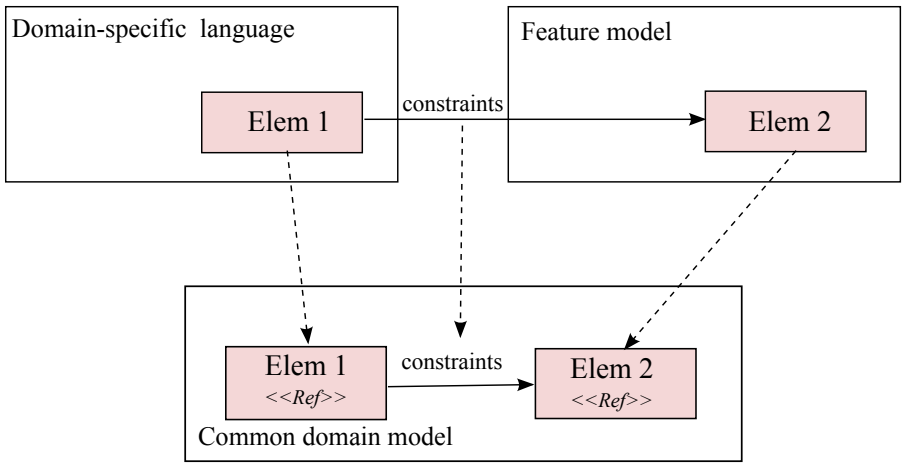


Figure 3.5: Illustration of an inter-model constraint between an element in a domain-specific language and an element in a feature model. The Common Domain Model contains references to the model elements and their constraints.

The application engineering process in the context of multi-paradigm modeling consists of three steps:

Step 1: Creation of application model. Application models can be derived from the domain model as in single-paradigm modeling.

Step 2: Connection of application model to domain model in order to automate the constraint checking mechanism. Application models are defined based on a domain model. In this step, this relation has to be made explicit.

Step 3: Automated constraint checking works as follows: First, the application engineering framework (Configuration space + Variant Description Model)[4] for the Common Domain Model has to be created. This model is then processed automatically:

The constraint checking mechanism iterates over all elements in the Common Domain Model. For each element, it has to be looked up whether or not this element exists in the external application model. If it does, the state of the element in the Variant Description Model is set to selected and to unselected otherwise. If the element (in the Common Domain Model) has an attribute, the value of this attribute has to be looked up in the corresponding external application model as well and set in the Variant Description Model.

For each connection in the variant description model it is evaluated if a connection of this type with the corresponding connection source and target exists. If it does, the state of the element in the Common Domain Model is set to selected and to unselected otherwise. After processing the complete Variant Description Model, the pure::variants constraint checker evaluates the resulting model for validity.

3.3 Establish production capability

Hypothesis 3: Consistency, application domain independence and various binding times can be supported by an abstraction of variability descriptions to a more high-level single point of variability control.

The "Establish production capability"-phase provides the production infrastructure for the software product line. For this thesis does it mean the integration of variability mechanisms in various tools and artifacts, respectively, and mechanisms to consistently control them.
The solution space describes one or more technical realizations of the platform. Usually, there is a defined development process which has to be enhanced with variability mechanisms.
Here, a V-model is used as a sample development process, because this is the predominant process in the case study domain (see Chapter 4).
The main issues are the introduction of variability mechanisms at different stages of the development process (requirements, design, implementation, tests, documentation, etc.), as described in Section 6.5, and to keep variability information for all artifacts consistent. Zooming into Figure 3.2 (red dashed rectangular) reveals the *single point of variability control* concept in Figure 3.6. As an example, Figure 3.6 shows variability mechanisms using the example of Matlab/Simulink.

[4] http://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf

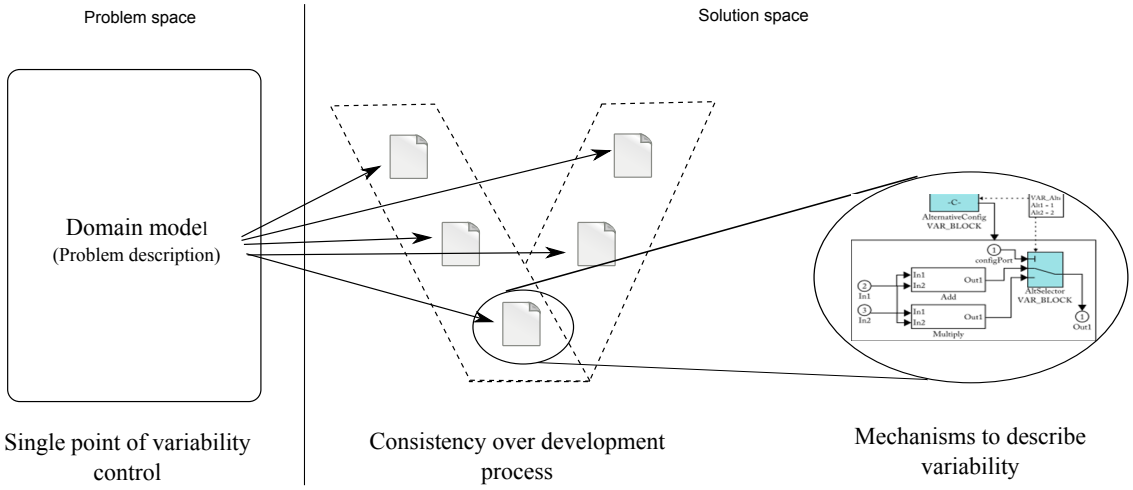


Figure 3.6: Concept of a single point of variability control showing the problem description (single-paradigm modeling or multi-paradigm modeling) on the left-hand side and the consistency over the development process through the use of dedicated variability mechanisms using the example of Matlab/Simulink on the right-hand.

3.4 Summary

This thesis is organized following the Adoption Factory Pattern described in literature [3]. It describes several strategies to reduce or handle the representation complexity arising from variability in an innately complex domain. Not all of the strategies are new. Some of them have been described before in different contexts. The major contributions of the present work are a systematic approach to select a modeling paradigm, means to represent heterogeneous domains while reusing existing technology, and means to ensure consistency over the entire development process.
Chapter 4 applies the strategies proposed in this Chapter.

Chapter 4

Case study - Control software for Hybrid Electric Vehicles

The motivation and case study for this work is an automotive project called HybConS[1]. The overall aim of the project is the development of a generic software architecture for control units of hybrid electric vehicles (HEV) [51]. A hybrid electric vehicle basically consists of at least one electric motor and some other kind of energy source, usually a combustion engine. The main advantages are significant improvements in vehicle performance, energy utilization efficiency, and polluting emissions. Hybrid electric vehicles may vary in different drivetrain configurations (e.g. mild hybrid or full hybrid), different mechanical components (e.g. different types of transmissions), different software-supported functionalities (e.g. pure electric drive), different supported markets (different legal constraints) and more.

4.1 Domain characteristics

An investigation of the domain results in the classification as a heterogeneous domain. It is therefore split in subdomains based on a stakeholder view identification. Stakeholders have different interests and, therefore, different views on the overall system.
The split process resulted in:

- **a software view**
  which describes the control software functionality. Software can either be optional or implemented in different alternative variants.

- **a mechanics view**
  which represents the mechanical system, here the system under control. Different drivetrain configurations or different mechanical elements (e.g. different electric motors, different types of batteries, etc.) have to be reflected in the control software.

- and **a marketing view**
  Vehicles are sold on various markets, as various vehicle types, with different legal constraints and they have to fulfill diverse customer requirements.

[1] http://www.iti.tugraz.at/hybcons

Several other views, which are not part of this example, but are important in practice are as follows: An electronic control unit view which describes the allocation of software functions to control units. Another aspect is the fact that various control units may have different hardware architectures, which of course has an essential impact on the software; e.g. electronic control units may have or have not a floating point unit. Functional safety, as defined in ISO 26262, is an important aspect for automotive control software and can be reflected in a safety view.

From now on, this work will focus on control software (as the main problem), mechanics (the system under control) and partly marketing. The MADMAPS approach, described in Section 6.3, suggests to represent software concerns in a feature model (see Figure 4.3) and drivetrain configurations (mechanics) in a domain-specific language based on energy flows as illustrated in Figure 4.1 and Figure 4.2.

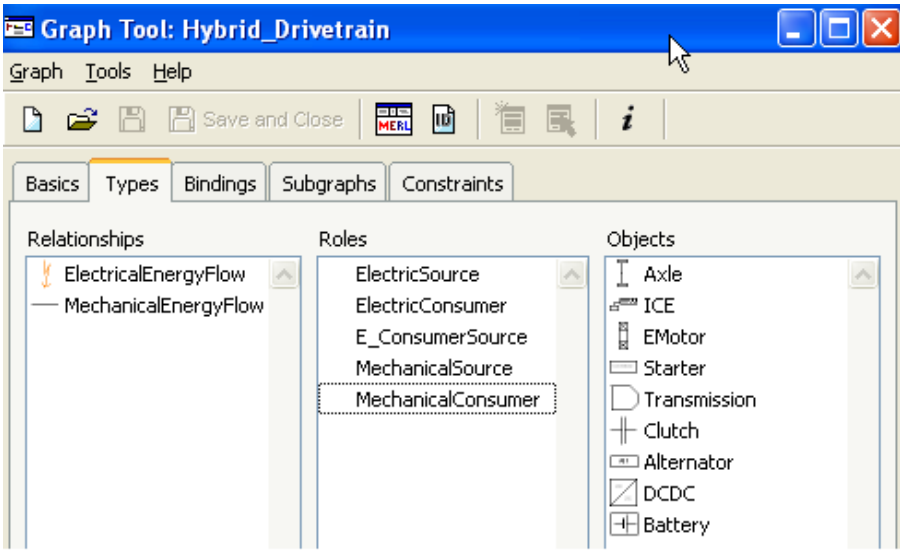


Figure 4.1: Domain-specific language to describe hybrid electric vehicle drivetrain configurations based on energy flows implemented in MetaEdit+. The language consists of several objects, which are the main elements of a HEV drivetrain. Different roles can be assigned to each object. Relationships define connections between objects. Figure 4.2 describes in more detail how bindings can be used to specify valid configurations.

Four different binding times have been defined for this domain:

1. **Model configuration binding time** (mcbt) which results in concrete application models for model-based development.
2. **Code generation binding time:** Variability is bound by the code generator.
3. **Compilation binding time:** Variability is resolved by the compiler.
4. **Parametrization/Calibration binding time:** Variability is bound by fine-tuning parameters, which are generally used in automotive embedded software.

This work mainly focuses on the first binding time.

4.2 Application of concepts

This section describes the application of the strategies proposed in Chapter 3.

4.2.1 Establish context

Hypothesis 1: Choosing a "wrong" modeling paradigm results in high representation complexity.

First, it has to be defined what representation complexity means and how it can be measured. Section 6.4 proposes different metrics to evaluate the complexity of domain representations. These metrics are intentionally kept simple and are mainly used to compare different representations. They again follow the main building parts of model-based development (elements, connections, properties) as mentioned before.

Interface complexity evaluates the complexity of possible element combinations in the domain model. An interface here is either a connection (Relationships in MetaEdit+ or variation points in feature modeling) or a requires/excludes constraint.

Element complexity indicates the number of variable elements.

Property complexity. Elements in both paradigms might have properties. If they are not fixed, they can be used to adapt the resulting application.

Table 4.2 summarizes the respective formulas for domain-specific modeling (focus on MetaEdit+) and feature-oriented domain modeling.

Complexity	DSM	Feature modeling
Interface complexity (C_{if})	$n_{RT} + n_{constraint}$	$VP_{alt} + VP_{or} + n_{constraint}$
Element complexity (C_{elem})	n_{elem}	VP_{Opt}
Property complexity (C_{prop})	n_{prop}	n_{prop}
Overall complexity ($C_{overall}$)	$C_{if} + C_{elem} + C_{prop}$	$C_{if} + C_{elem} + C_{prop}$

Table 4.2: Overview of complexity metrics to evaluate the representation complexity of domain-specific modeling and feature models (n_{RT} - Number of Relationships, VP_{alt} - Number of alternative - exactly one of a group - variants , VP_{or} - Number of or-related - one or more of a group - variants, VP_{Opt} -Number of optional - one or none - variants, $n_{constraint}$ - Number of requires/excludes constraints).

These metrics have been used to investigate different domain models. The results show that the complexity performance of one modeling paradigm is always worse than the other. Furthermore, these metrics have been used to evaluate the suggestion of the variability modeling paradigm selection method (in Section 3.2.1). For the investigated domains the suggested paradigm always results in a domain model with lower complexity.

Hypothesis 2: The overall representation complexity of heterogeneous domains can be reduced by a combined representation in a multi-paradigm model enabling improved description of various, independent domain views.

Section 3.2.2 proposes a multi-paradigm modeling framework which enables a combined representation of various views modeled with different modeling paradigms in one domain model. A simple example illustrates the use of the multi-paradigm model concept:

Step 1: Modeling paradigm selection
Table 4.3 shows the results of the modeling paradigm selection process. In a first step, the entire domain, including control software and the system under control (mechanics), has been assessed following the MADMAPS approach. The result gives no clear recommendation for one modeling paradigm, since the difference between the utility values is not significant. Therefore, the domain has been split in a software and a mechanics view and both parts have been assessed separately. Now, there is a clear recommendation to use feature-oriented domain modeling for the software view and domain-specific modeling for the mechanics view.

	Domain	**DSM**	**FODM**	**Recommended**
1	HEV CU (control unit)	**55**	**49**	-
2	HEV CU - Software	**-38**	**56**	FODM
3	HEV CU - Mechanics	**11**	**-24**	DSM

Table 4.3: MADMAPS utility values for the HybConS domain. First, for the complete domain, second for the software view only and third the mechanics view only (see Section 6.4).

Step 2: Development of independent domain models for each view using the appropriate paradigm.
Based on the results from Step 1, a domain-specific language has been developed to describe hybrid electric drivetrain configurations and a feature model represents a simple distinction between two alternative features *Mild hybrid* and *Full hybrid*. A more complete feature model is shown in Figure 4.3.

Step 3: Adding model element references to the Common Domain Model.
This example shows a simple constraint, which defines that the feature *Mild hybrid* (Feature Model) is only valid if there is a direct connection *MechanicalEnergyFlow* between the Objects *ICE* and *EMotor* (Domain Specific Language). Therefore, a reference to the *Mild hybrid* feature and a reference to the connection *MechanicalEnergyFlow* with connection source *ICE* and connection target *EMotor* have to be added to the Common Domain Model.

Step 4: Definition of constraints between different modeling paradigms.
Figure 4.4 illustrates how the two models can be combined using the multi-paradigm modeling approach.

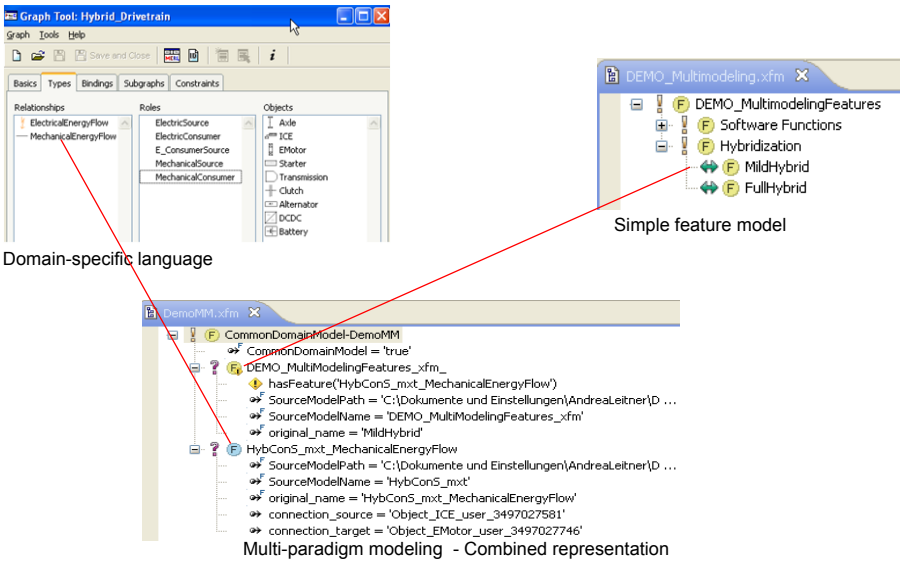


Figure 4.4: Common Domain Model combining a feature model and a domain-specific language on domain level. Feature *Mild hybrid* requires a direct Relationship *MechanicalEnergyFlow* between the Objects *ICE* and *EMotor*.

The main tasks of the application engineering process are as follows:

Step 1: Creation of an application model for each view
Application models can be defined as in single-paradigm modeling. Figure 4.6 illustrates an example for domain-specific modeling as well as for feature modeling.

Step 2: Connection of each application model to corresponding domain model
The multi-paradigm modeling framework implementation requires a configuration for each application. Therefore, application models have to be explicitly related to the domain model they are based on. Without this connection, it is not possible to automatically check the validity of the product. Figure 4.5 illustrates the binding of domain and application models.

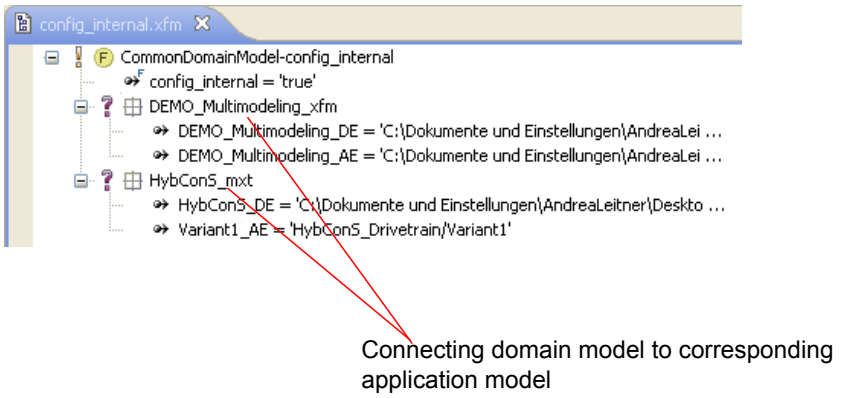


Figure 4.5: Configuration of the multi-paradigm modeling framework for a concrete product. Each application model has to be connected to a domain model. This example shows the binding for pure::variants and MetaEdit+ models.

Step 3: Automated constraint checking

The automated constraint checking mechanism evaluates the validity of the overall system as described in Section 3.2.2. If any constraint is violated, the corresponding element will be marked as shown in Figure 4.6.

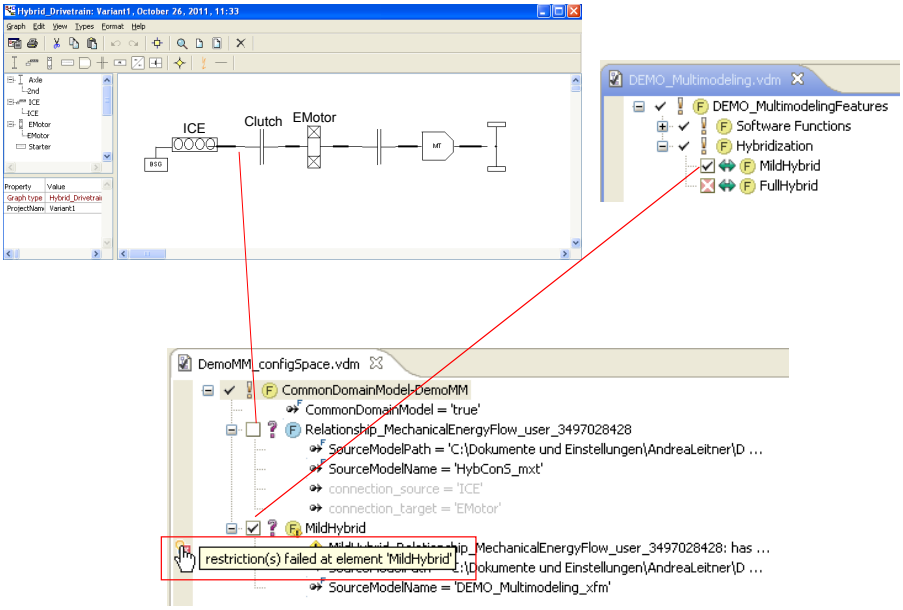


Figure 4.6: A constraint violation has occurred, because there is no direct *MechanicalEnergyFlow* between the combustion engine (ICE) and the electric motor (EMotor), which has been defined as a requirement for a mild hybrid configuration in domain engineering.

Results Figure 4.7 compares the complexity values for 3 different representations: Single-paradigm modeling with domain-specific modeling for both views, single-paradigm modeling with feature-oriented domain modeling for both views and multi-paradigm modeling

(combining both). The chart compares the complexity values for four different drivetrain configurations. This scope is reasonable, because it enables a wide variety of vehicles. Already for the third configuration the representation complexity of the multi-paradigm approach is lower than the single-paradigm representations. Section 6.4 discusses the results in more detail.

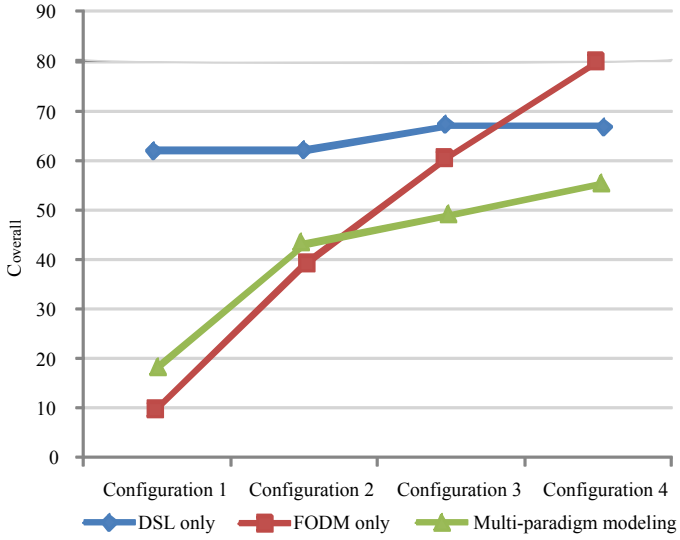


Figure 4.7: Comparison of complexity values for four different drivetrain configurations. Both views (software, mechanics) modeled with single-paradigm modeling with DSM, single-paradigm modeling with FODM, and multi-paradigm modeling (see Section 6.4).

4.2.2 Establish production capability

Hypothesis 3: Consistency, application domain independence and various binding times can be supported by an abstraction of variability descriptions to a more high-level single point of variability control.

Each tool in the development process has to provide variability mechanisms, which can be controlled from an external problem description (single-paradigm or multi-paradigm representation). In the following, the variability mechanisms and the connection to the *single point of variability control* are described exemplary for the automotive development process. Four concrete tools have been connected to pure::variants, the tool which enables the single point of variability control (Common Domain Model) in this example.

Architecture description EAST-ADL2 and AUTOSAR, two standards for automotive architecture description have already been introduced in Section 2.3. Since AUTOSAR is used in practice more and more, and EAST-ADL2, intended as a systematic way to describe the domain, may also be adopted at some point, both concepts should be regarded in the design of the development process. Currently, there is an established development process and especially EAST-ADL2 is not used in practice so far. These are the main reasons for the evolutionary and lightweight approach as described in Section 6.8 and [52].

First, the proposed lightweight introduction of EAST-ADL2 in the development process requires a transformation from AUTOSAR to EAST-ADL2. EAST-ADL2 provides built-in variability management, but tool support is not mature and not all stages of the development process are covered. Therefore, EAST-ADL2 models are treated as any other development artifact, which need to be connected to the single point of control. Basic variability information can be automatically extracted in the transformation step. Figure 4.8 illustrates the lightweight introduction and Section 6.8 explains the details of the implementation.

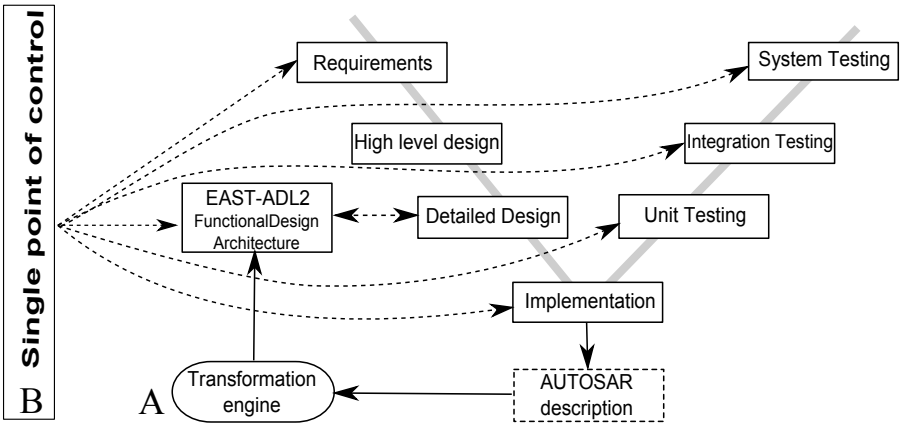


Figure 4.8: Extended V-model development processes illustrating the lightweight introduction of EAST-ADL2 (A) and the connection to a single point of variability control (B) (see Section 6.8)

Implementation Matlab/Simulink is the predominant development environment for model-based development of embedded automotive software. One main goal in the course of this project is the implementation of mechanisms to realize model configuration binding time [23]. This means that it should be possible to reduce a Matlab/Simulink model containing variability to a concrete product model.
Predefined variability template blocks implement variability mechanisms based on existing technology [23]. The *pure::variants Connector for Simulink* provides means to represent variability and support compile time binding and code generation binding. The current work extended this mechanisms to support model configuration binding time as well.
Variation points are used to control variability mechanisms. The *pure::variants Connector for Simulink* can be used to import variation points and to connect them to features (domain engineering). In the transformation step (application engineering), the selected variation point values are propagated back to the Simulink model and can there be used for binding. Model configuration binding is implemented as a Matlab script (*.m file), which removes variability information and disabled parts from the model (subtractive variability approach) [53].

Testing in this case mainly refers to unit testing (testing one unit of composition). In the case of Simulink, a unit under test is a concrete subsystem no matter on which level of abstraction.

If the implementation contains variability, tests have to provide variability as well. Of course, both aspects need to be configured consistently. This has two implications: First, all tests for a specific product can be selected consistent to the implementation. Second, it enables automated testing for a defined set of products in specified time periods in order to ensure the correct functioning at all times [54].

Co-Simulation is a powerful approach for holistic simulations of heterogeneous systems. Different parts of the overall system can be modeled and simulated using an appropriate domain-specific simulation tool. The co-simulation platform handles the coupling of subsystems realized in different simulation tools. Coupling basically means the connection of output and input values of different models in order to exchange simulation results.
ICOS[2] is an independent co-simulation environment developed at the Virtual Vehicle Competence Center. It enables cross-domain co-simulation for a wide range of engineering disciplines in the field of automotive engineering. In concrete, simulations can be performed in various simulation environments and exchange intermediate results via the ICOS platform. Co-simulation supports the automotive development process, since different aspects of the overall system can be simulated in a holistic view in early development phases. The use of co-simulation in a software product line context enables the simulation of the software in a more holistic context. Especially in case of different variants, the impact of changes on the overall system are very useful.
Variability mechanisms in this tool environment are used to handle model variability (substitution of models), linking variability (coupling in- and output parameters), coupling variability (e.g. step size) and boundary condition variability (different initialization values). The sources of variability and the requirements for variability mechanisms are described in more detail in Section 6.9.
For each variability type, a specific variation point has been provided. A domain model defined in this tool environment can easily be imported in pure::variants and connected to the problem description.

Illustrative example - single point of variability control

This example shows the single point of variability control concept on a simple real world example. It demonstrates the consistent control of Simulink-based implementation and co-simulation with ICOS. The Simulink model implements generic hybrid control unit (HCU) software, which supports mild hybrid as well as full hybrid functionality. Figure 4.9 illustrates a schematic electronic control unit network showing the integrative role of the hybrid control unit. Figure 4.10 outlines the implemented drivetrain topology, which is available as a plant model in AVL Cruise[3], a tool for vehicle system and driveline analysis. The AVL Cruise plant model provides predefined routes which can be used for the simulation of typical drive cycles. It can be configured to provide a separation clutch (full hybrid) or have no separation clutch (mild hybrid). It is further possible to change the type (two alternatives: LiIon, NiMH) and capacity of the battery.
The ICOS co-simulation environment couples the Simulink and the AVL Cruise model in order to simulate the performance of different operation strategies in typical driving

[2] http://vif.tugraz.at/en/products/icos/
[3] https://www.avl.com/cruise1/

situations. Simulink as well as ICOS provide variability mechanisms, which are consistently controlled by a feature-oriented representation. Of course, the multi-paradigm modeling approach, described in Section 3.2.2, can be applied here, but for simplicity the example focuses on the single point of variability concept solely.
Two possible scenarios show useful applications:

1. **Alternative high voltage batteries**
   State-of-the-art technology is evolving evermore leading to the availability of various high voltage battery technologies with diverse characteristics.

   Purpose of scenario:
   This scenario demonstrates the influence of the high-voltage battery type on different properties as for example fuel consumption. Lithium-ion batteries have a much higher power density than nickel-metal hydride batteries. This means that for batteries with the same weight, lithium-ion batteries provide more capacity. The overall weight of the vehicle of course has a huge influence on the fuel consumption.

   Variability description:
   The generic design of the software easily enables the representation of different battery types in the Simulink model by simply adapting global parameters. Global parameters are defined in .m-files which are represented in plain text. Existing variability mechanisms for textual representations can be used here. The AVL Cruise model provides two alternative high voltage batteries. In our case, they are given by two alternative plant models. These variants can be described by an ICOS model-exchange variation point.

2. **Switch between full- and mild hybrid implementation**
   Full hybrid describes a setting with the ability to drive purely electrical, whereas in a mild hybrid setting the electric motor is used to support the combustion engine.

   Purpose of scenario:
   Instantiate software with corresponding simulation environment.

   Variability description:
   The software contains two optional modi which have to be deactivated in a mild hybrid setting. A full hybrid topology provides *EDrive* and *ELaunch* functionality which simply means to drive or launch purely electrical (without combustion engine). In this example, we completely remove these modi from the Simulink model if they are not required.

   The plant model is configured with parameters which define whether or not it is possible to open the separation clutch, and define the capacity of the battery by providing different initialization values.

Domain engineering - Building a platform Figure 4.11 illustrates the layout of the product line following the layered structure shown in Figure 3.1. A feature model describes

the problem space and controls two family models (description of variability mechanisms), one for Simulink and one for ICOS. ICOS provides a variability extension called ICOSVM. This extension can be used stand-alone. For the integration in a software product line context, ICOSVM can be connected to pure::variants. The pure::variants family model conforms to this ICOSVM model.
In case of a mild hybrid configuration, two modi have to be removed from the HCU model in Simulink. Two exchangeable AVL Cruise models realize different battery types. The boundary condition server (provides initialization values for co-simulation environment) initializes the models according to the current variant selection.
Figure 4.12 illustrates the basic steps of the domain engineering process for Simulink models. It shows, that first Simulink-based variability mechanisms are used to introduce variability in a Simulink model. This variability information can then be imported in pure::variants and connected to a problem description. Figure 4.13 shows the same for ICOS. Here, variability is first described with the ICOSVM extension, which can then also be imported in pure::variants and connected to the same problem description.

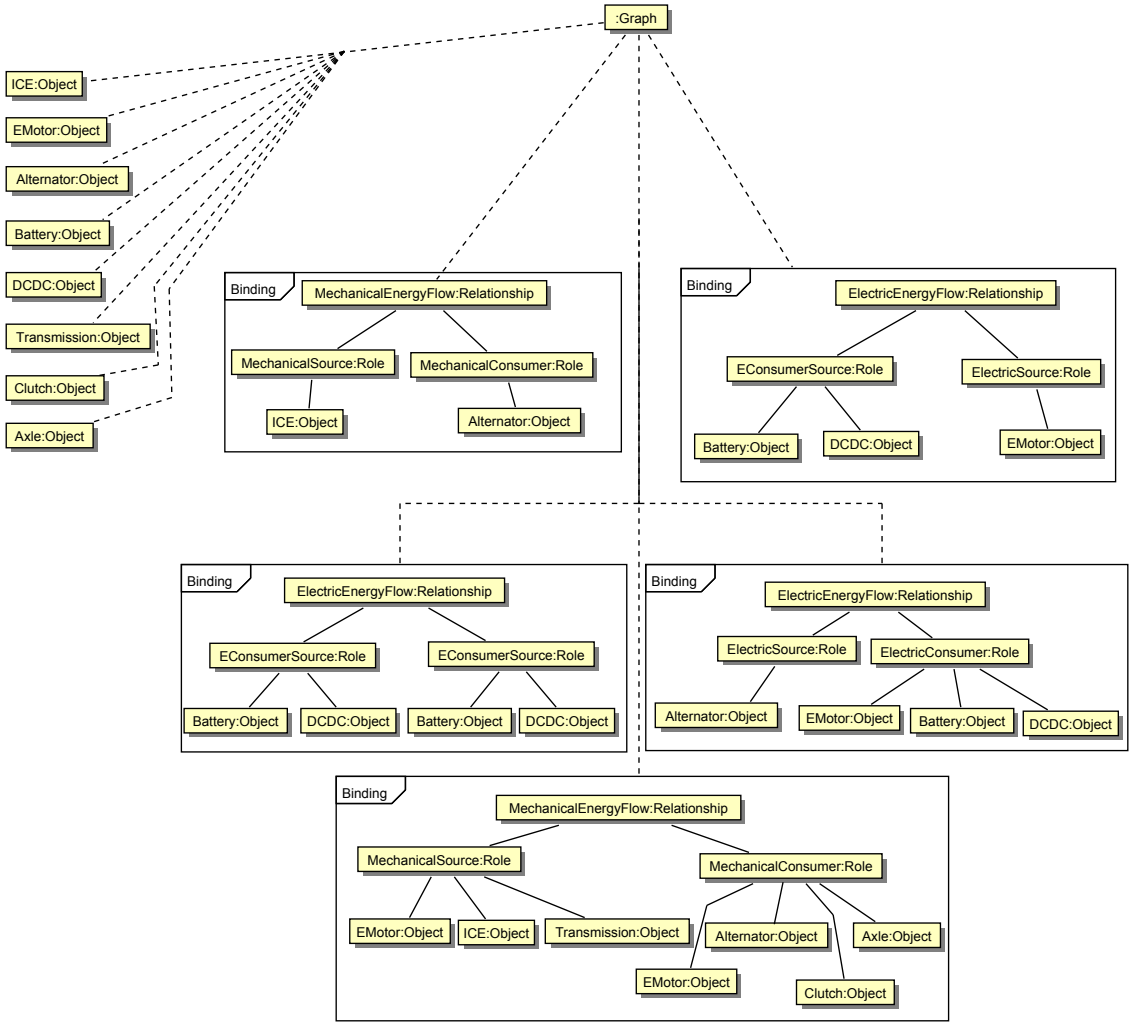


Figure 4.2: Meta-model describing the objects and bindings of the hybrid electric vehicle drivetrain DSL. Bindings consist of a Relationship, which defines the type of a connection, and Objects, which can be connected. Objects take over a role in order to define semantics of the Relationship. E.g. object *ICE* with role *MechanicalSource* can have a connection *MechanicalEnergyFlow* with an object *Alternator*, which has the role *MechanicalConsumer*.

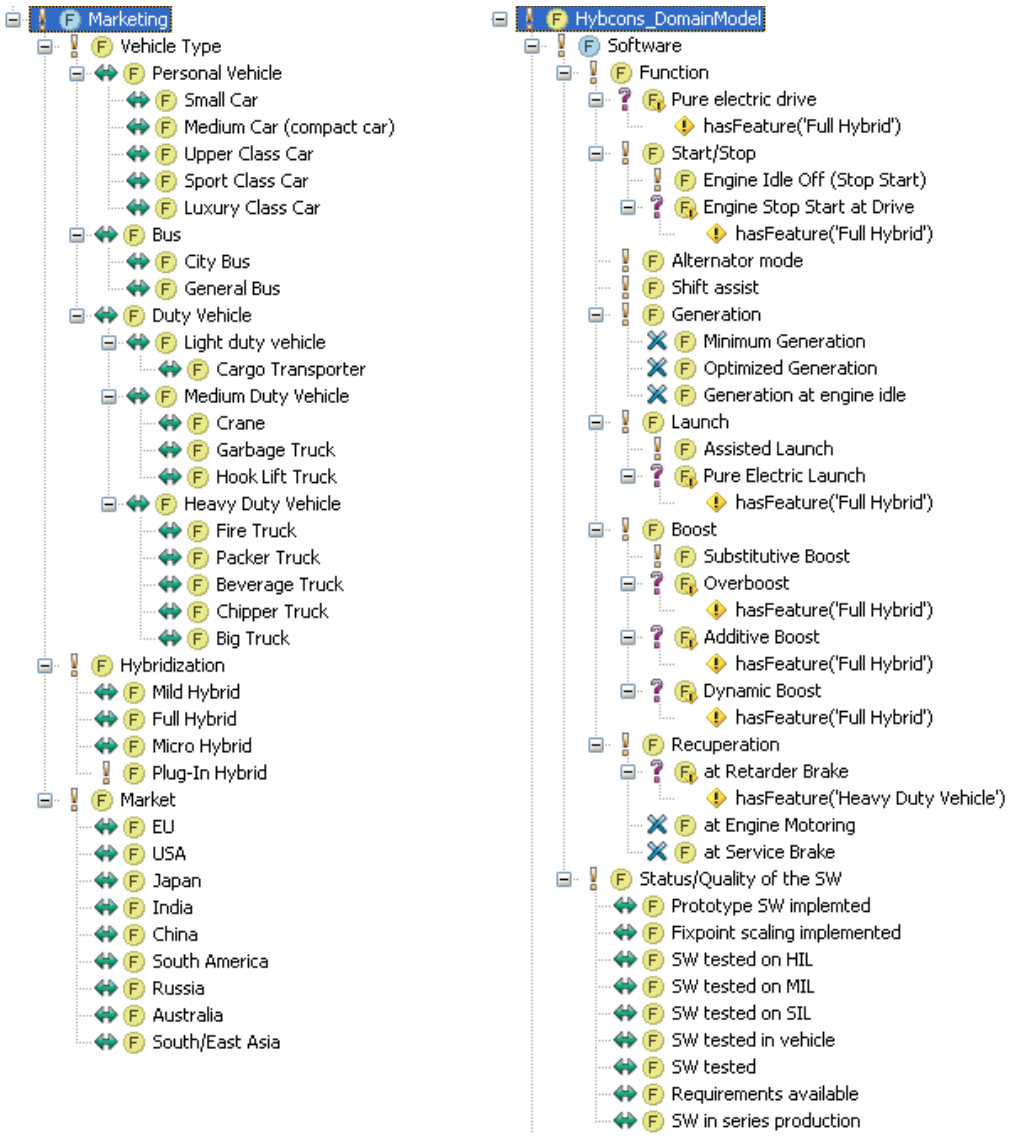


Figure 4.3: Feature model describing management (marketing) aspects on the left and the software view on the right.

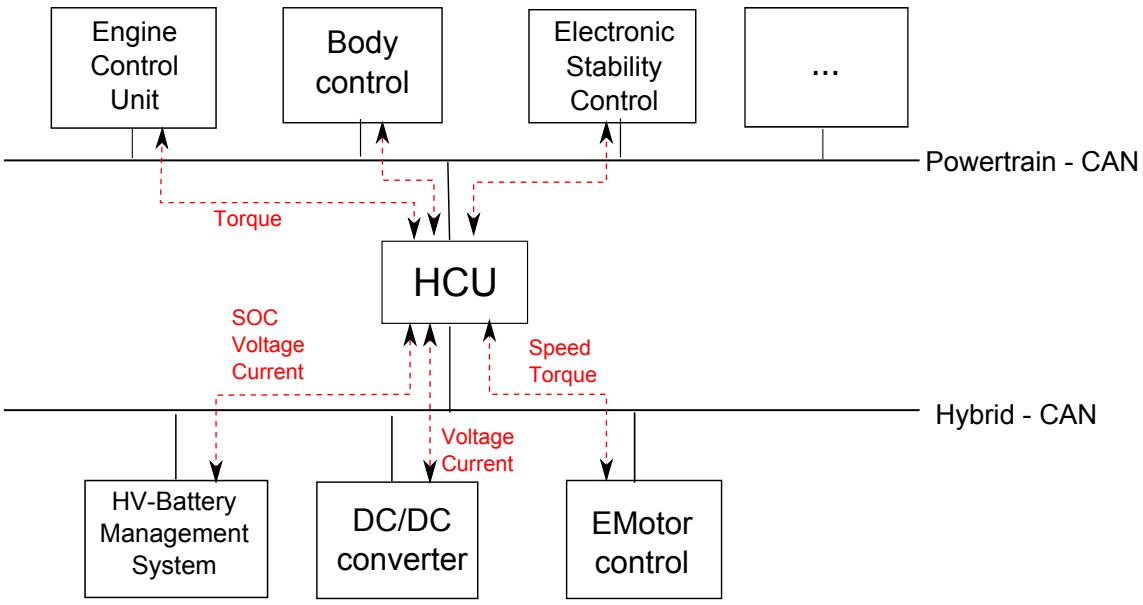


Figure 4.9: Typical electronic control unit network showing the integrative role of hybrid control units (HCU). Various mechanical components are connected to electronic control units, which exchange signals via a CAN (controller area network) bus system. Vehicles have several bus systems. This picture shows two of them (Powertrain-CAN and Hybrid-CAN), which exchange information via the hybrid control unit.

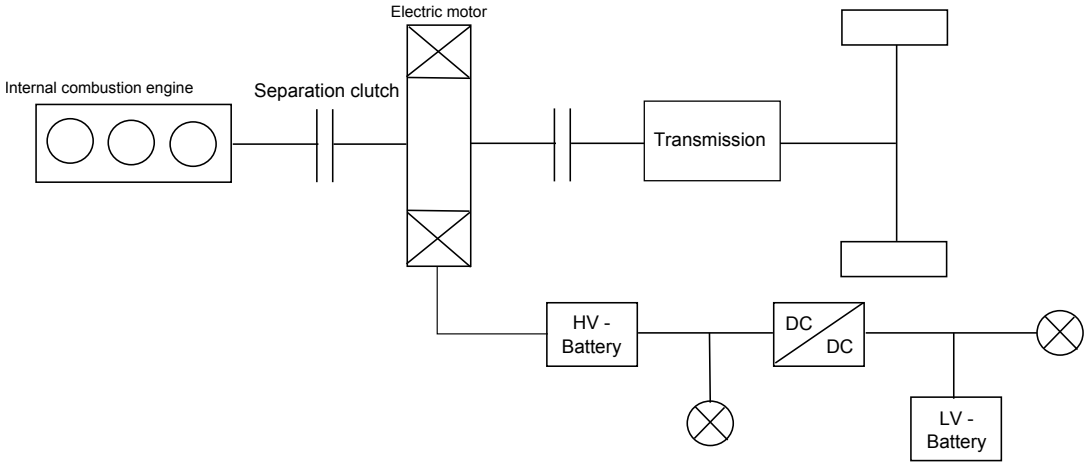


Figure 4.10: Drivetrain configuration used in this example to demonstrate the single point of variability control concept for model-based development with Simulink combined with co-simulation in ICOS. This topology describes a full hybrid configuration because there is a separation clutch which enables decoupling of the two energy sources (ICE, EMotor).

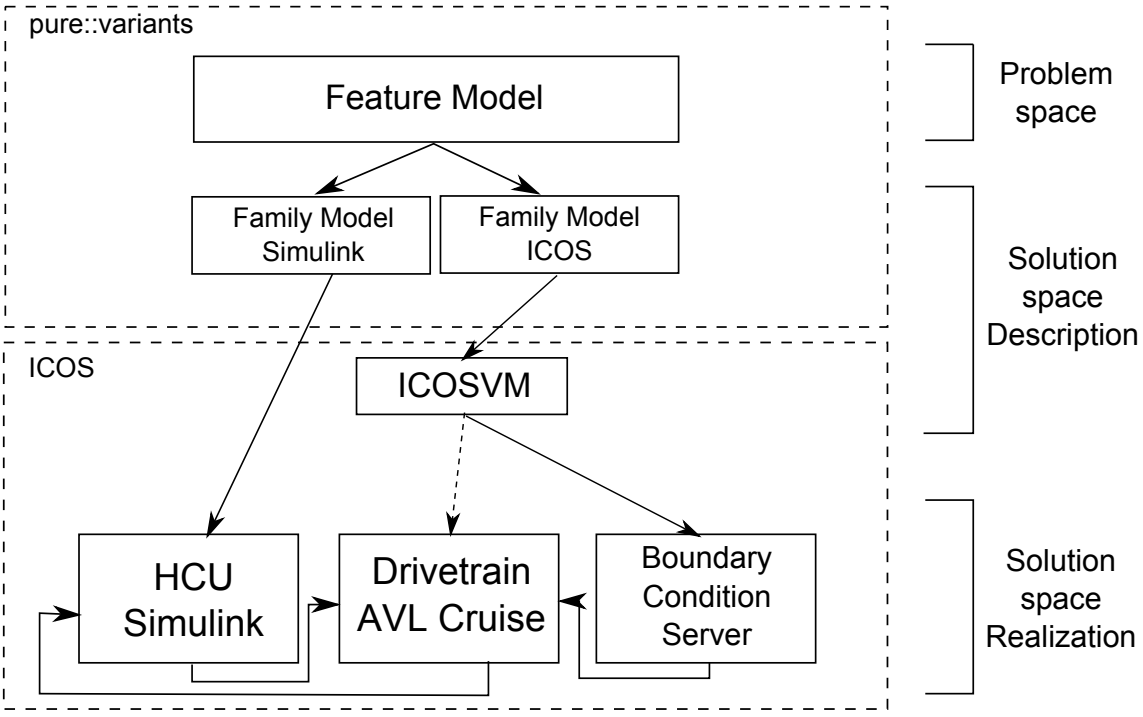


Figure 4.11: Illustrates the basic structure of the product line consisting of a pure::variants and an ICOS/Simulink part. The pure::variants feature model controls variability from a central point. Variability decisions are propagated via family models to the respective technical realizations in ICOS and Simulink, respectively. Simulink models are configured directly, for the co-simulation part only the ICOS environment is configured, not the participating models directly.

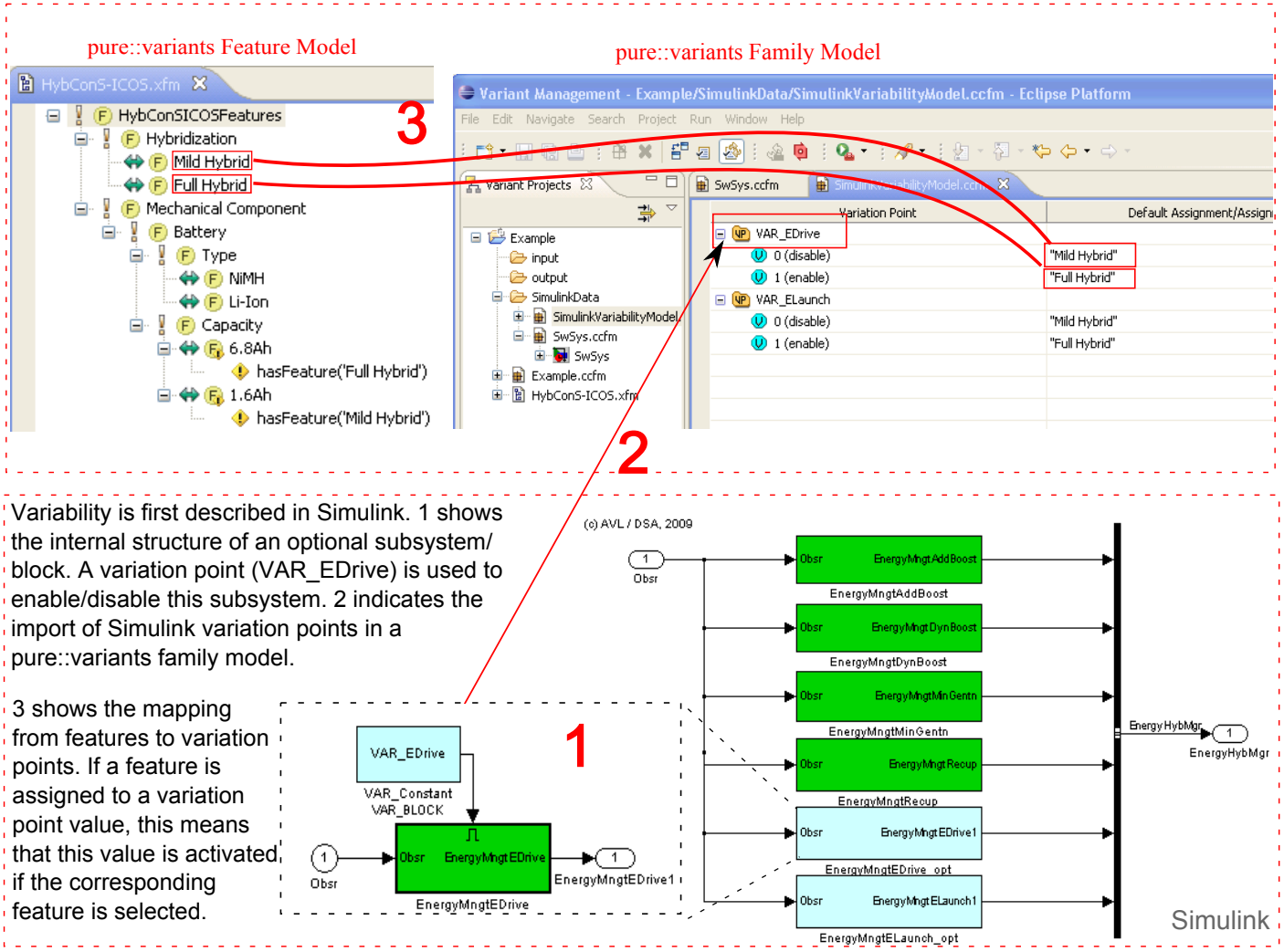


Figure 4.12: Illustration of the domain engineering process for the integration of Simulink variability.

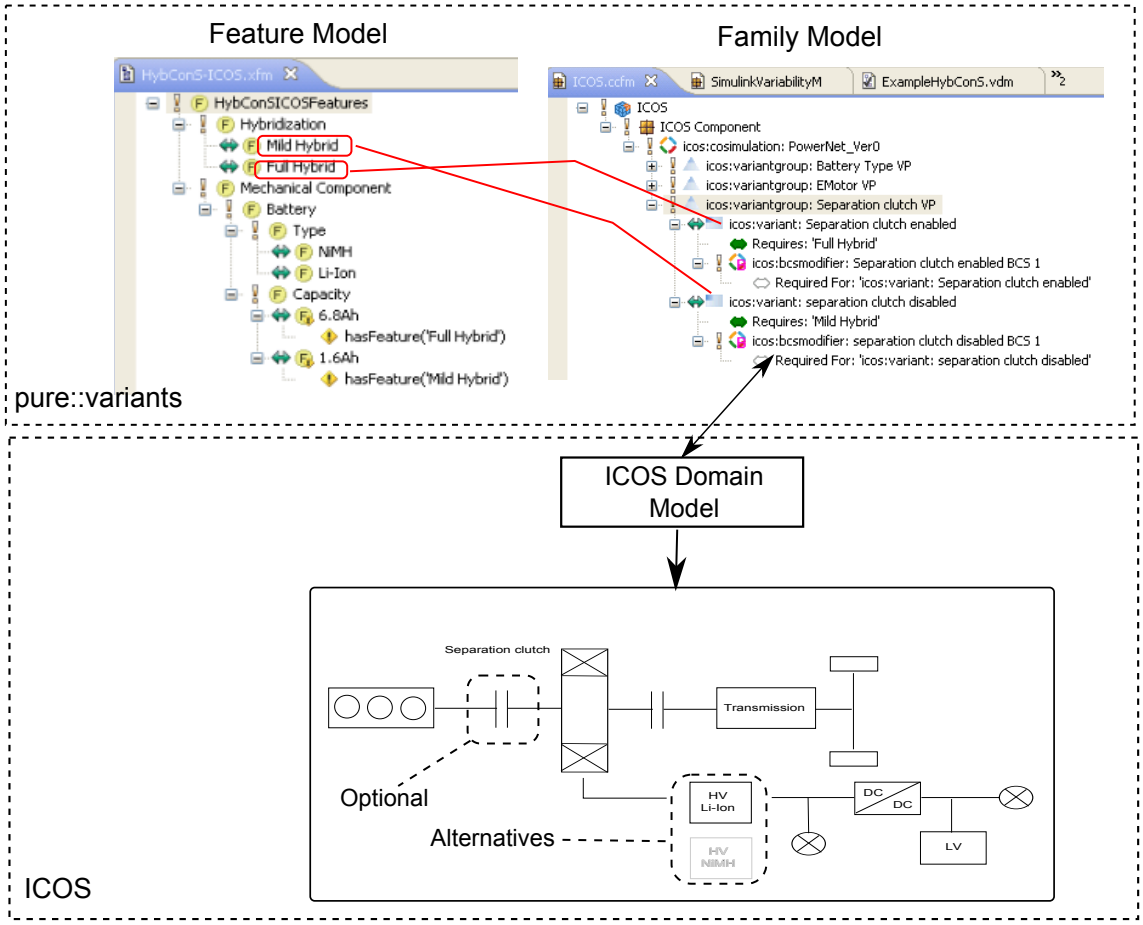


Figure 4.13: Schematic illustration of domain engineering for ICOS. ICOS and ICOSVM are described in XML. To improve understandability, a schematic representation has been chosen here. Variability is first described in an ICOS domain model, which can be imported into pure::variants. Variants can then be connected to features (i.e. if this feature is selected, the corresponding variant is activated.

Application engineering - Building concrete products Figure 4.14 illustrates a product derivation scenario. The user selects variants in the feature model. The selection is automatically propagated to the corresponding family models, which is then propagated to the Simulink and ICOS environments, respectively. Due to the single point of variability control concept, different resulting HCU software models and ICOS environments can be configured consistently.

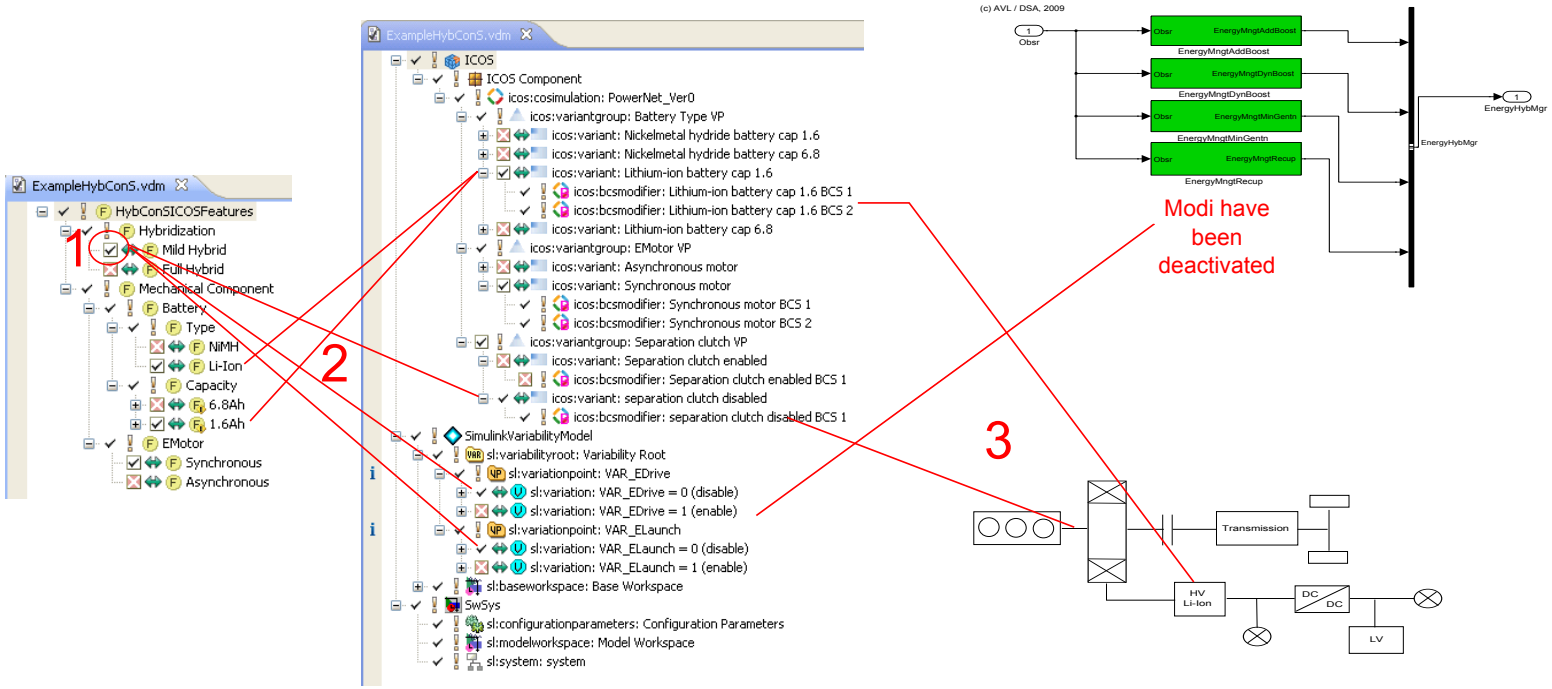


Figure 4.14: Feature selection (1) in the feature model and automatic propagation (2) to the technical realizations (family models). Unused modi are removed from the Simulink model and the plant model is configured as illustrated in the schematic representation (3).

4.3 Summary

Table 4.5 summarizes the four implemented and described variability mechanisms. The described development artifacts have been connected to a common problem description using the multi-paradigm modeling approach. As a consequence, all artifacts can be configured consistently from a concrete problem description. In the scope of this work, a single point of variability control connector for EAST-ADL2, unit testing and co-simulation has been implemented. Additionally, model configuration binding time for Simulink, testing, and co-simulation has been realized.

Dev. artifact	Variability mechanism	Binding time	Product
EAST-ADL2	built-in mechanisms	mcbt	EAST-ADL model
Simulink	Optional subsystems Alternative subsystems	mcbt code gen. compile t.	Simulink model
Unit Testing (Simulink)	Alternative subsystems	mcbt	Simulink model
Co-Simulation (ICOS)	Model-related variability Var. parameter connections Environment variability Coupling variability	mcbt	ICOS project file

Table 4.5: Overview of various development artifacts in an embedded automotive software development process, their variability mechanisms, provided binding times and the product resulting from variability binding.

Chapter 5

Conclusions, Limitations, and Future Work

This chapter concludes this thesis with a discussion on completeness and directions for possible future work.

5.1 Concluding remarks

The strategies introduced in this work are not restricted to the automotive domain but are applicable in many other domains as well.
The main contribution is the improvement of the problem space representation. It raises the level of abstraction and is therefore an important means to reduce complexity and increase productivity. In previous works, there has been no systematic selection methodology for modeling paradigms. This work suggests a simple, practically applicable method to get at least a recommendation for a modeling paradigm. Especially for complex, heterogeneous domains, the description of the problem space can pose a problem. The proposed multi-paradigm modeling approach provides a viable concept to overcome this problem. Contrary to other approaches from literature which identified the shortcomings of modeling paradigms, no new representation language should be invented. Based on existing technology, a multi-paradigm modeling framework has been proposed to describe inter-model constraints. The problem space is important because it is used as a single point of variability control.
The approach has been evaluated using the example of an automotive project.

5.2 Limitations and possible future research directions

The adoption of software product line engineering for practical use still poses a big problem. Few success stories show the benefits of this approach. The present thesis addresses only a subset of current challenges and, of course, has several limitations. This section discusses current limitations and proposes ideas and directions for future work (see also Section 6.7).

Evaluation in other problem domains
So far, the proposed strategies have been applied on the automotive domain and on a

rather small example only. It is challenging to get industry support to evaluate emerging concepts, but at least parts could be shown on real world examples. To get more results and a more in-depth evaluation of the proposed strategies, there should be a step-by-step introduction in industry.
The generic description of concepts makes the strategies domain independent. Cross-domain reusability can be shown by an application in several other domains.

More in-depth study of domain characteristics
Domain characteristics have been identified only on a small set of domains. The main challenge, again, is the availability of industry data. Nevertheless, one investigated product line produces commercial products. A second investigated product line has conceptually been developed with a big company and is based on real world data. The derived methodology is acceptable for a fast evaluation of domains. For a more scientific modeling paradigm selection method, the criteria have to be evaluated in more detail.
Furthermore, the paradigm selection method gives only rough suggestions until now. Of course, this is an improvement to the current situation where there is no systematic support at all. But, it is just a first step which has to be enhanced and evaluated in more detail in order to get definitive propositions. One main limitation is the fact that there is no real threshold to decide whether or not the difference between utility values is big enough for a clear result.

Integration of decision models
Decision models are a promising approach to describe variability of systems. In future research, it should be evaluated whether or not it is useful to extend the current investigation for domain-specific modeling and feature-oriented modeling with decision modeling.

Multi-paradigm modeling

A limitation of the multi-paradigm modeling approach is the need for connectors to communicate with the respective models. Until now, parsers have to be implemented to retrieve the required information. Currently, there are parsers for ecore, pure::variants and MetaEdit+ models. An idea for future work is a framework to generate these parsers based on, for example, XML schemas.

Extension to the solution space
The current implementation mainly focuses on an improved representation of the problem space. To further improve the entire domain model, an extension to the solution space is required. Both, domain-specific modeling and feature-oriented domain modeling, have to be investigated towards the abstraction of variability mechanisms to the main building parts (element, connection, property). In case of pure::variants this is trivial, because family models realize exactly this representation. In fact, all each model containing variation points can easily be transformed into this representation. The main challenge are code generators for domain-specific languages. Until now, there have been no investigations of code generators in this context, but code generators are often implemented following template based patterns [55]. Therefore, it can be assumed that they can be abstracted in the same way.

Evaluate applicability of the approach to support model integration
A common problem with model-based development in general is the integration of different models or model types throughout the development process. It should be investigated whether or not it is possible to use the multi-paradigm modeling approach to improve traceability between different models in the development process.

Automated testing
Some of the proposed concepts could also be applied for the automation of tests. This could be done in several contexts. For example, tests can be configured consistent to the implementation. For a predefined set of products, tests can be performed automatically whenever something has been changed e.g. in the code base. Another scenario is the implementation of variable tests in order to verify a component in different contexts with configurable test cases. Some of these scenarios have been shown for co-simulation, but they sould be applicable to e.g. unit testing as well.

Standardization of variability mechanisms
Until now, there is no standardized variability interface description. This makes it difficult to control variability in different artifacts with a common problem description and, therefore, hinders the evolution of the main strength of a software product line engineering approach - the consistent variability control for a defined set of products. One task for future work is the definition of such a standardized interface for variability mechanisms.

Chapter 6

Publications

This chapter contains publications by the author of this thesis which explain the approach presented in Chapter 3 and the case studies presented in Chapter 4 in greater detail. The following publications are included within this chapter:

Publication 1: *Effective development of automation systems through domain specific modeling in a small enterprise context*, Journal on Software and System Modeling, Springer-Verlag Berlin Heidelberg, 2012

Publication 2: *Managing ERP Configuration Variants: An Experience Report*, KOPLE 2010, First International Workshop on Knowledge-Oriented Product Line Engineering, Reno, USA, 2010

Publication 3: *MADMAPS - Simple and systematic assessment of modeling concepts based on the nature of the domain*, EuroSPI 2012 , EuroSPI Conference - European Systems & Software Process Improvement and Innovation ; 2012

Publication 4: *Analyzing the complexity of domain model representations*, ECBS 2012, 19th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Novi Sad, Serbia, 2012

Publication 5: *A development methodology for variant-rich automotive software architectures*, OVE E&I - Special Issue on Automotive Embedded Systems, SpringerWienNewYork 2011

Publication 6: *Improving domain representation with multi-modeling*, KOPLE 2012, 3rd International Workshop on Knowledge-Oriented Product Line Engineering, Salvador, Brazil, 2012

Publication 7: *Extending the multi-modeling domain representation from problem space to solution space*, 9th IEEE Workshop on Model-Based Development for Computer-Based Systems co-located with IEEE ECBS, Novi Sad, Serbia, 2012

Publication 8: *Lightweight introduction of EAST-ADL2 in an automotive software prod-*

uct line, HICCS-45 2012, Hawaii International Conference on Systems Sciences, Maui, USA, 2012

Publication 9: *Requirement identification for variant management in a co-simulation environment*, SPLC 2012, 16th International Software Product Line Conference, Salvador, Brazil, 2012

Softw Syst Model
DOI 10.1007/s10270-012-0289-1

THEME SECTION PAPER

Effective development of automation systems through domain-specific modeling in a small enterprise context

Andrea Leitner · Christopher Preschern · Christian Kreiner

Received: 29 November 2010 / Revised: 20 April 2012 / Accepted: 20 August 2012

Abstract High development and maintenance costs and a high error rate are the major problems in the development of automation systems, which are mainly caused by bad communication and inefficient reuse methods. To overcome these problems, we propose a more systematic reuse approach. Though systematic reuse approaches such as software product lines are appealing, they tend to involve rather burdensome development and management processes. This paper focuses on small enterprises. Since such companies are often unable to perform a "big bang" adoption of the software product line, we suggest an incremental, more lightweight process to transition from single-system development to software product line development. Besides the components of the transition process, this paper discusses tool selection, DSL technology, stakeholder communication support, and business considerations. Although based on problems from the automation system domain, we believe the approach may be general enough to be applicable in other domains as well. The approach has proven successful in two case studies. First, we applied it to a research project for the automation of a logistics lab model, and in the second case (a real-life industry case), we investigated the approaches suitability for fish farm automation systems. Several metrics were collected throughout the evolution of each case, and this paper presents the data for single system development, clone&own and software product line development. The results and observable effects are compared, discussed, and finally summarized in a list of lessons learned.

Keywords Domain-specific modeling · Small enterprise cost model · Automation system · Software product line · System development process

Communicated by Dr. Jeff Gray, Juha-Pekka Tolvanen, and Matti Rossi.

A. Leitner · C. Preschern · C. Kreiner (✉)
Institute for Technical Informatics, Graz University of Technology, Graz, Austria
e-mail: christian.kreiner@tugraz.at

A. Leitner
e-mail: andrea.leitner@tugraz.at

C. Preschern
e-mail: christopher.preschern@tugraz.at

C. Preschern
HOFERNET IT-Solutions, 9811 Lendorf, Austria

1 Motivation

It is quite usual to develop automation systems (AS) in a traditional way as single-system projects. Approaches such as model-driven development and code generation are rarely applied in practice. This leads to problems throughout the entire system lifecycle. One reason is that domain knowledge is only implicitly available in the heads of project developers. In small companies, there is normally no systematic process in effect for securing domain knowledge, which instead has to be acquired by each developer working on an AS in a given domain. This in turn leads to high development efforts. The maintenance of AS also quite often requires substantial efforts. Interviews with companies working in the AS domain have identified a broader range of challenges beyond pure software development. These are typically related to stakeholder communication and economic limitations in small companies.

Software product lines (SPL) are a viable method for overcoming some of these challenges, but they have the drawback of high upfront costs, which result from initial platform development. This makes the adoption of the new approach a crucial phase. Breivold et al. [4] described a transition process from legacy systems, developed in individual projects, to an

Published online: 16 October 2012

SPL. This approach is well suited for enterprises that have several legacy systems and can cope with the high upfront costs.

1.1 SPLE in small enterprises

Since the high initial investment excludes many small enterprises, there is a need for an incremental approach for the adoption of an SPL [40]. In a small company, the upfront development of an SPL would tie up all resources, which means that it would not be possible to make any profit during this time. Furthermore, such an introduction affects all areas of a company, business strategy, technology, processes, methods, tools and organization [39]. This again poses a problem for small enterprises, where a limited number of people have to take on expert roles in several or all of these areas. In many cases, these topics receive less attention due to a strong, exclusive focus on completing the customer project. In particular, investments in reusability fall victim to this mindset. Critical success factors [39] and recommendations [4] have been described for transition projects towards SPLE in larger organizations. Our case studies revealed that, as in larger enterprises, the minimum required success factors are SPLE and domain knowledge management, as well as SPL orientation from the very beginning on. However, these factors take on different forms in the small enterprise context.

One main contribution of this work is the use of an adoption strategy and an experience report that focus on small enterprises. In small enterprises, employees usually have to handle a relatively complete range of aspects, such as domain understanding, stakeholder communication, business strategy, technology, development efficiency, etc. Furthermore, the combination of Domain-Specific Modeling (DSM) and automated artifact generation can lead to increased productivity, increased product quality, decreased development costs, and decreased maintenance costs. The consistency of models and other product parts (e.g. system documentation) greatly improves stakeholder communication. In fact, communication and liability issues between the AS developer and the electrical contractor were an important motivation for the SPL transition in Case Study II (Sect. 4.2). DSM also helps to secure domain knowledge effectively. The following sections describe some important characteristics of AS and common problems.

1.2 Basic attributes of automation systems

Data concerning the development of AS was gathered via interviews in AS companies. The interviews were based on two questionnaires, one focusing on AS companies and the other on electrical contractors [33]. The aims of the questionnaires were to get an overview of the current situation and

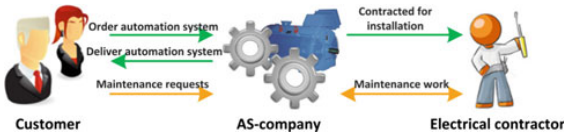


Fig. 1 Common stakeholder setting for automation systems

to gather information on the needs and requirements of these two stakeholders. The most important results are summarized below:

- One basic characteristic of an AS is the fact that hardware considerations and electrical installation are at least as important as software. Often, different companies or different departments within a company are responsible for the development and installation of an AS. In a common AS development scenario (Fig. 1), a customer orders the AS from the AS company. The AS company develops the system and subcontracts an electrical contractor to carry out the electrical installation for the system. The customer only communicates with the AS company.
- An AS company often specializes in a specific kind of AS (e.g. automation of gravel pits, automation of fish farms, logistics automation).
- In many cases, an AS is very domain specific. The domain knowledge has to be acquired from the customer. Getting the necessary knowledge is a difficult and time-consuming task.
- The same employees often develop different products in the same AS family. They build up a significant amount of implicit knowledge about both the domain and the AS family.
- Software errors turn up commonly in AS, especially during installation and the first weeks of operation. This is probably because only limited testing can be conducted beforehand, as the real environment is often not available during software development. On-site testing and debugging can involve additional risks because potentially faulty software is controlling real, physical equipment.
- The AS software often has to be changed due to the installation of new components.
- Often, a so-called “clone&own” approach is used. This is a very basic reuse approach, whereby software is copied from a similar project and adapted to fit the new requirements.

1.3 Problems arising with traditional development approaches

Most AS development environments do not provide features like Unit Tests, variable scoping or object-oriented pro-

gramming. According to [3,8], these features offer important means for increasing software quality. This may be one reason why automation software usually contains more errors than pure software projects. Moreover, this seems to be particularly true for the widely used IEC-61131 Programmable Logic Controller (PLC) languages. As mentioned above, clone&own is often used in AS development. An existing software is copied and adapted based on the requirements of the new system in the same domain. Programmers often forget to adapt certain parameters or fail to adapt the software consistently, which leads to many errors. The experts interviewed agreed that this is a major problem.

Another reason for faulty software and high maintenance efforts is the high complexity due to distributed system installation. Insufficient stakeholder communication can easily lead to misunderstandings, resulting in system integration errors. Especially during on-site installation, many errors occur due to a lack of communication between AS developers and electrical contractors. The additional complexity further hinders maintenance tasks such as error identification, which is already rather difficult in AS because errors can be located in either software or hardware.

Automation systems are often very domain specific, and automation developers have to obtain detailed domain knowledge before designing and implementing the software. They often only implicitly possess the domain knowledge used in the construction of the software. Especially for a small company, this can lead to a dangerous situation where a single employee may possess most of the domain knowledge. This knowledge is a valuable property which is lost if the employee leaves the company. To facilitate the work of succeeding developers, it would be better to make the domain knowledge explicit.

We propose an approach that can overcome the problems described above. Section 3 describes the transition from single system development to a product line approach. Before starting the transition process, appropriate tool support has to be selected, which is described in Sect. 2. Sections 4.1 and 4.2 describe two case studies that followed this approach: the use of an AS product line for logistics and for a fish farm. Next, Sect. 5 compares and discusses the results, and Sect. 6 summarizes some relevant related work. Finally, Sect. 7 provides a conclusion and an outlook for further work.

2 Tool selection criteria

The selection of tool support is an essential factor in Software Product Line Engineering (SPLE). The use of an appropriate tool is particularly essential for the development of a domain-specific language and different code generators. We propose the use of a multi-attribute utility theory (MAUT) analysis [1] to evaluate and select different tools. We used a list of selection criteria in several SPLE projects [15,23,33]. In MAUT, criteria are weighted according to their importance for the domain.

2.1 Criteria for the selection of a DSM approach

Selecting a domain modeling paradigm is one of the first decisions to be taken. In turn, this largely determines supporting tool decisions. In practice, three main domain engineering methods are currently in use: Domain-Specific Modeling [22], Feature-oriented approaches (FODA [17], FORM [19]), and Ontologies [13]. Ontologies are the least common approach and currently there appears to be no mature tool support for ontologically based domain modeling. Feature-oriented approaches and the use of domain-specific languages have been used for a long time. The selection of one of these two approaches is an important decision that does not often receive proper consideration. However, depending on the nature of the domain, this decision can have a significant impact during the lifecycle of domain models [25].

Some characteristics would seem to require a DSM approach:

- *Few specialized meta model elements* DSM seems to be best suited for applications or domains that consist of only a small number of different elements.
- *Focus on architecture instead of functionality* Describing parts and their relations is more important than describing specific functionality, especially in terms of variability. Figure 2 shows a more detailed representation. The main point of this illustration is to show the different foci of the approaches. In a feature-oriented approach, the structure is fixed in a tree-like manner. The focus here is on the features that are supported by a common system architecture. For DSM, in contrast, the focus is on the domain-specific system architecture, which is modeled

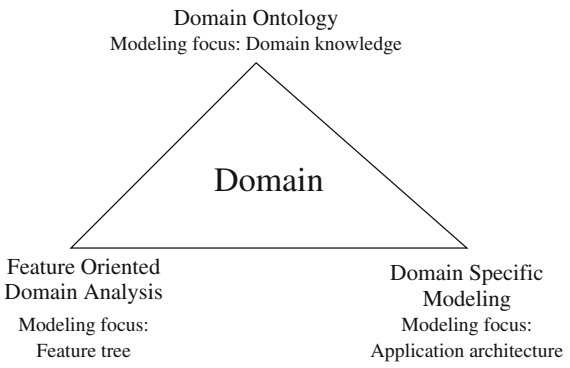


Fig. 2 Domain modeling approaches and their main focus

from elements and constraints defined in the DSL meta model. The ontological approach poses no such restrictions on modeling (i.e. neither a domain-specific common architecture, nor a common meta model). The focus is on the representation of domain knowledge in general. Ontologies for domain modeling can be seen as a higher abstraction. This approach is currently less popular.
- *Several instances of components* A DSM approach supports the use of several instances of a single component.

All these characteristics are valid for most automation systems. Therefore, we argue that a DSM approach is very likely the right choice for domain engineering in the automation system domain.

2.2 DSM tool requirements for AS

We identified several major requirements for DSM tools used in the automation system domain.

The ability to draw a graphical model with the designed DSL is essential for our approach. It makes it easier to specify the location of the hardware components in the automation system. This location information is used whenever it is necessary to generate project-specific documentation. This documentation is needed, since multiple companies are often involved in an automation project. A consistent documentation can serve as a good communication base for different stakeholders.

Furthermore, the tool must allow for the specification of connections between objects. In automation systems, there are commonly multiple connections from and to an object. For example, a sensor component can have more than one output and may have to be connected to several other objects in the model. A convenient concept that enables this kind of modelling is needed.

Another requirement is to allow changes after software generation. Since several parts of a specific PLC software are not general enough to be considered in the DSM, little adaptations or functions sometimes have to be implemented manually. These manually implemented code snippets must not be overwritten if a newer version of the PLC software is generated from the model.

It is also important that any potential changes in the DSM do not affect previously produced systems. Existing models should still be fully functional after minor changes.

2.3 Developing DSMs with MetaEdit+

We conducted the tool selection MAUT analysis based on given requirements. The analysis revealed that MetaEdit+ would be the best option for the two AS case studies covered below. The MetaEdit+ suite provides tools for developing domain-specific languages (DSLs) and using them in a software product line. Two main parts of MetaEdit+ are the Workbench and the Modeler. The Workbench can be used to construct a DSL, which is then used with the Modeler to construct specific system models. DSL elements and their possible connections are defined with the Workbench using a concept called GOPPRR (Graph-Object-Property-Port-Role-Relationship), which is described in [38]. GOPPRR defines the basic meta-types which can be used to construct the DSL. Code generators for objects and their connections can be developed using either the MERL script language or alternative approaches accessing the model via SOAP. The MERL language provides powerful features for navigating through a model [42].

3 SPL transition process: considerations for a small enterprise

3.1 Viewpoint architecture: typical AS project setting

Figure 3 illustrates the main stakeholders and their interests in the AS context, based on the 4+1 architecture presented by [20].

The customer requirements represent the logical view. Common approaches for gathering requirements include regular meetings with the customer and checklists, such as those proposed in [34]. These requirements can be used to construct most of the use cases, which are basically scenario descriptions.

Both the hardware components and the final AS software are linked to the physical view. The main stakeholder for this view is the electrical contractor, who is responsible for the installation of the hardware. Research regarding the different types of hardware that might be used in combination with the AS must be conducted before the software is developed.

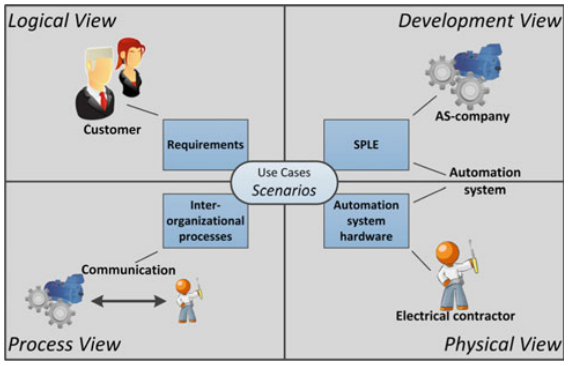


Fig. 3 Viewpoint-architecture for automation systems

The software development approach is presented in the development view. Frequently, many systems from the same automation domain are produced. Therefore, we recommend an SPLE approach (Sect. 3.3). The primary advantage of SPLE is that the explicit storage of domain knowledge in a meta-model enables the systematic reuse of this knowledge, which leads to increased software quality [29].

The process view describes interactions between parallel processes. The main stakeholders here are the AS company and the electrical contractor. Both work on the same AS, which makes it necessary to coordinate their efforts. To prevent misunderstandings and to provide a common basis for communication, we recommend the automatic creation of an electrical installation plan for the AS (Sect. 3.3). This support minimizes the number of errors that might occur during the hardware installation process.

3.2 Cost structure

Basically, one can distinguish between domain engineering (platform) costs and application engineering (project-specific) costs. Figure 4 provides an overview of the different types of costs and their classification.

- *Domain engineering (DE) costs* cover the development of the framework for the SPL. This includes the effort required to gather the domain knowledge, identify variability and create a DSL out of this information. It further includes the effort required to develop the initial PLC software template and a code generator for the specific domain.
  Domain engineering costs occur mainly at the beginning of an SPL project and therefore lead to higher upfront costs than other development approaches. If a certain number of products are produced, the additional upfront investments pay off very quickly. Of course, this is a very rough estimation, since domain evolution costs occur during the whole lifetime of the SPL.
- *Application engineering (AE) costs* include all expenses for a new product that is derived from an existing SPL. In general, this includes costs for the creation of a model describing the concrete system. This model is used as input for source code generation. In the proposed cost model, software adaptations are one part of the variable costs. These costs can also occur in the clone&own approach, where software has to be adapted according to new requirements. For AS, additional costs arise for the on-site and electrical installation. Electrical installation changes originate from misunderstandings and lack of communication between the developers and the electrical subcontractor. The goal is to keep these project-specific costs low.

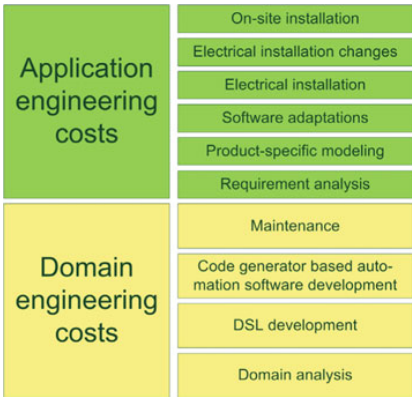


Fig. 4 Cost structure for software product line-oriented AS development with systematic reuse

3.3 Domain-specific modeling

The SPL is the core aspect of the proposed architecture. It is a very useful approach for achieving various business goals. Development work time and the corresponding variable costs are low for systems derived from an existing product line [2].

One approach to implement an SPL is based on domain-specific modeling (DSM). DSM enables the description of a system on a high level of abstraction through the usage of a DSL [22]. For that, a DSL for a given domain has to be designed, which then enables easy and efficient development of application models for the domain. A code generator is then used to generate source code from the specific domain model. In the case of AS, in addition to the automation source code, a consistent hardware installation plan is generated. This leads to higher upfront investments in terms of development work time for code generators, but also reduces installation time and errors and improves stakeholder communication. Figure 5 shows the different layers of modeling, based on the OMG 4-layer architecture.

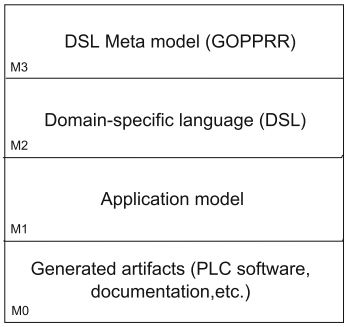


Fig. 5 Four meta model layers of domain-specific modeling

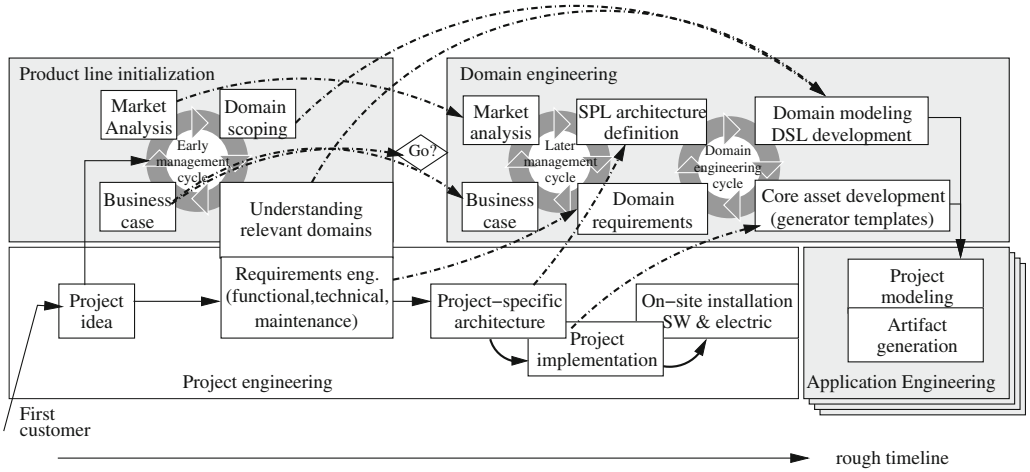


Fig. 6 Initializing a product line for an automation system domain

3.4 The transition process in detail

We propose an approach for the adoption of an SPL approach in small enterprises. The relevant literature makes it clear that the introduction of an SPL is a risky task, especially for a small enterprise. Although the proposed approach was verified in the automation systems domain, we believe it could also be applied in any other domain with similar characteristics. The proposed approach is deliberately lightweight and therefore well suited for small enterprises, since small enterprises could easily fail if they cannot afford the potentially high upfront costs characteristic of general transition approaches. Figure 6 illustrates the proposed concept. First of all, the approach is divided into the well-known domain and application engineering processes. Parallel with the domain initialization phase, a first project is started. We suggest carrying out a customer project that can be used as a basis for the domain. The project initiation comes from the customer, who has specific requirements. In parallel, the AS company has to decide if there is a potential market for the product. The remainder of this section explains the approach in more detail.

3.4.1 Project management

Compared to the management of single system projects, the management of a software product line engineering (SPLE) process is more complex. The management of an SPLE project is positioned closer to the strategic level than the management of a single system project.

This is even more true for SPLE in a small enterprise. Usually, a small enterprise serves a single specialized domain, as opposed to many domains. Therefore, the introduction of SPLE is a purely strategic decision in the context of a small enterprise, which shows the importance of making a careful decision.

The business goal is to produce high-quality end-products and to develop a framework which enables the efficient production of these products. The aim of the SPLE management is to develop universally reusable components, which can be used to assemble the product in an effective way. Usually, a project is a temporary venture with a defined input and output. In contrast, SPLE does not have a defined end point. SPLE lasts as long as products are produced and maintained. One additional challenge is the fact that factors in the project environment (e.g. technologies, competitors) have to be taken into account for much longer lifetimes [9]. Another reason why the term project management is a bit complicated in SPLE is the fact that there are two types of development processes, which run in parallel. In fact, there is one application engineering project for each new customer or product to be developed. However, these projects are not unrelated. The possibilities provided by the domain engineering process are used in application engineering, and feedback from application engineering is used to improve the domain.

In the proposed approach, the domain engineering is based on single system development. There we have a typical project, but once again, it is accompanied by an initialization and domain engineering phase. The separation of concerns is again very difficult.

3.4.2 SPL initialization phase

The SPL initialization phase is a non-recurring process step. In this phase, the early management cycle coordinates the main activities. The initiation for this cycle is a customer idea or request, and the aim is to describe a business case. This

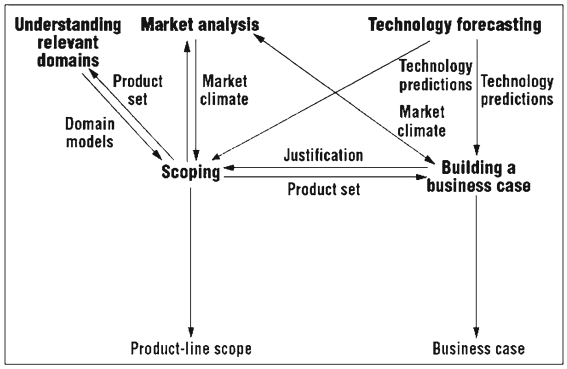


Fig. 7 "What to build" pattern [9]

business case is the base for the go/no-go decision, which is made at the end of the SPL initialization phase.

Several activities provide the basis for this decision. As mentioned before, for a small enterprise it is essential to base decisions on a solid ground. This is of course true for other enterprises as well, but for small enterprises, wrong decisions can lead down the road to ruin. The "What to build" pattern, illustrated in Fig. 7, gives a good overview of the most important upfront decision making activities and their interrelations. The first important prerequisite is a proper understanding of the domain. To be successful, a small enterprise may want to invest in a market niche. One characteristic of a market niche is the fact that there are fewer or no competitors, which makes it easier to gain market share. For this market niche, an appropriate market analysis has to be performed. Customer needs and potential competitors have to be investigated. If there are competitors, reasons why the customer needs are not satisfied have to be analyzed or improvements and additional customer benefits have to be identified. There is a floating transition to the scoping process, where the boundaries of the domain are defined.

3.4.3 Domain scoping

The definition of the domain scope is the first important preparation step for each domain engineering activity. Especially for a small enterprise, the definition of the domain boundaries can be crucial. Too broad of a scope can lead to financial ruin. On the other hand, a scope that is too limited can hinder the development of new projects and thus again lead to economic disaster. As mentioned before, one way to be successful as a small enterprise is to fill a market niche. In such cases, it is important to provide enough variability to support as many projects as possible. This again is a question of good scoping decisions. To make good scoping decisions, the domain has to fulfill the market needs and the needs of the different stakeholders. Therefore, once a market has been found, the different stakeholders have to be identified. Each stakeholder has a different view of the domain and thus needs different kinds of information. Smaller enterprises usually serve smaller domains. For these small domains, the use of questionnaires is an appropriate domain analysis method.

3.4.4 Requirements

An SPL has a long lifecycle and is used to produce many different products. Therefore, it is even more important to conduct a thorough requirements analysis. This is particularly true in the proposed approach, since product requirements are reused as a base for SPL requirements. When performing this analysis, it is important to keep a few recommendations in mind. First, in addition to the functional requirements, technological requirements have to be formulated. This task is covered by the "Technology forecasting" point in Fig. 7.

For the customer, this requires no additional effort because it is also of interest in single system development. The customer has an interest in getting the best state-of-the art technology, which is particularly true if the product has a long life cycle and/or contains special hardware. Especially if hardware is involved, it is important to get spare parts, if necessary. Another factor is the development of a modular, reusable and maintainable design. These factors are mostly covered by quality requirements. Of course, the requirements of the first pilot project do not cover all aspects of the domain. As illustrated in Fig. 6, the project requirements have to be carried over to domain requirements. Domain requirements are a generic description of system behavior. This means that domain requirements include variability.

3.4.5 Domain engineering

The domain engineering process consists of two different cycles. The later management cycle is a support cycle in the domain engineering process. It includes the continuous adaptation of the business case and the monitoring of market dynamics, which is necessary to react to changing market demands. The second cycle is the domain engineering cycle often described in the literature.

3.4.6 Domain modeling phase

Using information gathered in previous process steps, it is now possible to create a domain model. This meta-model or DSL is a framework of reusable components. The aim of domain modeling is to provide a platform allowing economical production of products via generative programming. One important part of this framework is the elaboration of variabilities. The development of a precise and complete domain model is very important for the success of the SPL, as reengineering of the domain model in a later development phase

involves significant additional work. A domain model is a representation of common and variable parts. Commonalities are features that are the same for all products, while variabilities are features which differ between products of the SPL [28].

Variability can be seen as a delayed design decision. It is specified in the form of variation points. Possible variants are defined for each variation point.

One essential task in SPLE is to specify the granularity of the variability. Fine-grained variability allows for the modeling of most projects in a domain. However, the disadvantages are the resulting complexity of the meta-model and the increased difficulty of system modeling [16].

3.4.7 Core asset development

The code generator translates an application model to either another model representation or to the corresponding source code. In SPLE, this generator is often parameterized with templates [44], which define the implementation for the domain objects. Generators also check and optimize input and sometimes even add missing information to the input. From this revised input description, the output code is generated [7].

3.4.8 Application engineering

The AE process handles the actual generation of a product with the help of the developed framework. The products are matched to the specific customer requirements. The system is modeled using the functionalities of the DSL which has been established during the domain engineering process. Source code is generated out of this model. Depending on the SPL and the given domain, the produced product might have to be altered. If too many parts of the code have to be changed, the possibility of changing the domain model must be considered. This can be the case if the domain was not well understood during the engineering phase [29].

3.4.9 Software maintenance

This aspect is not explicitly represented in Fig. 6, since it is not crucial for the SPL initialization. However, it is crucial for the overall success of the project. Even more than in single system development projects, it is important to keep the maintenance costs low. First, the domain has to be maintainable with little effort. Domain maintenance is a recurring task, so there is a lot of potential for improvements. As mentioned in the previous sections, maintainability of automation systems is often a problem. A modular structure of the resulting products can help to improve this aspect.

4 Case studies

4.1 Case study I: a PLC controlled inventory system

4.1.1 Domain description and aims

The domain of the first case study is a logistics system which is built of conveyors, rotary tables, cranes and high bay racking components.

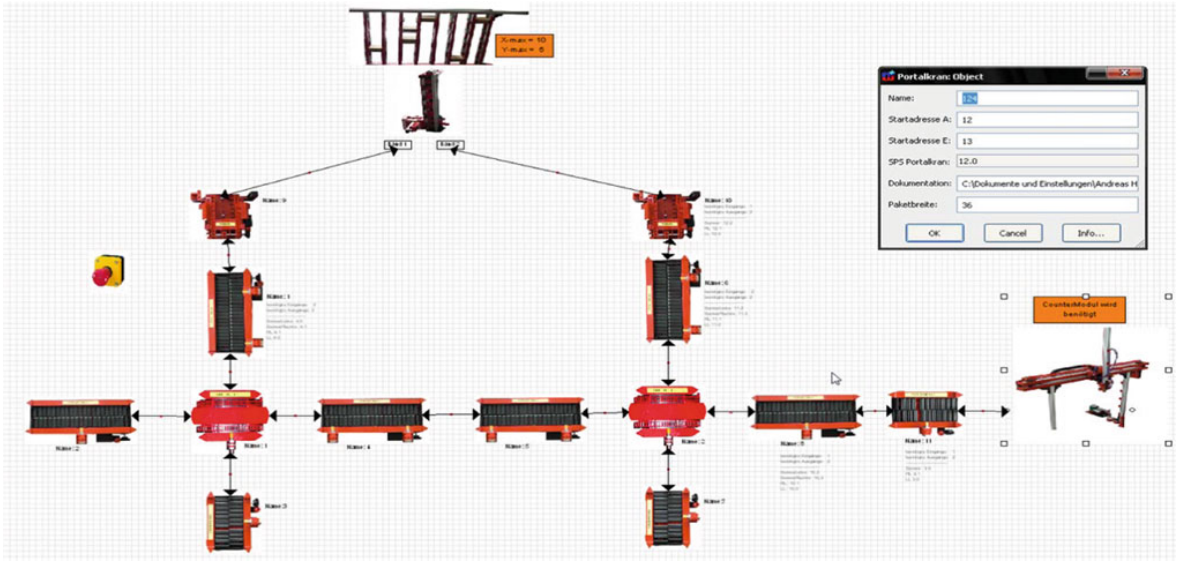


Fig. 8 MetaEdit+ model of an automated logistics system [15]

Fig. 9 Small-scale inventory system lab model [15]

The result is an SPL platform which can be used to create AS software, documentation, etc. for various assemblies of these components. Figure 8 shows a sample graphical model of the domain. An extensible set of language elements is stored in some kind of reusable element pool, where each element should be adaptable to customer needs. A graphically specified logistics model gets transformed to complete, fully functional PLC program source code and a consistent PLC cabling documentation. Afterwards, the program is imported into the SIMATIC® manager. This is a programming environment for the SIMATIC S7® PLC, which was to be supported in this project. Application engineering contains the graphical assembly and configuration of the logistics system. After the logistics system has been defined, the PLC code and a documentation are generated.

4.1.2 Context, stakeholders, and project characterization

This project serves as a pilot project to investigate the suitability of DSM for AS. During the project, we identified several problems, which we sought to overcome in a second case study. The differences and lessons learned are described later on in this work. First, we will describe the setting of this case study.

The study was conducted as an academic project in a hypothetical small enterprise setting for an educational small-scale lab model at our Institute (see Fig. 9). Although no industry partner was directly involved, we applied realistic requirements and used cases from industry. In particular, customer (logistics expert), AS developer, and electric contractor roles were assigned to simulate a real-life setting.

The initial AS functionality, PLC configuration, and cabling were implemented by students as a single system project. Although designing the AS software for reuse was one initial goal, this ultimately had to be abandoned due to time pressure, which is a typical scenario in industry practice as well.

Domain engineering towards SPL development was performed in a separate, subsequent student project. To start, it was necessary to spend some time becoming familiar with the domain. This phase was quite similar to the single system development project described before. As it turned out, it was necessary to rewrite substantial parts to make the AS software components truly reusable. In particular, a domain-wide handover protocol had to be devised that allows for free assembly of any number of components. Some subtle problems had to be solved to make this general enough to work correctly in an IEC-61131 compatible PLC. The challenge is the change in behavior which occurs when the sequence of function blocks is altered. This sequence, however, is determined by the DSM model and cannot be predicted in AS component development. Compared to AS component and generator development, designing a DSL was straightforward. Figure 8 gives an impression of the graphical logistics DSL.

4.1.3 Domain modeling and generator architecture

Because there is no real customer for this project, system requirements were provided as input for the student projects. The domain scope was restricted by the functionality of the lab model. There were a number of devices, such as a high bay racking, a gantry crane, conveyors, a rotary table and rack servicing units. All of them were modeled as language elements of the domain. They can have several properties and constraints, and there are certain rules about how the different elements can be combined. In particular, we developed a general synchronization protocol for all element interfaces that deal with pallet handover and queuing. This is the primary interface type and enables the free combination of logistics components handling pallets. Other rules check the plausibility of the topology and the property values modeled.

Figure 10 provides an overview of the implemented modeling and generation architecture. The upper layer shows

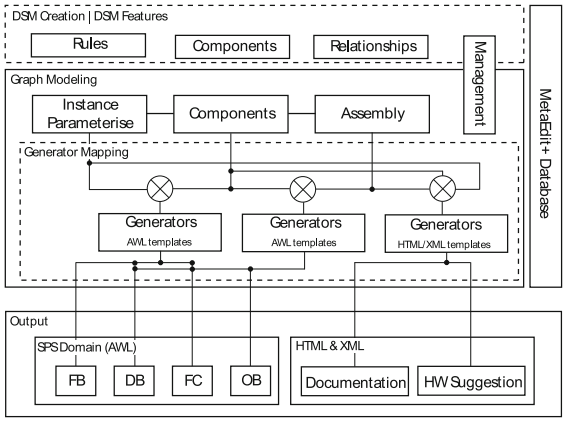


Fig. 10 Modeling and generator architecture of the implemented inventory system [15]

the domain modeling process with all the components, rules and relationships. Dashed frames identify the domain, while black frames are application specific. On the second layer, the application is assembled. The main parts are components, which have to be instanced, parametrized and arranged as needed. The management supervises both the domain modeling and the graph modeling (application engineering).

Generators are based on the graph modeling layer. They extract information required to generate the output. As indicated, each generator requires different information to prepare the output files. All generators are highly domain specific. The database (i.e. the repository of MetaEdit+) contains the models, rules and generators. The generators are implemented using MERL, the MetaEdit+ scripting language. A large amount of domain-specific knowledge is captured within the generators to produce the required output artifacts. The PLC code generator is the most complex one and creates all necessary program files for the PLC using the IEC 61131-3 languages FBD (function block diagram) and IL (instruction list). The program files are imported into the PLC development system, translated, and deployed on the PLC itself. On the generator level, error checks validate different values (e.g. the limited size of the high bay racking and the connections of the rack servicing unit, which must fit within the given dimensions). In addition, the existence of all important values is tested. Theoretically, this could also be done while modeling using constraints that are checked immediately. However, this would not be practical, as there are often inconsistencies during the modeling process that can only be resolved when the generator runs. Generator warnings are produced for elements lacking neighbors. Also, an error is flagged if there is no kill switch included in a concrete project, since this is mandatory in the logistics domain.

The documentation is dynamically created and organized as a website. The documentation MERL script generates *.HTML files with embedded Javascript functions and a corresponding style sheet. The advantage of a website is the interactive behavior. Unlike in a text-based documentation, it is possible to link corresponding elements. MetaEdit+ enables the export of an application model image using Javascript commands. This image is shown on the main page of the documentation. Clicking on an element shows detailed information. There are different parts included in the generated documentation:

- *Main document* Gives an overview of the system.
- *Pin binding* Shows a wiring diagram.
- *Generated files* Lists all the generated files.
- *Recommended tests* Dynamically generates standard test instructions.
- *Available instructions* Each new system has new commands which are calculated dynamically. These commands are presented on this page.
- *Hardware suggestion* XML-based hardware suggestions for the generated system.

For a discussion of this project and a comparison to the following "PISCAS" case study, see Sect. 5.

4.2 Case study II: PISCAS: a pisciculture automation system

The second case study was performed as an industry project. Since this project has more significance, we will take a closer look at it. It involves the development of a fish farm AS.

This section describes the main project setting. Detailed information can be found in [33]. Section 5 presents and discusses the results of this case study.

4.2.1 Domain description and aims

The main functional requirements for a fish farm AS are feeding and oxygen monitoring, including an alarm system. In addition, water level monitoring, pH value measurement, and standard functions like switching and light controls have to be developed.

One main aim of the AS is to make the work of the fish farm owner easier and to save resources. Therefore, we introduced a fish growth model to keep track of the current fish weight in a pond and to adjust the amount of food according to that weight. Oxygen monitoring enables energy savings by switching off the oxygen supply if the oxygen level of a pond is high enough. The pH value and water level monitors provide quick identification of critical pond states, which in turn helps to take timely countermeasures.

The goal of the AS developing company (HOFERNET) is to keep system development and maintenance efforts low and thereby potentially open up a new market.

Fig. 11 PISCAS stakeholders [33]

4.2.2 Context, stakeholders, and project history

The three main stakeholders are the companies Elektro Tisch, HOFERNET, and Kärnten Fisch. HOFERNET is a very small company that developed the SPL for the AS. Elektro Tisch was responsible for on-site electrical and AS hardware installation. The fish farm company Kärnten Fisch was the customer ordering the AS from HOFERNET. Figure 11 provides an overview of this stakeholder scenario.

4.2.2.1 Evolution history So far, HOFERNET has automated two fish farms (Radenthein and Feld /Austria). Both consist of several ponds. All ponds are equipped with a feeding automat, and some of them have oxygen level monitoring. Hardware components (sensors and actors) already existed before the installation of the PISCAS system. Feeding had been triggered by a time switch, and the aerator supplying oxygen had been operating continuously.

Initialization phase: Development started as a single system project for one site. Getting the fish farm AS up and running was clearly the primary goal. At HOFERNET, only two persons were involved in this project: the CEO and one of the authors.

However, as contracts for additional sites became a possibility, a vital interest arose in designing the AS (including software, hardware, cabling, sensors, and actors) for reusability. For the same reason, the potential of the fish farm AS market was analyzed, and consideration of a business case began.

Overall, the project was considered a success. Two observations, both of which concern external stakeholder communication, became the motivation for SPL transition:

- The most underestimated aspect was understanding the fish farm domain, including its principles, rules, procedures, and language (i.e. the communication with the fish farmer).
- The communication with the electrical contractor turned out to be an issue as well. Neither the abstraction level nor the formal kind of specification was clearly defined. When this was combined with a hands-on mentality, misunderstandings ensued, which even made it necessary to rework the cabling (including liability discussions).

Software product line transition phase: When starting with a second project, we decided to organize future PISCAS development as an SPL. Starting from the initial project's experience and substance, the same two persons carried out the SPL transition. Two parallel threads of activity were kicked off:

- First, we developed a "standard" PISCAS cabling concept and standardized terminal blocks. We complemented this environment with an also new catalog of PISCAS supported sensors and actors. In this way, electrical contractors should get enough guidance to deliver a perfect installation without substantial rework.
  The PLC software used in this activity was based on the initial project and modified according to the new fish farm.
- We started the PISCAS domain engineering activities based on work done in the initial project. This was done by the software developer of the initial project. In this way, he was able to fully reuse his knowledge of the domain, its requirements, and the AS software. The main tasks were designing a DSL, writing AS code generators, and creating generators for consistent documentation and cabling plans.

4.2.3 PISCAS development

The approach proposed in Sect. 3 was used to develop an AS SPL. We identified requirements for the specific domain during meetings with the customer and with the help of VOLERE checklists [34]. These requirements were then used to construct use cases to enable checking of the functional project results.

Meetings with the customer brought insight into the variability of the domain. In addition, scenarios for other known fish farms were constructed.

These variants were used to construct a DSL with the tool MetaEdit+ [41]. This tool enables the creation of graphical editors for modeling domain-specific systems and provides the ability to generate code from these models. The MetaEdit+ model elements for the fish farm DSL can be seen in Fig. 12 and are further described in [32].

As a base for the code generator, we developed a general PLC software prototype for fish farm AS. Due to the modular design, the functionality of the PLC software can easily be

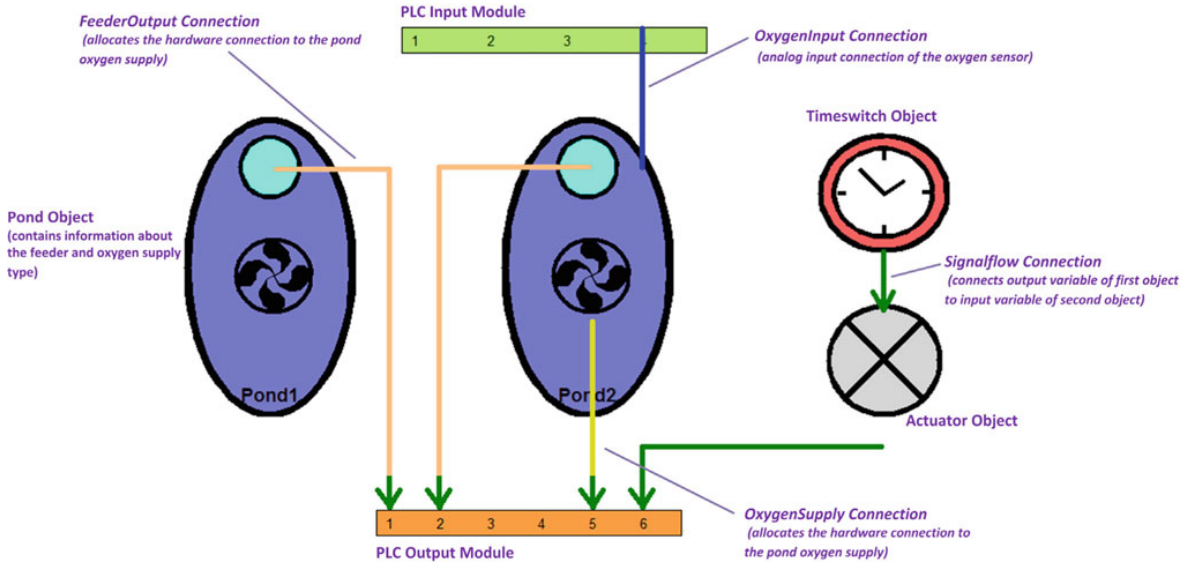


Fig. 12 Model elements for PISCAS fish farm software product line (MetaEdit+) [33]

enabled or disabled by setting parameters. In addition, this PLC software is equipped with a test mode, which enables easy testing of the installed hardware components. For example, it includes an automated test of the emergency power supply unit. With the help of this test mode, the electrician or the AS developer can verify the correct function of hardware components during on-site installation. These test functions should also decrease maintenance effort, as they make it easier to locate hardware faults. The PLC software template is used to generate the PISCAS software using the DSL application model for the concrete system which is described in more detail in [31].

In addition, to the PLC software, project documentation for the fish farm system is automatically generated. Besides a user manual for the AS, the documentation includes an electrical installation plan, which serves as a basis for inter-organizational communication between HOFERNET and Elektro Tisch. Table 3 provides an overview of the generated artifacts and a side-by-side comparison with Case Study I.

4.2.4 PISCAS implementation

Due to time pressure, we had to be put the AS into operation before the software generator was completed. At this point, we had already finished the template and just had to configure it to work with the specific fish farm in Radenthein (Austria). This method is known as clone&own. During the configuration of the software, many errors occurred, which led to significant extra work. Nevertheless, this initial operation provided important data for the cost evaluation of different development approaches. The same system in Radenthein now runs with a software generated from a model.

The development of the second system in Feld (Austria) started when modeling and code generation tools were fully functional, and it was therefore possible to carry out this project in an SPL manner using DSM.

Both PISCAS AS have been in operation since April 2010, and no major software errors have occurred. Only minor errors have needed to be corrected. Several updates have been necessary, due to functional extensions. These maintenance efforts are reflected in the cost analysis.

5 Discussion of results

This section shows real-life data based on the development of the SPL and the installation of the fish farm systems in Radenthein and Feld (Austria). Since the automated systems are about the same size, the results gathered during the installations of these two systems are comparable. Where available, data from Case Study I is given as well.

The analysis is conducted for three different types of development approaches.

- *Single system development* In this scenario, an individual project is implemented for each AS. It is assumed that there is no prior knowledge about the domain, and no part of the software is reused for the development of a new system. This is not a realistic scenario, since there is always some kind of knowledge and code reuse, but it serves as a good comparison base for the other methods.

- *clone&own* This is the most common approach. It is often used for the development of AS in practice. In this scenario, existing software is copied and altered according to the requirements of the new project.
- *SPL* This is the approach suggested in this paper. After the domain analysis step, a DSL for the specific domain is designed, and code generators are implemented. Specific systems are modeled using the DSL. The code generators are used to generate code from the individual system model.

5.1 Cost model

The cost model used here is based on the model proposed in [10], which has been adapted to the needs of AS. Three scenarios are important for our estimation:

1. Building n products with single system development

$$\sum_{i=1}^{n} C_{unique}(p_i) + C_{prod}(p_i) + C_{install}(p_i) + C_{maint}(p_i)$$

2. Building n products with clone&own

$$C_{cab2}() + \sum_{i=1}^{n} C_{unique}(p_i) + C_{prod}(p_i) + C_{install}(p_i) + C_{maint}(p_i)$$

Remark: $C_{cab2}()$ is used as a base for clone&own development since both single system development and template development have almost equal efforts.

3. Building an SPL with n products

$$Cost_{DE} = C_{org}() + C_{cab1}() + C_{cab2}() + \sum_{i=1}^{n} C_{maint}(i)$$

$$Cost_{AE}(p_i) = C_{unique}(p_i) + C_{reuse}(p_i) + C_{install}(p_i)$$

$$CostsSPL = Cost_{DE} + \sum_{i=1}^{n} Cost_{AE}(i)$$

For our project we can assume the following:

- C_{org} Code generator development
- C_{cab1} DSL development
- C_{cab2} PLC template development
- C_{unique} Requirements analysis
- C_{reuse} Creation of a product specific model
- C_{prod} Manual development (without SPL)

This includes PLC Project SW development, SW changes and electrical installation changes.

In addition, we introduce two more types of costs, which are relevant for AS development:

- $C_{install}$ Electrical and on-site installation
- C_{maint} Maintenance (platform)

5.2 Cost analysis

We gathered data for the three different development methods during the development of the two case studies. Figure 4 illustrates the division of costs into domain engineering and application engineering costs. Domain engineering costs mainly arise from the first system developed in the domain and also include domain analysis, DSL development, and generator development for all kinds of artifacts needed. Subsequent systems are instances of this platform and lead to application engineering costs.

Table 1 shows the costs for all three scenarios: single system development, clone&own and DSL-based SPLE. We included efforts from Case Study I in Table 1 where applicable and extrapolated efforts using the cost model from Sect. 5.1 for several development scenarios. Figure 13 summarizes the results.

5.3 Break-even point estimation

By comparing the extrapolated efforts from Fig. 13 for each case study, we can identify break-even points for the different scenarios. The break-even point indicates the number of products for which the efforts for SPL-based and clone&own development are equal. clone&own and SPL costs for a number of n products were extrapolated from the data we collected during the case studies.

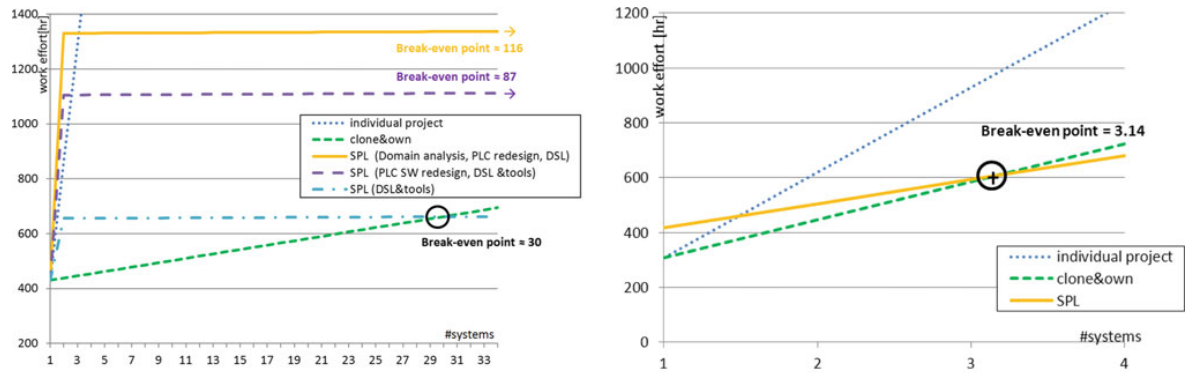
Table 2 shows the break-even estimation for several scenarios in both case studies side-by-side. Single system versus SPL development are compared, as well as SPL- versus clone&own-based development. For Case Study I, we tried to isolate several effort-generating effects (see Fig. 13a) in the case of SPL to gain more insight into the process of initiating an SPL.

A comparison of single system and SPL development shows that an SPL pays off after very few derived applications. This observation confirms reports from the relevant literature [30]. However, here we are mainly interested in the break-even points of the SPL and the clone&own approaches, as this is the most common practice.

Here, the results for Case Study II are better than expected. Significant costs during on-site and electrical installation can be saved with the SPL approach due to the much less error-prone generation of PLC code and consistent documentation for electrical installation, test, and system use. It also turned out that the design of a DSL for the automation domain was

Table 1 Summary of work efforts for alternative development approaches in both case studies described

Tasks		Work efforts (hours)									
		Case Study I: Inventory					Case Study II: PISCAS				
		Single	Clone&own		SPL		Single	Clone&own		SPL	
		Var.	Base	Var.	DE	AE	Var.	Base	Var.	DE	AE
DE (DSL dev.)	C_{cab1}				225					29	
PLC Project SW Dev	C_{prod}	430	430		430		187	171			
PLC Template Dev	C_{cab2}				450					187	
Code Generator Dev	C_{org}				225					115	
Requirements Analysis	C_{unique}						20		20		20
Product specific model	C_{reuse}					0.21					2
Software Adaptations	C_{prod}			8 est.					16		
Electrical Installation	$C_{install}$						40		40		40
Electr. Inst. Changes	C_{prod}						5		5		
On-site Installation	$C_{install}$						40		40		20
Maintenance	C_{maint}						17		17	5	
Base development effort			430		1350			171		331	
Effort per application		430		8		0.21	309		138	5	82



(a) Case Study I: inventory AS **(b)** Case Study II: fish farm AS

Fig. 13 Work efforts for different development approaches: single system development, clone&own, and SPL

not as difficult as expected. This may be due to the favorable characteristics of the AS domain, as described in Sect. 2.1.

Case Study I looks much worse in this respect. When compared to Case Study II, several factors apparently led to that inferior outcome:

- We considered only PLC software, without additional efforts (e.g. installation). Thus, the savings resulting from systematic reuse are not that high.
- The logistics elements basically had only one complex type of interface, with protocols facilitating the handover of pallets and queuing. The development of a generic solution for these functions required higher upfront efforts than in Case Study II.
- There was no efficient domain knowledge transfer (documents only) between the initial implementation and the SPL project. This resulted in an additional extended domain analysis phase.
- Although developed for reuse, the original PLC software framework turned out to be not very reusable in the context of a DSL-controlled generator. Not surprisingly, the main focus of this issue was on the interface protocol mentioned above. Apparently, the initial reuse scenarios were tacitly chosen more in a technical context than in a domain context. A new PLC software framework had to be devised for use in an SPL setting, which required substantial extra effort. This is discussed in more detail in Sect. 5.5.

Table 2 Extrapolated break-even numbers from Case Studies I and II compared

	Case Study I: Inventory AS	Case Study II: Fish farm AS
SPL versus single sys. projects	≥3	≥2
SPL versus clone&own		
Overall		≥4
PLC SW only	≥116	
PLC SW only (w/o domain knowledge transfer)	≥87	
PLC SW only (w/o redesign and knowledge transfer)	≥30	≥8

Table 3 Artifacts generated in the presented AS software product lines

	Case Study I	Case Study II
Generated software		
PLC code (IEC 61131)	Full (FBD, IL)	Full (ST)
User interface	No, manual	Full
Test mode	Rudimentary	Full
PLC		
PLC brand	SIMATIC S7	B&R
PLC config	–	Complete
Documentation		
User doc	Site overview, components	Site overview, PISCAS docs, hardware bill of materials, variable reference
PLC cabling	PLC I/O list	Graphical PLC I/O & cabling
Design support		
DSL	Yes	Yes
PLC hardware selection assistant	Yes	Modeled

To better understand and compare the consequences of these effects, Fig. 13a depicts three sub-scenarios within the SPL approach, which are then compared in Table 2:

- The measured Case Study I SPL introduction with extended domain knowledge acquisition phase due to a developer new to the domain and the re-development of the PLC software framework that became necessary.
- A "what-if" scenario without additional domain knowledge acquisition effort. This would be expected if the same developer (team) did both the initial project and SPL domain engineering.
- Another "what-if" scenario in which both the domain knowledge acquisition effort described above and the PLC software redesign are removed. This scenario is then very similar to the concrete and more successful evolution in Case Study II (see Table 2).
  Considering only software effects for Case Study II, taking into account the lower upfront costs and thus higher relative savings per SPL-derived application, both cases end up in a comparable range. Nevertheless, while it was achieved in Case Study II, this shows the potential that unfortunately was not realized in Case Study I.

5.4 Lessons learned

5.4.1 Similarities identified in the case studies

Even though the presented SPLs (Sects. 4.1, 4.2) have different domains, both share characteristics typical for AS. They also share the ramp-up scenario in Fig. 6. In addition, the list of artifacts generated from an application model in the respective DSL looks quite similar (see Table 3).

Applications are specified using the DSL. All PLC-relevant artifacts are generated from this application model. By separating the domain knowledge from the technical realization, we become independent from specific hardware vendors. In this way, switching the hardware vendor seems within reach by simply exchanging the code generator(s). This is especially interesting for small companies, as they are most susceptible to vendor lock in.

5.4.2 Start SPLE lightweight and early

As can be seen in Fig. 13, starting a domain-specific AS SPL can pay off after 3–4 application projects. This figure is

reported in the literature as well. However, the upfront cost can pose a problem, especially for small companies. We were able to demonstrate that the use of the presented approach makes this feasible for small AS companies as well. This is particularly true when early revenue is gained from the initial project.

5.4.3 *Efficient knowledge transfer from single system development to domain engineering*

In Fig. 6, many arrows indicate knowledge flows from initial project engineering to domain engineering. Domain understanding, requirements, solution architecture, and implementation artifacts all must be generalized to be usable in domain engineering. Based on the two case studies, the best approach seems to be that the developer of the initial project does this her/himself. Especially in small enterprise contexts, this is not only feasible, but seems to be necessary to achieve a reasonably efficient domain development. In turn, this would yield an attractive break-even point.

5.5 Knowledge transfer in the domain engineering process of a small company

Knowledge transfer means the process of making implicit domain knowledge explicit and thereby transforming the existing documentation of a legacy system into a reusable domain model. All interfaces described in Fig. 6 represent knowledge transfer. Any time, when knowledge is transferred from a single project setting to the domain, problems may occur. There seem to be some common factors which enable or hinder the knowledge extraction process.

One obstacle to knowledge extraction in a large domain was identified in our previous work [24]. In this paper, we identified the existence of a domain expert team: a management expert responsible for scoping, and experts responsible for providing methodical background, variant expertise, and target platform expertise. These domain expert team roles can certainly be taken over by one and the same person in a "small" domain, given that this person has enough expertise in all of the necessary fields. In a complex domain, there will almost certainly be a team of several experts.

Another overview of roles in an SPL project presents a more detailed listing of roles related to the three main SPL activities (management, domain engineering, and application engineering) [43]. Again, the number of project members taking these roles depends on the complexity of the domain.

In a small AS developing enterprise, the domain will probably be relatively small, so the aforementioned domain expert team could consist of one or maybe two individuals. For a large domain, the problem is the coordination and cooperation of this domain expert team. If this is lacking, it can hinder the domain modeling process. Since this cooperation problem seems to be insignificant for a small company, there have to be other problems, which are not obvious at first sight.

We investigated two different problems. Both case studies are from the AS domain, comparable in size and artifacts needed to the system context. In each project, a single person worked on the introduction of the SPL approach, thereby becoming an expert in the domain, technology, and DSL methods and tools. In one project [33], the domain modeling process worked without problems. In the other project [15], the domain modeling was a slow-moving process at the beginning. Only after some time did it function without major problems. The question is: what are the differences between the two projects that can explain the relative levels of success or failure?

Here are some observations which we believe make the difference:

- *Design for reusability* In Case Study I (Sect. 4.1), the initial project was originally designed by different developers who had reusability in mind, but without a concrete idea about the reuse context. In Case Study II (Sect. 4.2), the initial system was developed with the SPL approach in mind.
  SPLs use a predictive software reuse strategy. Opportunistic reuse, on the other hand, would involve building a library of components that can be reused [21]. For us, this is the main difference between the two approaches. In the first approach, we developed the components with reuse in mind, but it was not clear when and how the components would be reused. Therefore, the reuse capability was based on the capabilities of IEC 61131 languages, and not on the explicit variability requirements from the domain context. In the second approach, we defined variation points, perhaps not in the software itself, but in the design of the software. This made the transition to the SPL approach significantly easier.
- *Implicit knowledge versus documentation* In the more successful fish farm case, domain knowledge was implicitly available from a domain expert. In the other project, the knowledge had to be gained from documentation and legacy software. This method of domain knowledge acquisition required significantly more effort. Furthermore, documentation is often written from one specific viewpoint. To get a proper understanding of the domain, it is necessary to have different viewpoints.
  Hidden design decisions were another issue. In documentation, the actual design of software is described, but many design decisions are not explained in detail. However, especially for reuse, knowledge about design decisions is of great value.
- *Coupled development projects* For Case Study I (Sect. 4.1), the single system development project was more or less finished before the start of the domain engineering

project. The domain engineering process was executed completely independently from the individual project. This also means that the developers were not available for consultation. Thus, two parties had to familiarize themselves with the domain: the project developers and the person who developed the domain. This involved duplicate work, which is not really necessary.

In Case Study II, the domain engineering developer was also heavily involved in the single system project. In this way, domain understanding was already present to a large extent.

We suggest two different scenarios to avoid this problem. Either the two projects have to be really parallel, or one person has to be involved in both projects.

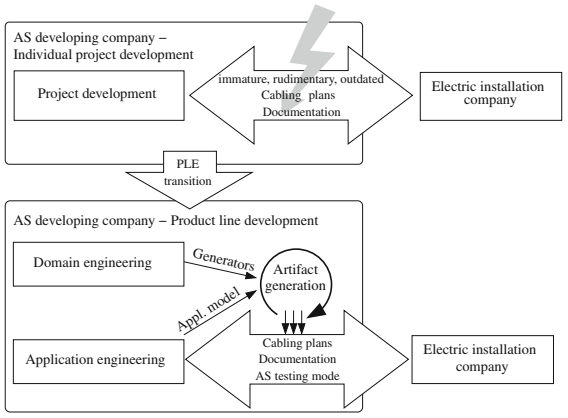


Fig. 14 Improving and structuring AS developer to electrical contractor communication utilizing generated, consistent, up-to-date documentation artifacts and AS test mode

5.6 Communication problems

An issue mentioned in Sect. 1 is the far-from-perfect communication between the AS developers and the electrical contractor responsible for electrical installation. Experience from Case Study II (Sect. 4.2) showed that insufficient communication leads to significant extra effort. Figure 14 illustrates the basic communication problem. In a single system project, there is usually no defined communication between the two parties. The introduction of an SPL approach provides the possibility to generate basic artifacts that support stakeholder communication. This means that it is possible to easily introduce a defined communication. Due to generation of documentation, cabling plans, and even test mode PLC software, these artifacts are up-to-date and consistent with the application model. Furthermore, they are available at almost zero cost. This is an important factor, particularly for small enterprises with limited engineering resources. The case studies described here show that this is one way to reduce errors and installation efforts.

6 Related work

SPLs have been suggested as a systematic reuse approach for software-intensive systems [11]. The approach has been proven a number of times (e.g. [12,36,39]. The SPL approach involves two fundamental development cycles: domain engineering (core asset development) and application engineering (product development) [11,30].

One of the most important issues is the SPL adoption strategy. The organization, the processes and the methods have to be adapted to the SPL concept [5]. The major question is how an organization can make the move from developing one-off products to SPLE without major interruptions in the day-to-day-work. The different needs of organizations prevent the description of a generic transition strategy, and each organization must define its own action plan [26] because they will vary in size, structure, product type, culture, reuse potential, etc.

One agile systems' engineering industry survey identified the use of SPLs as one success factor for multi-discipline domains (as AS) as well [37]. This approach helps to facilitate quick, market-driven product variants. The same study also identified efficient communication paths with customers and partners in which their own language is used as another success factor. The approach presented here incorporates both aspects.

Gonzales and Benavides[14] propose an iterative process built on incremental domain engineering based on iterative application engineering. In this process, the domain is built from a basic product and then iteratively refined through feedback from application engineering.

Other reference processes assume the determination of stakeholders and the definition of business goals to be the most important preparation step [5,35]. As mentioned before, business goals are the major drivers of an SPL. The goals are highly dependent on the different stakeholders. Therefore, product-line goals have to be derived from different points of view (e.g. financial, customer, people, process, infrastructure and innovation) [26]. According to [5], the realization of the business case is defined in an adoption plan, which defines the current state, the desired state and the strategies for getting there. Many transition strategies are available. The adoption plan must stipulate the most suitable strategy for the current situation. Once an adoption plan has been chosen, the SPL is launched. However, this alone is not enough for a successful SPL. The SPL has to be institutionalized. This means that the processes have to be perceived as stable [5]. One important aspect is supporting the process with defined roles and responsibilities. Without assigned roles, processes can

become chaotic and useless [14,35]. Much more research has been done on the issue of legacy code analysis and integration of existing code into a core asset base than on organizational transition [18].

Bekkers et al. [6] propose a software product management competence model. It organizes focus areas within four business functions and sets this model in context to external and internal stakeholders. The business functions and their product management focus areas are: *Requirements management (Requirements gathering, Requirements identification, Requirements organizing),* Release planning (Requirements prioritization, Release definition, Release definition validation, Scope change management, Build validation, Launch preparation), *Product planning (Roadmap intelligence, Product roadmapping, Core asset roadmapping), and* Portfolio management (Market analysis, Product lifecycle management, Partnering and contracting). External stakeholders are the market, customers, and partners. While devised for larger organizations, this model also maps well to activities, responsibilities, and stakeholders found in our approach (cf. Fig. 6) for small enterprises.

Moser et al. [27] propose an ontology-based approach called *Engineering Knowledge Base* to capture knowledge in a multi-discipline engineering process. This is done to translate and ensure correct communication and cross-discipline joints, also for tools. This problem pattern shows up in an AS setting both at the interface to the customer (domain expert) and the electrical contractor.

7 Conclusion and future work

This paper presents a DSM-based transition process approach to SPL for small AS developing companies. We applied this approach for two AS development settings in the logistics and fish farm domains. Qualitative and quantitative results were given and discussed.

Compared to the traditional approach, some of the main advantages of our approach are:

- Domain knowledge is made explicit in the meta-model (DSL) and can therefore be reused by other developers. This makes the enterprise more independent from specific developers.
- High-quality software is produced, since automatic code generation reduces the number of errors.
- Lower setup time for the on-site installation is expected due to the generated installation plan and the test mode of the PLC software.
- Efficient development of applications from a product line can be expected, since the systems are modeled on a higher level of abstraction, and generators produce all artifacts in consistent variants.

The lessons learned from our case studies show that some common disadvantages reported in the literature can be overcome by the present approach, especially in the context of small enterprises:

- When starting with an initial customer project, this early revenue can be used to fund domain engineering tasks.
- For AS, using a DSM workbench can make domain engineering very efficient. This is one factor that makes the approach profitable after only a few application projects (see also Table 2).
- Knowledge transfer from the initial project to domain engineering has to be very broadband and efficient. Ideally, the same person performs both tasks, which again makes domain engineering efficient.

7.1 Future work

Further investigation is needed to generalize the presented approach to other AS domains. The observations stated in this paper are based on interviews with companies and two SPL introduction case studies. More interviews and SPL introduction projects must be analyzed to gain better understanding of a generalized AS context.

Reference data for equal systems developed with and without the proposed system architecture would be of great interest. More systems developed with different approaches or more applications derived from the same SPLs would enable further validation of the cost model and break-even estimates.

It would also be of interest to adapt an AS SPL for a different domain. This is of special interest for small enterprises who want to expand into another domain. Information about the similarities and variabilities of different AS DSLs would be of interest. Research into this domain change would yield desirable information about the efforts required to successfully make the transition. Another area of interest would be to gather information about the efforts required to change the PLC vendor. Variability differences for the software of different PLC vendors would be of interest. It would also be desirable to acquire data about product lines developed with different PLC vendor hardware. This could lead to a ranking of the vendors regarding their suitability for SPLE in the automation sector.

Further study of inter-organizational processes in the automation sector could yield interesting results. Currently, the lack of efficient communication between stakeholders leads to extra costs, mostly during AS installation. We suspect that modeling these effects to enable quantitative analysis could eventually lead to improved process quality and repeatability.

Acknowledgments This work was funded by an FFG Innovationsscheck.

References

1. Abdellaoui, M., Gonzales, C.: Multiattribute Utility Theory, pp. 579–616. ISTE (2010)
2. Boehm, B., Brown, W.A., Madachy, R., Yang, Y.: A software product line life cycle cost estimation model. In: International Symposium on Empirical Software Engineering (2004)
3. Brito e Abreu, F., Melo, W.: Evaluating the impact of object-oriented design on software quality. In: Proceedings of the 3rd International Software Metrics, Symposium, Mar 1996, pp. 90–99 (1996)
4. Breivold, H.P., Larsson, S., Land, R.: Migrating industrial systems towards software product lines: experiences and observations through case studies. In: SEAA '08: Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications, pp. 232–239. IEEE Computer Society, Washington, DC, USA (2008)
5. Böckle, G., Munoz, J.B., Knauber, P., Krueger, C.W., do Prado Leite, J.C., van der Linden, F., Northrop, L.M., Stark, M., Weiss, D.M.: Adopting and institutionalizing a product line culture. In SPLC 2: Proceedings of the Second International Conference on Software Product Lines, pp. 49–59, London, UK. Springer, Berlin (2002)
6. Bekkers, W., van de Weerd, I., Spruit, M., Brinkkemper, S.: A framework for process improvement in software product management. In: EuroSPI 2010 Proceedings, pp. 1–12 (2010)
7. Czarnecki, K., Eisenecker, U.W.: Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York (2000)
8. Constantinescu, R., Iacob, I.M.: Testing: first steps towards software quality. J. Appl. Quant. Methods. **3**(3), 241–253 (2008)
9. Clements, P.C., Jones, L.G., Northrop, L.M., McGregor, J.D.: Project management in a software product line organization. IEEE Softw. **22**(5), 54–62 (2005)
10. Clements, P.C., McGregor, J.D., Cohen, S.G.: The Structured Intuitive Model for Product Line Economics (SIMPLE). Technical report, Software Engineering Institute at Carnegie Mellon University (2005)
11. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
12. Dager, J.C.: Cummins's experience in developing a software product line architecture for real-time embedded diesel engine controls. In: First Conference on Software Product Lines: Experience and Research Directions, pp. 23–45. Kluwer Academic Publishers, Dordrecht (2000)
13. de Almeida Falbo, R., Guizzardi, G., Duarte, K.C.: An ontological approach to domain engineering. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE '02, pp. 351–358, New York, NY, USA. ACM, New York (2002)
14. Gonzalez, J., Benavides, D.: Transition from Products to Products Lines (Case Study). http://www.esi.es/Cafe/pdf/Product_line_transition_and_adoption.zip (2010). Accessed 15 Oct 2010
15. Haselsberger, A.: Design and Implementation of a Domain Specifc Architecture for PLC. Master's thesis, TU Graz, Graz (2009)
16. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in software product lines. In: Proceedings of the 30th International Conference on Software Engineering, pp. 311–320, New York, NY, USA. ACM, New York (2008)
17. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute (1990)
18. Kim, K., Kim, H., Kim, W.: Building software product line from the legacy systems "Experience in the Digital Audio and Video Domain". In: SPLC '07: Proceedings of the 11th International Software Product Line Conference, pp. 171–180. IEEE Computer Society, Washington, DC, USA (2007)
19. Kang, K.C., Kim, S., Lee, J., Kim, K., Kim, G.J., Shin, E.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. **5**, 143–168 (1998)
20. Kruchten, P.: The 4+1 view model of architecture. IEEE Softw. **12**, 42–50 (1995)
21. Krueger, C.W.: Introduction to software product lines. http://www.softwareproductlines.com/introduction/introduction.html (2012). Accessed 9 Oct 2012
22. Kelly, S., Tolvanen, J.-P.: Domain-specific modeling: enabling full code generation. Wiley, New York (2008)
23. Leitner, A.: A software product line for a business process oriented IT landscape. Master's thesis, TU Graz, Graz (2009)
24. Leitner, A., Christian, K.: Managing ERP configuration variants: an experience report. In: KOPLE 2010: Knowledge-Oriented Product Line Engineering, pp. 1–6. ACM Digital Library (2010)
25. Leitner, A., Weiß, R., Kreiner, C.: Analyzing the complexity of domain models. In: ECBS12 Proceedings (2012)
26. Mansell, J.X.: Product-line action plan specification. http://www.esi.es/Cafe/pdf/Product_line_transition_and_adoption.zip (2010). Accessed 15 Oct 2010
27. Moser, T., Mordinyi, R., Biffl, S.: An ontology-based methodology for supporting knowledge-intensive multi-discipline engineering processes. In: ODiSE 2010: 2nd International Workshop on Ontology-Driven Software Engineering, pp. 2:1–2:6. ACM Digital Library (2010)
28. Metzger, A., Pohl, K.: Variability management in software product line engineering. In: ICSE Companion, pp. 186–187 (2007)
29. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques, Secaucus, NJ. USA. Springer, New York (2005)
30. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering. Springer, Berlin (2005)
31. Preschern, C., Kajtazovic, N., Kreiner, C.: Applying patterns to model-driven development of automation systems: an industrial case study. In: EuroPLoP (2012)
32. Preschern, C., Leitner, A., Kreiner, C.: Domain specific language architecture for automation systems: an industrial case study. In: ECMFA12 Proceedings (2012)
33. Preschern, C.: PISCAS-Pisciculture automation system product line. Master's thesis, TU Graz, Graz (2011)
34. Robertson, J., Robertson, S.: Volere requirements specification template. http://www.volere.co.uk/ (2012). Accessed 9 Oct 2012
35. Schreiber, A.: Transition process for switching to product-family engineering. http://www.esi.es/Cafe/pdf/Product_line_transition_and_adoption.zip (2010). Accessed 15 Oct 2010
36. SEI Carnegie Mellon. Product Line Hall of Fame. http://splc.net/fame.html (2012). Accessed 9 Oct 2012
37. Stelzmann, E.S., Kreiner, C.J., Spork, G., Messnarz, R., König, F.: Agility meets systems engineering: a catalogue of success factors from industry practice. In: EuroSPI 2010 Proceedings, pp. 245–256 (2010)
38. Smolander, K., Lyytinen, K., Tahvanainen, V.-P., Marttiin, P.: MetaEdit: a flexible graphical environment for methodology modelling. In: Advanced Information, Systems Engineering, pp. 168–193 (1991)
39. Steger, M., Tischer, C., Boss, B., Müller, A., Pertler, O., Stolz, W., Ferber, S.: Introducing PLA at Bosch gasoline systems: experiences

and practices. In: SPLC '04: Proceedings of the 3rd International Software Product Line Conference, pp. 34–50 (2004)
40. Schmid, K., Verlage, M.: The economic impact of product line adoption and evolution. IEEE Softw. **19**, 50–57 (2002)
41. Tolvanen, J.-P., Kelly, S.: MetaEdit+: defining and using integrated domain-specific modeling languages. In: OOPSLA '09: Proceeding of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, pp. 819–820, New York, NY, USA. ACM, New York (2009)
42. Tolvanen, J.-P., Pohjonen, R., Kelly, S.: Advanced tooling for domain-specific modeling: MetaEdit+. http://www.dsmforum.org/events/DSM07/papers/tolvanen.pdf (2012). Accessed 9 Oct 2012
43. van der Linden, F.J., Klaus, S., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer, Berlin (2007)
44. Völter, M.: A catalog of patterns for program generation. In: Proceedings of the 8th European Conference on Pattern Languages of Programms (EuroPLoP '2003) (2003)

Author Biographies

Andrea Leitner received her Dipl.-Ing. (MSc) from Graz University of Technology in 2009. Since 2010, she is working on her PhD at Graz University of Technology. Her main research fields are Software Product Lines, variant management and knowledge-oriented software engineering, with a strong focus on the needs in automotive industry.

Christopher Preschern received his Masters degree in Telematics from Graz University of Technology in 2011, focusing on software product lines, automation systems and IT-security. He is working on his PhD in Electrical Engineering at the Institute for Technical Informatics at Graz University of Technology. His research focuses on safety and security in embedded systems with focus on patterns for embedded architectures.

Christian Kreiner graduated and received a PhD degree in Electrical Engineering from Graz University of Technology in 1991 and 1999, respectively. From 1999 to 2007, he served as the head of the R&D department at Salomon Automation, Austria, focusing on software architecture, technologies, and processes for logistics software systems. He was in charge to establish a company-wide software product line development process and headed the product development team. During that time, he led and coordinated a long-term research programme together with the Institute for Technical Informatics of Graz University of Technology. There, he currently leads the Industrial Informatics and Model-based Architectures group. His research interests include systems and software engineering, software technology, and process improvement.

Managing ERP Configuration Variants: An Experience Report

Andrea Leitner
Institute for Technical Informatics
Graz University of Technology, Austria
andrea.leitner@tugraz.at

Christian Kreiner
Institute for Technical Informatics
Graz University of Technology, Austria
christian.kreiner@tugraz.at

ABSTRACT

The concepts of Software Product Line Engineering (SPLE) have been adapted and applied to enterprise IT systems, in particular the ERP systems of a production company. Based on a 2-layer feature model for the domain of the company's business processes, individual, albeit similar division's ERP system configurations can be derived by feature selection forming a variant description model. It is indicated that regular release upgrades can also benefit from the SPLE approach.

The customization capabilities of the ERP platform are captured in another model; building up this model is automated according to information extracted online. As well, customizing an ERP system – based on the models mentioned – is performed online with the help of a connector developed in this project.

Quantitative analysis and lessons learned during the project conclude this experience report.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Domain engineering

General Terms

MANAGEMENT, ECONOMICS

Keywords

Software product line engineering, IT management, Enterprise resource planning, experience report.

1. INTRODUCTION

Systematic, "strategic" reuse as suggested by Software Product Line Engineering (SPLE), promises the reconciliation of business goals and flexibility – within a clearly focused domain [9]. Especially in the last decade, this approach has been proved to be useful a number of times, especially by producers of consumer and software products [16, 18, 14].

KOPLE '2010, October 17th, 2010, Reno, NV, USA

In the pilot project reported here, we tried to apply the SPL approach within a production company with several autonomous divisions doing their business in different countries for a number of customers. In this case, IT is seen as a supporting process helping to run the main operational business processes that have nothing to do with software production. However, reuse in their division's IT landscape is very much desirable, most prominently in their ERP solutions. Due to the same nature of the division's businesses, a domain forms naturally. Equally well, business process variants will occur due to local peculiarities like local customer mixes, laws, production facilities etc.

The paper is structured as follows. In Section 2, we first introduce the business context and scenarios. We describe the architecture and the technical realization of our SPL. Section 3 describes the cost model we used to evaluate our results. Section 4 finally interprets the results and gives an overview on the lessons learned during this project. In Section 5 we review related work about SPL engineering, the influence of release updates on the system and its configuration, and the configuration of ERP systems in general. Section 6 finally concludes our discussions.

2. A BUSINESS PROCESS ORIENTED SPL

2.1 Business context and scenarios

The target organizations for this project were the European divisions of a metal forming company. These divisions are scattered over 6 countries. Each of the divisions typically employs a few hundred people. Internal organization is business process oriented, with defined process owners for several functional areas like production processes, purchasing, logistics, shipping, finance, quality management, human resource management, and IT processes.

There is one primarily used ERP system brand in this company. Organizational structure and internal jargon of/for processes are heavily influenced by this platform. While each division has its own ERP system and IT services, massive development of ERP or other systems is definitely out of reach of the lean organized IT departments.

Prior to the start of the project, three coarse scenarios described below have been compared in the company's context.

Isolated ERP solution development (1). Currently, development of an isolated solution for each division is the most common solution; possibly with the help of consultants. For ERP systems, this task is often referred to as "customizing".

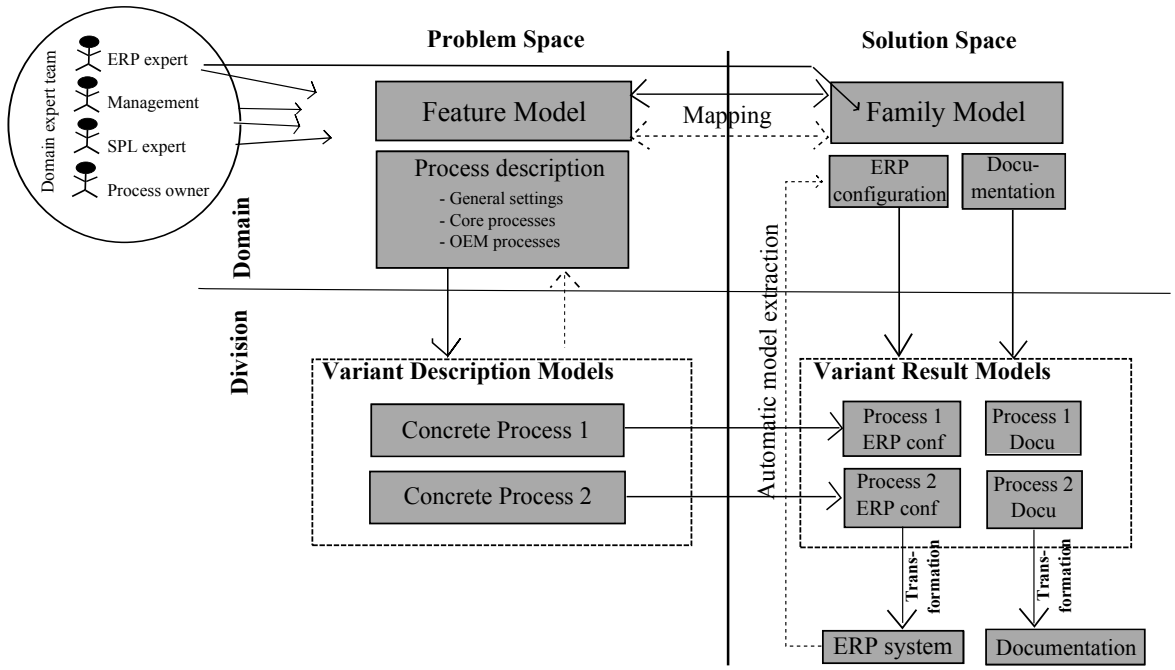


Figure 1: Conceptual model showing the relations: problem – solution resp. domain – variant (based on [1] with project extensions)

"Template"-based solutions (2). With the help of additional tools, such customized configurations can be managed, compared, copied and rolled out to several systems.

Product line solutions (3). Based on a centrally maintained repository for the entire domain of company business processes, the division's solutions can be generated within the domain's variability space. This scenario is not supported by the ERP vendor.

The comparison criteria were: initial costs, maintenance costs (regular release changes), support people needed, knowledge reuse (availability of experts on company scale), and process harmonization. The cost models used were based on the quantitative SPLE cost model in [8].

Under these criteria the three scenario models were explored in 2 dimensions starting from the current number of existing divisions: adding new divisions, and integrating a certain number of existing divisions (obviously not relevant for (1)). The results suggested the product line scenario (3) best for the transition of 3 or more divisions' ERPs to SPLE.

2.2 Business process SPL architecture

Fig. 1 shows the conceptual overview of the product line set up. The figure's structure largely follows the 4 quadrants used by the product line modeling approach of pure::variants [1], the tool we selected for this project:

The Feature Model captures all features and their relations in the domain as suggested by [7]. In our case the features are a representative subset of the company's business processes features.

Business processes have been chosen for the representation of the so called "'problem space"' because they are used to describe the procedures in the company and are generally the basis for the configuration of the ERP system. Construction rules for internal (hierarchical) structure of this model have been developed. A model validation plugin was developed to enforce the correct structure while modeling.

In addition to the business process features, another layer of abstraction has been introduced. Configuration features for specific customers have been packaged so that a customer specific feature collection becomes selectable as a simple compound feature. The realization follows conceptually the feature specialization described in [5]. The "packaged" features are represented in the same feature model, but only have relations to other features.

Each Variant Description Model below in the "problem" column records a concrete variant (concrete settings and process description) for a certain division, based on the variability defined in the Feature Model.

This means that the Variant Description Model describes the same structure as the Feature Model, but with all variants resolved.

A Family Model contains a platform's configuration options including its constraints, at least those aspects necessarily known within the SPL tool. A Family Model exists for every platform used, in our case the customizing description for the ERP system, but also

documentation. Mappings between the Feature Model and each of the Family Models must exist before a concrete platform configuration can be calculated from a Variant Description Model.

The introduction of an appropriate mapping strategy between the Feature Model and the corresponding Family Models is task for further work. In the current solution, some ERP system specifics have been included in the Feature Model, although the Feature Model should be completely independent of concrete target platforms. This simplifies the mapping in this first version of our solution.

Variant Result Model is the term used for a concrete platform configuration. When calculating this model, all other model's restrictions and constraints are taken into account. It can be used by generators to control a platform's configuration. The existence of a modular representation of the result is the big advantage of this approach. It enables the comparison of different versions and configuration and makes the result independent of different transformation engines.

2.3 Technical realization

Pure::variants, the tool we used, is shipped as an Eclipse plugin and as such can easily be extended by further plugins. In this project three plugins have been developed:

1. A model validation plugin enforces a domain specific internal structure of the Feature Model, as pointed out above.
2. In order to set up the ERP Family Model quickly and error-free, an import plugin extracts the customization menu items, its views and parameters from the ERP system (SAP), and builds up the Family Model in an automated way (c.f. "Automatic model extraction" in Fig. 1).

As a useful by-product, differences in the (large) sets of customization points of different ERP system releases can be found using pure::variants' model compare function.

3. The actual SAP ERP system customization is accomplished by a transformation plugin, by setting the customizing parameters according to the Variant Result Model.

3. COST MODEL DEVELOPMENT

The development of our cost model has been based on literature research. First it has to be mentioned that most of the cost models (e.g. [3]) act on the assumption that software has to be developed and will be reused in further projects. Contrary to this assumption we do not develop software, but configure a third party software and want to reuse the configuration knowledge, which makes most of the existing cost models applicable only partly.
Our cost model is based on the SIMPLE cost model in [2]. Three important scenarios for our estimations have been taken and adapted to our specific needs. The SIMPLE model can be adapted to fulfill the requirements for our use case. For this project we assume:

$\boldsymbol{C}_{org}$ = SPL tool development time
$\boldsymbol{C}_{cab}$ = domain engineering activities
$\boldsymbol{C}_{unique}$ = domain analysis
$\boldsymbol{C}_{reuse}$ = variant production
$\boldsymbol{C}_{prod}$ = manual configuration of the ERP system

In addition, we use an SPL completion factor C_{SPL} $(0 \ldots 1)$ that considers the reduced scope in our use case (see Sec. 4.1 for further discussions). With these assumptions we can write the formulas mentioned above as follows:

For development with an SPL

$$\frac{\sum_{i=1}^{n}(C_{cab}(pr_i) + C_{reuse}(pr_i) + C_{unique}(pr_i))}{C_{SPL}} + C_{org}(pr_i)$$

For individual system development

$$\frac{\sum_{i=1}^{n}(C_{unique}(pr_i) + C_{prod}(pr_i))}{C_{SPL}}$$

For an additional system

$$\frac{C_{org}(pr_i) + C_{cab}(pr_i) + C_{reuse}(pr_i) + C_{unique}(pr_i)}{C_{SPL}}$$

3.1 Definition of fine-grained cost model

We refine the above mentioned cost model to get more detailed estimations. Therefore we have split up the defined cost functions and have collected metrics during our ERP SPL experiment (see Tab. 1). We use the maximum number of features to get a worse case estimation. This results in the following concrete formulas:

1. SPL setup and first product

$$T_{SPL-I} \leq \frac{C_{cab}(1) + (t_F(1) * \max(n_F) + C_{unique}(1))}{C_{SPL}} + C_{org}$$

2. Lower bound for single system development (without SPL)

$$\min(T_{Dev}) = \frac{(C_{unique}(1) + C_{prod}(1) * \max(n_F))}{C_{SPL}}$$

   (a) Lower bound for single system update (without SPL)

$$\min(T_{Dev}) = \frac{C_{prod}(1) * \max(n_F))}{C_{SPL}}$$

3. Additional product (with SPL)

$$T_{SPL-\Delta} \leq \frac{C_{unique}(2) + C_{cab}(2) + t_F(2) * \max(n_F))}{C_{SPL}}$$

For the development without SPL it has to be mentioned that a single system update can be seen as a special case of the system development. Both tasks are equal except the fact that we don't need to do the domain analysis (C_{unique}) again for system updates. Obviously there is no need for a separate formula for the development of an additional product in single system development.

The formula for additional product development has to be discussed shortly. Also the gross cost function includes the C_{org} it is not relevant in the detailed formula because the tool development has to be done only once. Further steps are automated through this tools, so that we can assume the C_{org} term to be zero.

One formula seems to be missing here. We are assuming the time to update the system with the SPL approach to be approximately 1 hour. This approximation is based on the average feature selection time and the average number of features selected. We further assumed the need for additional time to execute the actual customizing, which results in the estimate of one hour for the automated system update.

4. EVALUATION

Three divisions' processes were incorporated into the business process oriented product line. Each of their ERP systems configurations was derived by selecting business process features and entering parameters accordingly.

4.1 Quantitative Analysis

Besides others, the metrics in Tab. 1 have been collected during the project. The following comments are necessary here:

It has to be mentioned that there is no uniform measurement unit in Tab. 1. This is only for a better overview and readability and has to be regarded when calculating the break even points.

To keep this trial project reasonably small, we deliberately narrowed the business process scope and focused on 3 divisions only. The SPL completion status was estimated approx. 15% by the domain expert team in contrast to full completion.

This experiment was carried out iteratively on 3 operating divisions' ERP systems that we tried to integrate into our business process oriented ERP product line. This aligns to the "integration of existing divisions" dimension in the scenario analysis (Sec. 2.1). No new divisions were set up during that time.

No serious quantitative data were available about manually customizing a single ERP system in the same scope as this project (Sec. 2.1, scenario (1), Isolated development). However, the SPL-derived configuration for a division was entered manually into an ERP system yielding a productivity metric. In this way, comparisons of the two approaches at least allow for a worst case estimate.

The "template" approach – scenario (2) was not investigated in this experiment, because it fell short in the scenario analysis (Sec. 2.1); furthermore, necessary (expensive) vendor tools were beyond reach.

Based on these metrics collected, cost drivers can be calculated (cf. Sec. 3.1) for product line setup, derivation of an additional ERP system, and following an ERP release update (Tab. 2). These figures should not be mistaken with a complete setup of an ERP system (release) within an organization, since that can involve a lot more, like preparation, testing and possibly rolling back an ERP update, documentation, training, and even organizational change.

Tab. 3 finally shows the extrapolated break even points comparing isolated development vs. SPL development for three cases: 1 resp. 2 systems under product line control following ERP vendor release updates; for 4 or more systems, the product line approach always is advantageous.

Metric		Iterations (divisions) Initial (1)	2nd	3rd
Domain analysis	C_{unique}[hr]	28	3	1
Domain eng. & SPL setup	$C_{cab} = T_{DM} + T_{SM}$			
Domain modeling (Feature Model)	T_{DM}[hr]	25	7	2
ERP platform modeling (Family M.)	T_{SM}[hr]	2	1	1
SPL Tool development	$C_{org} = T_{TV} + T_{TI} + T_{TC}$			
Tool/model valid. plugin	T_{TV}[hr]	15		
Tool/FM import connector	T_{TI}[hr]	56		
Tool/ERP cust. connector	T_{TC}[hr]	112		
Variant production	C_{reuse}			
Variant Description M.	T_{VDM}[min]	48	24	7
No. of features selected	n_F[F]	48	61	17
Avg. feat. selection time	t_F[sec/F]	60	24	24
run transform. + ERP cust.	t_{run}[hr]	≈ 1		
Manual ERP cust. / Feat.	C_{prod}[sec/F]	120		
SPL completion reached in project (estimated, see text)	C_{SPL} [%]	15%		

Table 1: Metrics collected during ERP SPL experiment

Cost drivers	Est. system development effort (61 Feat.) Isolated [hr]	SPL oriented [hr]
Setup + first system	$\min(T_{iDev}) \approx 200$	$T_{SPL-I} \approx 556$
additional system		$T_{SPL-\Delta} \approx 76$
release update	$\min(T_{iUpd}) \approx 13.4$	$T_{run} \approx 1$ (run transformation)

Table 2: Cost drivers (calculated from Tab. 1)

Break even (SPL better) after no.	
Systems migrated to product line	release updates (worst case)
1	≤ 29
2	≤ 18 (9 per system)
3.9	*none*

Table 3: Break even extrapolation: SPL vs. isolated development scenarios (with figures from Tab. 2)

4.2 Lessons learned

Define/choose the domain focus. The domain focus – the company's business processes – was deliberately chosen without much compromise to reflect a common ground of understanding and jargon accepted by most of the company's employees. Especially an additional feature model layer, the even higher abstracted customer-specific process packages, is quite far abstracted from the underlying ERP system's way of configuration. This strategy helped a lot in communication and feature modeling.

Domain expert team. The term domain expert seemed to be confusing in the early discussions of the development process of our domain. Who is this domain expert? Is there really one expert who is responsible for the whole SPL development?

In our work we identified a team of four different expert groups which have an influence on different aspects of the domain and thus introduced the term domain expert team.

We got access to company-internal ERP system experts, a business process (modeling) expert, and a person covering the business and organizational goals. Product line and methodical know-how was supplied externally.

This domain expert team reflects the different views on the domain. The ERP system experts provide a technical view. The ERP system is very complex and built from various modules. Due to its complexity there is usually one expert for each module per system.

The business process expert introduces a process view. Typically there are several process owners which have detailed process knowledge. Process differences are causes of variants and thus are very important in the variability modeling process.

The management view specifies the scope of the domain, which is an essential task for the economic success of the SPL approach [13].

To combine all this experience and information in a structured way, methodical support is essential.

The existence and importance of this domain expert team has to be kept in mind during the whole SPL lifecycle to ensure the success.

However, the constitution of this team may vary from domain to domain. For a more general formulation of our findings four types of experts can be identified as necessary parts in an SPL project. Depending on the complexity of the domain one person can unite more than just one expert role. On the other side it can be useful to have a group of experts for each expert role as in our example.

The definition of the SPL scope and goals, as well as the alignment of these goals with the organization's business goals is part of the management expert. Further, it is important to have experts to provide methodical background, variant expertise, and target platform expertise.

The project core team was serving the development of all aspects within the Business-Architecture-Process-Organization (BAPO) model [16, 11].

Variants in space and time. While the division's configurations – existing in parallel – are a natural concept in SPLE, migrating to an ERP release update can also be seen as variant that can be controlled by SPL models as they give rise to substantial effort, similar to setting up a new variant. On the other hand, migrating to a new release is typically not driven by functional progress in the SPL domain context [12]. This implies, that existing variant models can probably be applied on the new platform release with little or no change at all. Furthermore, a release migration should be prepared centrally for the entire product line by setting up a new platform description, namely the Family Model (see Tab. 2).

Reuse and learning. The processes in the investigated divisions seemed to be similar. This assumption is backed by the fact that only small changes became necessary in the domain models when incorporating an additional division (taking into account the number of selected features, Tab. 1).

Despite supporting similar processes, the existing realizations in the ERP systems may differ to a high degree. The incremental domain modeling approach proved useful for comparing even these different implementations systematically on the undisputed, common basis of a (the) common business process domain model structuring. This greatly helps to develop a commonly accepted and clear understanding about commonalities and variabilities across divisions, a prerequisite for a common company-standard ERP implementation.

Automation of modeling. Due to the sheer quantity of customization parameters in the ERP system, importing these properties to an SPL tool (pure::variant's Family Models) has to be automated. Fortunately, the customizing parameters are already represented in a tree structure in the ERP system. Hence the structuring information can be imported from the source platform as well, resulting in an equivalent representation both in the ERP system and the variant management tool. The effort is reduced to the one-time implementation of the import plugin (see Tab. 1).

As a second side effect, the feature-relevant aspects of different platform releases can be compared - as long as the meta-model governing this import can accommodate the platform's configuration options into a new version's family model.

Quantitative prediction models as described in Sec. 2.1 proved extremely helpful especially for company-internal project marketing. Initially, the SPL idea, being intrinsically more complex than isolated development, was hardly regarded realizable by many of the persons involved.

No vendor support. The ERP system vendor, as well as consultants we contacted, were reluctant to provide support to this project. No clear reason was given; to us, apparently there was no interest in such a reuse approach.

5. RELATED WORK

Software Product Lines have been suggested as a systematic reuse approach for software intensive systems [9]. That has been proven a number of times (e.g. [15, 6, 4]). For further examples see the SPLC Hall of Fame website [14].

The product line approach to reuse comes along with two fundamental development cycles: domain engineering (core asset development) and application engineering (a.k.a. product development) e.g. [10, 9]. As illustrated in [11] both have different time constants. Due to a common long-lived

(domain) architecture the artifacts developed in an iterative domain engineering process can be reused for several application developments. The iterative nature of domain engineering allows for enrichments and corrections from product life cycles, determined by the third key process, a management process reflecting the business goals. However, a comparably stable and long-lived domain architecture and its reusable artifacts remain key for reuse. In this way, market needs, like economical product development constraints, short time-to-market, presence on multiple markets, can be met. Furthermore, the architecture can also evolve over time, thus effectively postponing it's expiry date, the so called Klein horizon [17].

In our setting due to using a "standard" third party (ERP) product, still another cycle time comes on to stage: a more or less regular "release heartbeat" of these products.

According to a study among 317 german companies [12], more than 50% follow their core system vendor's recommendations to regularly follow their product updates. For these migrations, they spend a substantial share of their IT budget: 44% of these companies spend more than a quarter of their total IT budget, 52% more range between 10-25% update costs. Even in the light of dramatically shrinking IT budgets, 59% stated, they will probably not change this policy in the future.

As far as we know until now there haven't been any experiences in using a Software Product Line or a similar approach to configure a third party software and systematically reuse configuration knowledge in various divisions. Therefore it is hardly possible to compare our results to previous experiences.

6. CONCLUSION

In this paper, the feasibility of applying Software Product Line Engineering (SPLE) to a production company's IT systems was shown, most prominently and initially to their (third-party) ERP systems. A domain model contains all features and configuration constraints of the company specific business processes. A domain specific internal structure is enforced by a SPL tool plugin. Within the feature model's limits, the ERP variants to be used in the company's divisions can be defined by selecting possibly parameterized features.

As for the ERP platform used here, its customization options are extracted automatically, and used to transform a defined variant into a customized ERP system instance. Equally well, consistent other products like documentation can be assembled from the same variant description.

Quantitative analysis of the project extrapolates to a break-even point at 4 systems or more. As ERP release updates also benefit from the product line, the break-even point can be even lower, dependent on the number of release upgrades over a system's lifetime. A number of qualitative lessons learned is presented as well.

7. ACKNOWLEDGMENTS

We gratefully thank Danilo Beuche and pure systems GmbH for their support.

8. REFERENCES

[1] D. Beuche and M. Dalgarno. *Software Product Line Engineering with Feature Models*. pure systems GmbH, visited 2010.

[2] G. Böckle, P. C. Clements, J. D. McGregor, D. Muthig, and K. Schmid. A Cost Model for Software Product Lines. In *PFE*, pages 310–316, 2003.

[3] B. Boehm, A. W. Brown, R. Madachy, and Y. Yang. A Software Product Line Life Cycle Cost Estimation Model. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 156–164, 2004.

[4] H. P. Breivold, S. Larsson, and R. Land. Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies. In *Euromicro SEAA'08 Proceedings*, pages 232–239, Washington, DC, USA, 2008. IEEE Computer Society.

[5] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[6] J. C. Dager. Cummins's experience in developing a software product line architecture for real-time embedded diesel engine controls. In *SPLC'00 Proceedings*, pages 23–45. Kluwer Academic Publishers, 2000.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[8] J. P. Nobrega, E. S. de Almeida, and S. Meira. InCoME: Integrated Cost Model for Product Line Engineering. *Software Engineering and Advanced Applications, Euromicro Conference*, pages 27–34, 2008.

[9] L. Northrop and P. Clements. *Software Product Lines: Practices and Patterns*. 2002.

[10] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering*. Springer, 2005.

[11] E. Rommes and J. Wijnstra. Implementing a reuse strategy: Architecture, process and organization aspects of a medical imaging product family. In *HICSS '05. Proceedings*, pages 312a–312a, Jan. 2005.

[12] A. Schaffry. Release-Wechsel verschlingen IT-Budgets (in german). CIO news, http://www.cio.de/878198, 2009-05-04.

[13] K. Schmid, S. Thiel, and J. Bosch. *Scoping*. ITEA, visited 2010.

[14] SEI Carnegie Mellon. Product Line Hall of Fame. http://splc.net/fame.html, visited 2010.

[15] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber. Introducing PLA at Bosch Gasoline Systems: Experiences and Practices. In *SPLC'04*, pages 34–50, 2004.

[16] F. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in action*. Springer, 2007.

[17] R. van Ommering. Software reuse in product populations. *Software Engineering, IEEE Transactions on*, 31(7):537–550, July 2005.

[18] D. M. Weiss, P. C. Clements, K. Kang, and C. W. Krueger. Software Product Line Hall of Fame. In *SPLC'06*, page 237, 2006.

MADMAPS - Simple and systematic assessment of modeling concepts for software product line engineering

Andrea Leitner, Reinhold Weiß, Christian Kreiner

Institute for Technical Informatics, Graz University of Technology, Graz, Austria

Abstract. Domain modeling is a key task in the development of a software product line. We identified two popular modeling paradigms to be predominantly used in practice: feature-oriented domain modeling and domain specific modeling. The appropriate choice of the modeling paradigm is a crucial decision for the development of an efficient and easy to use domain model.
In order to take such a decision systematically, we propose MADMAPS, a simple method to assess the nature of the domain.
MADMAPS is based on multi-attribute utility theory. The core part is a set of four discriminating criteria describing the characteristics of a domain. The result is either a recommendation for one modeling paradigm, or to split the domain in homogeneous subdomains.
Four use cases have been used to extract assessment criteria, as well as to evaluate MADMAPS. The evaluation is based on the complexity of the resulting domain model. It can be shown that the model complexity with the proposed approach is always lower than the complexity of a model represented by the other approach.

1 Introduction and motivation

Software product lines (SPLs) are a viable methodology to improve engineering of software intensive systems. Northrop et al. [1] highlight the importance of a well structured and documented domain model, since this is the central part of an SPL. But what does this mean? What makes a domain model well structured? We argue that a well structured representation of the domain model depends on the nature of the domain, and further, that the choice of an appropriate modeling paradigm is a first step towards a well structured domain model.

The first step before starting to model a domain is to select a modeling paradigm. This decision is often taken implicitly. In some cases, even workarounds become necessary just because the chosen paradigm is not well aligned with the nature of the domain. Currently, the most common paradigms are Domain Specific Modeling (DSM) and Feature-Oriented Domain Modeling (FODM).

Problems in the domain modeling paradigm selection process during one of our recent projects have been the motivation for this more systematic decision making process. We have extracted several characteristics of domains and derived criteria from them.

Besides having a concise and easy to understand domain model, we aim at an efficient way of product definition and derivation. This is worth striving for, as this step is done hopefully many times in an SPL. Both, domain modeling and product derivation, are heavily influenced by the modeling paradigm. Again, a "fit" paradigm and language to describe variability and variations in a concise manner helps to make this step more efficient.

The contribution of this paper is a simple method for systematic evaluation of a domain and decision support for a specific modeling paradigm. Fig. 1 shows the basic flow of the proposed decision making approach. The input is a domain, which has been bound in a previous scoping step and the result is a recommendation for one modeling paradigm.

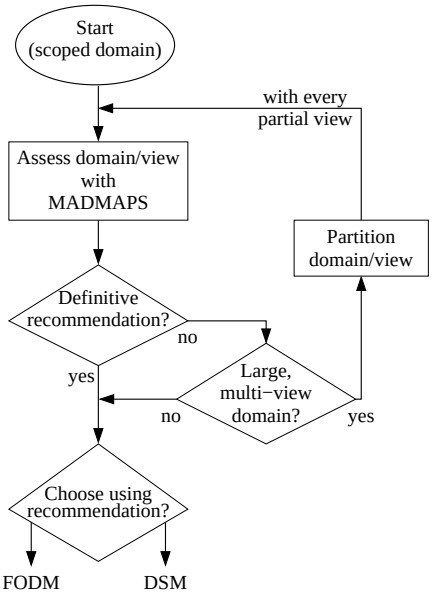


Fig. 1. Proposed flow for domain modeling paradigm selection with MADMAPS

Summarized, we

- take advantage of past research efforts.
- get a simple decision making support, which can be used at an early stage of development.
- define simple criteria that still work fine for a general suggestion.
- use the original paradigm definitions without additional extensions to exploit their advantages (e.g. the advantage of FODA is its simplicity) and to be independent from specialized tool implementations.

Sec. 7 summarizes the most important related work. Sec. 2 gives some background information about the underlying concepts. Sec. 3 and Sec. 4 describe the MADMAPS decision making method and underlying theory. Finally, Sec. 5 presents four case studies that have been part of the development process and evaluation. Sec. 8 concludes our work.

2 Background

This section introduces some theoretical background that serves as a base for our approach.

2.1 Introduction to Multi-Attribute Utility Theory

Multi-attribute utility theory[1] (MAUT) can be used as a decision aiding technology, if one alternative from many should be chosen depending on multiple attributes. The most important steps are listed below [2]:

Identify alternatives which should be evaluated (further noted as *alt*).
Establish assessment criteria (attributes) that should be used in the evaluation process. It is advisable to focus only on the most important and relevant criteria.
Determine weighting factors w_{crit}. Each attribute is weighted by its importance. The weighting factors are determined by a pairwise comparison of criteria.
Assessment of alternatives with respect to the defined criteria $c_{crit}(alt)$. For the assessment a scale has to be defined.
Calculate utility value. In this last step, the utility values $u(alt)$ for all alternatives *alt* are calculated:

$$u(alt) = \sum_{crit=1}^{n} (w_{crit} * c_{crit}(alt))$$

Finally, a decision has to be taken. The resulting utility values serve as a quantitative base for this decision.

2.2 Domain modeling paradigms

As mentioned before, we support the decision making for a specific domain modeling paradigm. The alternatives are the two, in our opinion, main modeling paradigms: domain specific modeling and feature-oriented domain modeling.

Domain specific modeling
Domain specific modeling (DSM) aims at the use of a higher level of abstraction and the direct usage of concepts and rules from a specific problem domain. Domain specific languages (DSL) are used to model a system within that domain. A key characteristic of DSLs is their focused expressive power [3], enabling the generation of products directly from these high level specification [4]. We focus on graphical DSLs in this paper because of the experience and results available from our case studies.

Feature-oriented domain modeling
By feature-oriented domain modeling (FODM) we are mainly talking about

[1] `http://ddl.me.cmu.edu/ddwiki/index.php/Multiattribute_utility_theory`

feature-oriented domain analysis (FODA) proposed by Kang et al. [5]. This approach has become the basis of many other feature-oriented approaches (e.g. FORM [6]).

"*Features are the attributes of a system that directly affect end-users.*" [5]

For the representation of features, a tree-structured feature model is defined using *consists-of* relations. The relations are either marked as mandatory, alternative, or optional. Further composition rules (e.g. "requires" or "mutually exclusive") are used to express relations between features that cannot be expressed in the tree structure itself [5].

Over the years, several extensions to the original FODA appeared (e.g. cardinality-based feature modeling [7]). These extensions are not investigated here.

3 Multi-attribute domain modeling approach for paradigm selection (MADMAPS)

This section describes a simple method aiding in systematic decision making on which domain modeling approach to choose.

The input of our evaluation method is a well-scoped domain.

3.1 MADMAPS – overview

The setting in our evaluation is slightly different from the original MAUT approach. The investigated alternatives in MADMAPS are FODM and DSM. Defined criteria describe characteristics of domains. The resulting utility value indicates how well the approach fits to the nature of a given domain. So we still compare the alternatives – the domain modeling paradigms – according to a set of criteria defined below.

In contrast to the original MAUT we assess the criteria for each new domain to get a domain specific w_{crit} vector. This vector is then used to calculate the utility values. The $c_{crit}(alt)$ values remain constant as they describe the criteria score of the paradigms.

Identification of alternatives We focus here on DSM and FODM as described in Sec. 2.2.

Establishing assessment criteria The first step towards appropriate assessment criteria is the identification and comparison of the main characteristics of the two paradigms. We searched particularly for strong discriminating criteria that are observable at an early stage of SPL setup. A training set of three domains (described in Sec. 5 Case Studies 1 – 3) was used for criteria definition. Tab. 1 lists the resulting MADMAPS criteria C1 – C4. A detailed description of the foundation of this criteria extraction is given in Sec. 4.

Criterion	DSM	FODA
C1 Fixed relations $\geq$ variable relations	**4**	**17**
C2 Several instances of elements	**31**	**4**
C3 Different binding times/views	**8**	**15**
C4 Domain model used by non expert	**8**	**15**

Table 1. Criteria C1 – C4 and assessments c_{crit} of domain modeling alternatives DSM and FODA

Questionnaire for determining weight factors Following the criteria C1 – C4 several questions have been formulated that we expect to be answerable in very early domain understanding phases. The questions are listed below:

Q1 Are there more fixed relations than variable relations? (Formula 1 (see Sec. 4.1) evaluates to true.)
Q2 Should it be possible to use several instances of an element?
Q3 Should there be more than one binding time or more than one view in the domain representation?
Q4 Should the domain model be used by a customer who is not a domain expert? (for example: car configurator)

The questions are answered following the Likert scale[2]. Tab. 2 shows the possible answers and the corresponding weight. This is later used to calculate the utility values and, thus, assess the applicability of the domain modeling paradigm to the specific domain.

Assessment of alternatives Tab. 3 shows the assessment schema used in MADMAPS. These values are used to evaluate the alternatives (DSM and FODM) in respect to the criteria C1 – C4. Tab. 1 finally lists the derived assessment values.

Strongly disagree	**-2**
Disagree	**-1**
Neither agree nor disagree	**0**
Agree	**1**
Strongly agree	**2**

Table 2. Likert scale for criteria weighting factors w_{crit}

Poorly or not at all	**0 - 2**
Fair	**3 - 5**
Good or complete	**6 - 8**

Table 3. Assessment schema of criteria fulfillment $c_{crit}(alt)$

[2] `http://www.socialresearchmethods.net/kb/scallik.php`

Calculate utility values and decision Sec. 2.1 describes the calculation of utility values.

As mentioned before, the resulting utility value is only an indicator for a representation paradigm. The final decision has to be taken by a domain expert, but can be based on this systematic assessment of the domain.

4 MADMAPS - Assessment criteria rationale

This section links the chosen criteria to certain characteristics of DSM and FODM, respectively. In the course of our investigations we were looking for discriminating characteristics, which show a strong tendency towards one of the two paradigms.

4.1 Ratio between fixed and variable relations (C1)

A domain model in general consists of elements and relations. Relations can further be divided into fixed and variable relations. Fixed relations do not vary between different products. Variable relations are interesting in this context. Variable relations mean relations that are either defined between two elements where the kind of relation changes, or a relation where the target element changes. For a very rough estimate we define the following indicator for high complexity and variability:

$$|elements| \leq |relations_{variable}| \tag{1}$$

A somewhat easier to grasp indicator for the nature of the domain is the ratio of fixed and variable relations. If for a given domain the following is true, this might be an indicator for a given structure common to all products:

$$|relations_{fixed}| \geq |relations_{variable}| \tag{2}$$

4.2 Instantiation of elements (C2)

As stated in [8], domain objects are good candidates for abstraction. This is an important guideline for the design of a DSL in practice. Generally, there is not just one instance of such objects in the real world. Therefore, it should be possible to instantiate objects in the domain representation as well. DSLs are intentionally designed for instantiation, whereas FODM is not. In the original definition of feature orientation [5] there is no such concept. As stated in Sec. 2.2, an extension to represent instantiation in feature models has been proposed by [7]. This extension has several disadvantages. Moreover, the extension has only been implemented by one prototype tool.

4.3 Different binding times and views (C3)

The variability mechanisms are predefined in the FODM approach and derived and implicitly codified in the language in DSM.

One important differentiating factor seems to be the time when flexibility is useful and needed. This becomes especially important when one wants to represent different abstraction levels or binding times in one domain model. This can be accomplished easier with feature-oriented approaches.

Binding times are defined in the original FODA definition [5]. There are three types: compile-time, load-time, and runtime features. In case of the FODM approach it is easier to describe variability with different binding times, of course only if the generic platform supports those. A DSL need not have an existing code base. Instead, code is generated for each new product model. Therefore, it is harder to introduce different binding times in a DSM approach.

4.4 Target group (Domain model used by non-expert, C4)

Features are end-user visible characteristics of a domain. This means they are an important mean to support the communication between developers and customers. This is another reason why a concise and easy to understand representation is essential. A DSL mostly requires a deeper technical understanding that customers not always possess.

5 Investigated use cases

This section introduces and describes four different projects realized with an SPL approach. Each of these projects addresses a unique domain. Thus, the requirements of the projects are quite different.

5.1 Case 1: Configuration of an ERP system

The aim of this project was the systematic reuse of configuration knowledge for Enterprise Resource Planning (ERP) systems of a group of companies [9]. Each of these companies acts in the same industry, they share similar customers and they have similar business processes. The regarded ERP system has to be configured according to the business process of the specific company. Due to the complexity of this configuration process, systematic reuse of configuration knowledge helps to reduce efforts in terms of money and time, and to improve the quality of the ERP instances. Business processes form the SPL architecture.

5.2 Case 2: Fish farm automation

The aims of a fish farm automation system is to make the work of the fish farm owner easier and to save resources. The main functional requirements for a fish farm automation system are feeding and oxygen supervision including an

alarm system. In addition, a water level supervision, pH-value measurement and standard functions like switches and lights have to be realized.

An important characteristic of this domain is the existence of several instances of elements. For example, a fish farm consists of a certain number of ponds. The elements may be assembled in several ways. In fact, the focus is much more on the assembly of elements than on supported functionality. Normally, no two fish farms look exactly the same. There may be several ponds where each pond may or may not have different supervision and feeding systems.

5.3 Case 3: PLC controlled inventory system

The domain in this case is a logistics system which is built of conveyors, rotary tables, cranes and high bay racking components. With this product line it should be possible to generate automation system software, documentation, etc. for various assemblies of these components.

The arrangement of elements is also more or less variable, which results in a high number of variable relations. The number of elements is very low, since there are only the aforementioned components available. However, these elements may be instantiated several times. The selection of different functionality is not required. Different binding times and abstraction levels are not important here as well. Fig. 2 shows a sample application model.

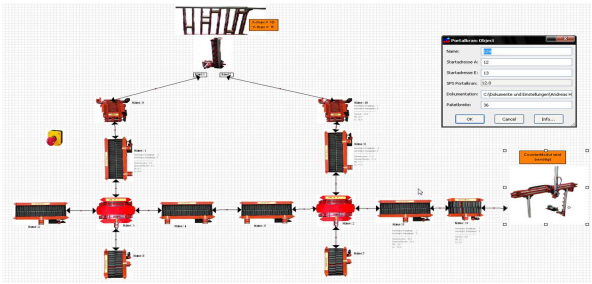


Fig. 2. Sample application model for a PLC controlled inventory system (realized with MetaEdit+ DSL)

5.4 Case 4: Control unit for an HEV

The target in this project[3] is the development of a generic architecture for an hybrid electrical vehicle (HEV) [10] control unit. Since this is an embedded

[3] `http://www.iti.tugraz.at/hybcons`

system, there are many connections to the environment (i.e. the overall system). This makes the domain very complex.

In the domain analysis phase we faced the problem that many criteria seem to be very important and necessary in some part of the domain, or another. There is certainly a focus on functionality, since the control unit provides functionalities. On the other hand there is also a focus on assembly, because the provided functionality is dependent on the layout of the drivetrain (e.g. full electric drive is only possible with a clutch between electric motor and combustion engine).

The result is not as clear as in the case studies above. Both utility values are positive and there is no real indication which approach is appropriate. Following the flow described in Fig. 1 we split domain into several subdomains, one for each viewpoint. The viewpoint identification resulted in a software view, an ECU view, a mechanics view and a safety view. In the next iteration step MADMAPS was applied to each of the subviews.

We applied MADMAPS to the software view, and to the mechanics view representing the drivetrain topology. Now, the situation looks quite different. For the software there is no need for several instances of elements or different assemblies of elements anymore and the drivetrain (mechanical view) can be described in a graphical representation. Fig. 3 shows an example for both representations. Due to the heterogeneity of subdomains a combined multi-modeling representation has been proposed in [11].

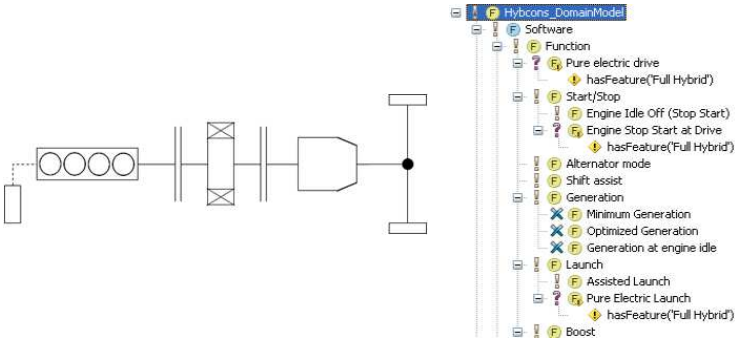


Fig. 3. Sample application model for a hybrid electrical drivetrain (ecore-based DSL) and a sample software feature model (pure::variants)

6 Lessons learned

Complexity. Previously [12], we defined metrics to evaluate the resulting domain model. One useful quality metric is the complexity of the resulting

model. Keeping complexity as low as possible is important in several aspects. First of all, it improves the usability and maintainability of the domain model. An investigation of the described domains show that the modeling paradigm suggested by MADMAPS always results in lower complexity.

MADMAPS results. Tab. 4 shows an overview of the resulting utility values. Additionally, the resulting complexity values (CV) are given for both paradigms in order to verify the results.
The utility values show that the approach gives a clear recommendation for all our test domains. For Case 4 we interviewed experts working with this domain models and asked them how well they are satisfied with the representation and what they like best. An evaluation of this interview showed that the experts agree and welcome the combination of the two different representations. In particular the graphical representation for the mechanics part seems to be much more suited than a feature oriented approach.

General considerations. Why should one use our approach. First, there is no real overhead because all the information necessary to answer the questionnaire is gathered during domain analysis. Domain analysis is an integral part of the domain engineering process and has to be performed anyway. Due to the simple and abstract formulation of the questions, knowledge from domain analysis should be enough to answer them. One benefit is, that the proposed criteria can be used during domain analysis to have a structured guideline how to investigate the domain. The big advantage however is that the decision is grounded on a systematic base.

Case	**Domain**	**DSM**	**FODM**	**Recommended**	**CV** (DSM/ FODM)
1	ERP system	**-62**	**11**	FODM	257/167
2	Logistics system	**46**	**-41**	DSM	65/ -
3	Fish farm automation	**38**	**-56**	DSM	70/ -
4	HEV CU (control unit)	**55**	**49**	-	78/80
	HEV CU - Software	**-38**	**56**	FODM	30/25
	HEV CU - Mechanics	**11**	**-24**	DSM	22/36

Table 4. MADMAPS utility values and complexity values (CV) for the case studies described

7 Related work

The importance of the appropriate representation in DSML specifications is mentioned in [13]: "*the correct representational paradigm depends on the audience, the data's structure, and how users will work with the data*". We extend

this statement to the next level and argue that not only the representation is important, but also the paradigm to create this representation in an effective way.

Some authors have investigated the differences between modeling approaches. In [14] the feature-oriented domain analysis approach has been compared to Organization Domain Modeling (ODM). For the authors these seemed to be the most important approaches. ODM is also based on features. The feature definition is more general than in FODA. One major difference is that ODM postulates the need for a flexible architecture. It is stated that a generic architecture is not suitable for domains with a high degree of variability. This statement is similar to our observation that DSM is better suited for domains with many variable relations, because of a flexible architecture.

Czarnecki [15] investigates the relation between feature models and ontologies. The major conclusion from his work is, that extensions of the original feature oriented approach are used to bring it closer to the expressiveness and formalism of ontologies. As stated before, these extensions are at the expense of the simplicity, which is a major advantage of feature models. Furthermore, the authors propose to combine FODM and DSM, which confirms our observations that it is sometimes not enough to model the entire domain with one representation. In contrast to our research, they do not split domains. Instead, two approaches are used to represent the same content in different views.

Haugen et. al [16] describe a separated language approach to specify variability in DSL models. They propose a Common Variability Language (CVL) and according variability resolution mechanisms embedded in the OMG metamodel stack. This allows to describe variability in potentially all MOF-based languages, including UML, as well as MOF- and UML-profile based DSLs. While being a general and clean approach to handle variability, it does not seem directly applicable to feature abstraction hierarchies and their complex constraints.

8 Conclusion

This work introduces a simple approach for the systematic selection of an appropriate domain modeling paradigm. Four criteria have been extracted from characteristics of the methods investigated - feature oriented domain modeling and domain specific modeling. We use these criteria to assess the nature of a domain in respect to a systematic selection of a modeling approach.

The major advantage of the MADMAPS approach is a decision which is based on a systematic method. As a result we can show that the use of the proposed modeling paradigm always results in a domain model with lower complexity.

Acknowledgment

The authors would like to acknowledge the financial support of the “COMET K2 - Competence Centres for Excellent Technologies Programme” of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the

Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG). We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

References

1. L. Northrop and P. Clements, *Software Product Lines: Practices and Patterns*. Addison Wesley, 2002.
2. J. W. Altschuld and B. R. Witkin, *From Needs Assessment to Action: Transforming Needs into Solution Strategies*, 1st ed. Sage Publications, 2000.
3. A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, 2000.
4. S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.
5. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.
6. K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.
7. K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *Proc. of the Int. Workshop on Software Factories*, 2005, pp. 16–20.
8. S. Thibault, "Domain specific languages: Conception, Implementation and Application," Ph.D. dissertation, l'Université de Rennes 1, 1998.
9. A. Leitner and C. Kreiner, "Managing ERP configuration variants: an experience report," in *Proc. of the 2010 Workshop on Knowledge-Oriented Product Line Engineering*, ser. KOPLE '10. New York, NY, USA: ACM, 2010, pp. 2:1–2:6.
10. M. Ehsani, Y. Gao, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design*, 2nd ed. CRC Press, 2010.
11. A. Leitner, C. Kreiner, R. Mader, C. Steger, and R. Weiß, "Towards multi-modeling for domain description," in *2nd Int. Workshop on Knowledge-Oriented Product Line Engineering: KOPLE 2011*, ser. SPLC'11, 2011, pp. 1–6.
12. A. Leitner, R. Weiß, and C. Kreiner, "Analyzing the complexity of domain models," in *ECBS12, Proc.*, 2012.
13. S. Kelly and R. Pohjonen, "Worst Practices for Domain-Specific Modeling," *Software, IEEE*, vol. 26, no. 4, pp. 22 –29, 2009.
14. K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.
15. K. Czarnecki, C. H. Peter Kim, and K. T. Kalleberg, "Feature Models as Views on Ontologies," in *SPLC '06: Proc. of the 10th Int. on Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 41–51.
16. O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding Standardized Variability to Domain Specific Languages," in *Proc. of the 12th International Software Product Line Conference*, 2008, pp. 139 –148.

Analyzing the complexity of domain models

Andrea Leitner, Christian Kreiner, Reinhold Weiß
Institute for Technical Informatics
Graz University of Technology,
Graz, Austria
{*andrea.leitner, christian.kreiner, rweiss*}*@tugraz.at*

***Abstract*—Software Product Lines (SPL) are a viable method for systematic reuse. An essential part of an SPL is the domain model. In order to be efficient the domain model should have an as low as possible complexity. In this way the usability and maintainability of the domain model can be improved. This is important because of the long lifecycle and the hopefully high number of derived products. One important influence factor for the complexity is the choice of the domain modeling paradigm. Another point is the design of the model. Various aspects can be modeled in different ways resulting in different levels of complexity.**

To the best of our knowledge there is no simple metric to measure and compare the complexity of different domain representations. To make it clear we do not measure the complexity of the domain itself, but of the representation. This work suggests simple metrics to estimate interface, element and property complexity, in our opinion the main building blocks of domain models. These three values are simply summed up for an overall complexity . In this way we compare the complexity of different representations.

In order to be able to show that our metrics yield useful values despite their simplicity we investigated several use cases. We show the influence of the modeling paradigm and various characteristics of the domain on the complexity of the representation. Finally, we show a method to reduce the complexity of complex, heterogeneous domains.

***Keywords*-Domain modeling, complexity metrics, modeling paradigm, multimodeling**

I. Introduction

Software product lines are a viable methodology to improve engineering of software intensive systems. Northrop et al. [14] suggest a collection of essential practice areas and patterns for product line development and operation. They further highlight the importance of a well structured and documented domain model, since this is the central part of an SPL.

The first step before starting the domain modeling process is the selection of a modeling paradigm. Currently, common choices are Domain Specific Modeling (DSM) and Feature-Oriented Domain Modeling (FODM). The decision for one of these two paradigms is often taken implicitly. In many cases it is possible to represent the domain with either of them. The resulting models will differ in their complexity however.

Reducing complexity is important in several aspects. First of all, it improves the usability of the domain model. This is an important quality attribute, since the model is used for the derivation of products hopefully many times. By improving usability this process can be performed more efficient and with a reduced amount of errors. Due to the long life cycle, and the resulting continuous evolution of the domain model, maintainability is another important aspect. We will show some aspects how the complexity of domain models influences their evolution. Considering this, the reduction of complexity results in an improvement of overall quality.

In a previous work [11] we stated that it is sometimes not advisable to use one domain modeling paradigm to represent the entire domain. In particular, complex domains are often heterogeneous. In order to build an efficient representation of the whole domain it is sometimes necessary to split the domain and build different view-based domain models, which are connected via a multimodeling framework. Again, the reason to do this is to reduce the overall complexity of the domain representation.

In order to be able to evaluate the complexity of a domain model and possibly to improve it, there has to be a metric to estimate a representations' complexity. This is even more important if we are aware of the fact that models can be designed in various ways. Depending on design decisions a model will be more or less complex. With the existence of metrics it is possible to compare and improve domain models.

The main contribution of this work are simple metrics to evaluate the complexity of domain model representations. We intentionally chose simple metrics that can easily be implemented. On several use cases we show the applicability of the metrics.

II. Background

An often used definition by Clements and Northrop [2] describes a software product line as

"'a set of software-intensive systems sharing a common, managed set of features that satisfy 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."'

Basically software product line development follows two fundamentals ideas:

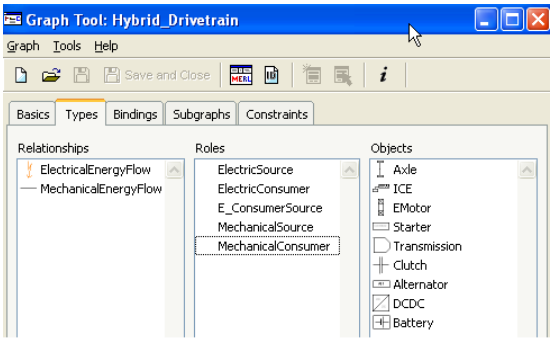


Figure 1. Domain specific language representing the HybConS drivetrain (Case 3) modeled with MetaEdit+

- differentiation of domain and application engineering and
- separation of commonalities and variabilities in domain engineering.

This works focuses on domain engineering and with it on the separation of commonalities and variabilities. An important result of the domain engineering process is the domain model. DSM and FODM are the two modeling paradigms used to create domain models for Software Product Lines. They are described shortly below. To get a better feeling for the two representations, we provide two figures. Figure 1 shows the DSL definition of the HybConS drivetrain domain described later (see Section V-C) and a feature model showing the same domain in Figure 2.

A. *Domain specific modeling*

Domain specific modeling (DSM) aims at the use of a higher level of abstraction and the direct usage of concepts and rules from a specific problem domain. Domain specific languages (DSL) are used to model a system within that domain. The key characteristic of DSLs is their focused expressive power [17]. Because of the narrow focus it is possible to generate products directly from these high level specification [9]. A DSL with a domain specific notation is used to describe a problem. In this way that notation becomes an important factor for modeling productivity [13]. In contrast to a general purpose language, DSLs are used to solve much smaller sets of problems in a specialized, deliberately narrowed area (the domain). Obviously, domain specificity is a matter of degree [13].

We focus on graphical DSLs in this paper because of the experience and results available from our case studies, but the approach can be applied on textual DSLs as well.

B. *Feature-oriented domain modeling*

By feature-oriented domain modeling (FODM) we are mainly talking about feature-oriented domain analysis

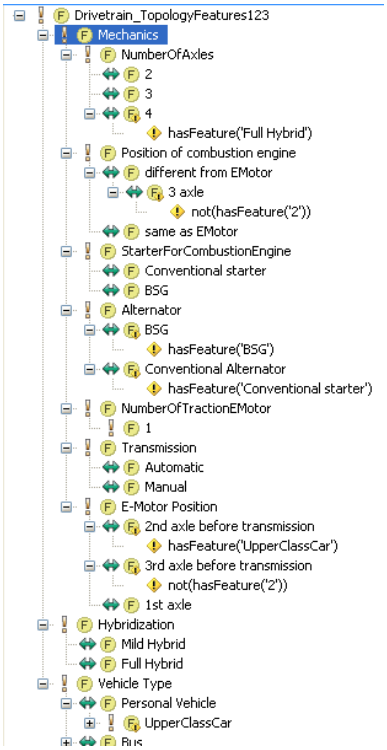


Figure 2. Feature model representing the HybConS drivetrain (Case 3) modeled with pure::variants

(FODA) proposed by Kang et al. [7]. This approach has become the basis of many other feature-oriented approaches (e.g. FORM [8]).

"*Features are the attributes of a system that directly affect end-users. The end-users have to make decisions regarding the availability of features in the system, and they have to understand the meaning of the features in order to use the system.*" [7]

For the representation of features, a tree structured feature model is defined using *consists-of* relations. The relations are either marked as mandatory, alternative or optional. Further composition rules (e.g. "requires" or "mutually exclusive") are used to express relations between features that cannot be expressed in the tree structure itself [7].

C. *Multimodeling*

In a previous work [11] we identified the need for the combination of the two mentioned domain representation paradigms. For more complex domains it is often advisable to split the domain and use different representations for the resulting subdomains. Of course, these different representations have to be connected. This means that there has

to be a possibility to define constraints between different subdomains and different modeling paradigms, respectively. Multimodeling can be used to reduce the complexity of domain representations as will be shown in the remainder of the paper. The main reason for the complexity reduction is the possibility to optimize each of the subdomains. The additional overhead for combining the subdomains is usually much lower as the complexity of the representation with one modeling paradigm.

III. Related work

Rossi et al. [15] introduce complexity metrics for models based on the OPRR metamodel. They use the metrics to evaluate the complexity of different object-oriented analysis and design methods.

Lopez-Herrejon et al. [12] propose to measure the complexity of Software Product Lines with variation point metrics. They state that variation points play a crucial role for the feature configuration at product derivation time. This is why they should be used for qualitative and quantitative metrics. In this specific work the focus is on cyclomatic complexity, which is a typical metric to indicate software quality. This metric has been reformulated to consider variation points. Contrary to our work this metric only focuses on feature representations.

Another kind of metrics has been proposed in [5]. There, the focus is on variability in UML artifacts. The result is the complexity of the product configuration. The focus here is on solution space, whereas in our approach the focus is on problem space.

Sprinkle [16] proposes an algorithm to analyze the complexity of domain specific languages. His approach is based on state models. In order to get insights into the complexity of the modeling language, a state model is generated and used to produce at least one instance of every model in the metamodel. The goal is to show how difficult it is to instantiate models from a metamodel. This approach only focuses on domain specific languages, not on feature-oriented modeling. Moreover it is not a simple, easy to use approach as the metric proposed in our work.

A more detailed investigation of structural metrics for the assessment of maintainability of feature models has been performed in [1]. Another work providing different kinds of metrics for product line architectures has been published in [18].

IV. Analyzing the complexity of domain representations

In domain engineering we can distinguish between problem space and solution space [3]. The problem space is an abstract representation of the domain, independent of the technical realization. In the solution space different technical realizations are described. In the following we are focusing on the problem space.

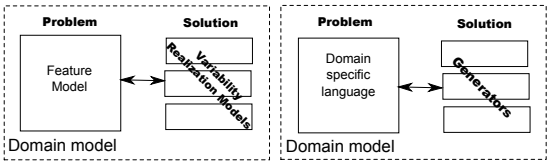


Figure 3. Conceptual differences between the two modeling approaches: feature-oriented and domain specific modeling

The main building blocks of both paradigms are elements, and different kinds of relations used to connect these elements. In a feature-oriented representation the structure is given by the use of features and a defined set of relations between those features. For a tool implementation for DSM, restrictions on the structure have to be given in some form or another. But these restrictions always have to tell which kinds of elements and relations are valid.

Figure 3 illustrates the conceptual similarities of FODM and DSM on an abstract level.

As mentioned before, elements and relations are the main building blocks. We extend this set with properties, which can be used to specify elements and relations in more detail. Based on this, we define 3 types of complexity metrics: *interface complexity, element complexity* and *property complexity*. For each type we developed a metric for DSM as well as for FODM. Since the concrete concepts of the two paradigms are different, the metrics differ as well. In the end, each metric results in a complexity value. Because they are based on the same concepts (interface, elements, property) the resulting values are comparable.

To get the overall complexity the three values are summed up and result in the domain representation complexity.

With these values it is possible to compare the complexity of different domain representations. A prerequisite of course is the existence of at least one domain model. In fact, for most domains one domain model seems to be sufficient, since estimations for the other modeling paradigm can be derived from this domain model. The complexity formulas for both, DSM and FODA, are described in more detail below.

A. Interface complexity

The interface complexity is the most complex metric. It can be used to evaluate the complexity of relationships in the domain model. Relations are used to describe how different elements can be combined.

Equation 1 compares the formula for DSLs (left) and feature models (right). For DSLs the number of relations (n_{RT}) and constraints in the domain model are counted and summed up. Constraints (in both cases) are all kinds of restrictions defined between two elements. This includes, for example, *mutual exclusion* and *mutual requires* as defined in [7]. The n_{RT} in a Feature model is made up of *alternative*

(*"one of many"*) and *or* (*"at least one of many"*) variation points (VP). For a DSL only the different types of relations have to be counted. In the special case of MetaEdit+ it is also necessary to define *Roles* for different *Objects*. In this case n_{RT} consists of $n_{Relationships} + n_{Roles}$.

$$C_{if} = n_{RT} + n_{constraint} \Longleftrightarrow VP_{alt} + VP_{or} + n_{constraint} \quad (1)$$

B. Element complexity

The element complexity indicates the number of variable elements. In the case of a DSL (left) these are all elements which are not fixed. In the case of a feature model (right) these are all optional elements. The two formulas are summarized in Equation 2.

$$C_{elem} = n_{elem} \Longleftrightarrow VP_{Opt} \quad (2)$$

C. Property complexity

Elements in both paradigms might have properties. If they are not fixed, they can be used to adapt the resulting application. To estimate the complexity we count the not fixed properties. The formula is the same for both modeling paradigms as can be seen from Equation 3.

$$C_{prop} = n_{prop} \Longleftrightarrow n_{prop} \quad (3)$$

D. Overall complexity

In order to get the overall complexity the three complexity values are summed up as shown in Equation 4.

$$C_{overall} = C_{if} + C_{elem} + C_{prop} \quad (4)$$

V. Use cases

In order to show that the metrics result in useful values we analyze different domain models. First, we describe the different domains and than we analyze their complexity.

A. Case 1: Fish farm automation

The main functional requirements for a fish farm automation system are feeding and oxygen supervision including an alarm system. In addition, a water level supervision, pH-value measurement and standard functions like switches and lights have to be realized.

One main aim of the automation system is to make the work of the fish farm owner easier and to save resources.

An important characteristic of this domain is the existence of several instances of elements. For example, a fish farm consists of a certain number of ponds. The elements may be assembled in several ways. In fact, the focus is much more on the assembly of elements than on supported functionality. There may be several ponds where each pond may or may not have different supervision and feeding systems. Normally, no two fish farms look exactly the same.

B. Case 2: Configuration of an ERP system

The aim of this project [10] was the systematic reuse of configuration knowledge for Enterprise Resource Planning (ERP) systems of a group of companies. Each of these companies acts in the same industry, they share similar customers and they have similar business processes. The regarded ERP system has to be configured according to the business process of the specific company. Due to the complexity of this configuration process, systematic reuse of configuration knowledge helps to reduce efforts in terms of money and time, and to improve the quality of the ERP instances. Business processes form the SPL architecture. The overall architecture is relatively stable since there is a high potential for reuse across the different companies' business processes. Furthermore, the structure of the business process itself is very stable. There is a fixed scheduling of process steps. There is no need for component instantiation. Functions are accessed in a service-oriented way.

For this case study we started to model the procurement process for prototype processes. In the next step we evolved the domain model to support series part development. In the third step, the domain model is extended in order to support the relocation of customer orders.

C. Case 3: Control unit for an HEV

The target in this project[1] is the development of a generic architecture for an hybrid electrical vehicle (HEV) [6] control unit. Since this is an embedded system, there are many connections to the environment (i.e. the overall system). This makes the domain very complex.

It is not only necessary to describe the software capabilities of the product family, but also the corresponding drivetrain topology since it has a big influence on what is possible in the software. Although there are several other subdomains (e.g. safety), we are focusing on the description of the software and the drivetrain here. The drivetrain consists of an internal combustion engine, one or more electric motors, one of various types of transmissions and starters and so on. All these parts may be assembled in various ways depending on the concrete drivetrain topology. The software consists of several functions which are possible or not, depending on the current drivetrain settings. As already proposed in [11] we splitted the domain in subdomains (drivetrain subdomain and software functionality subdomain). For the evaluation of the domain we developed a drivetrain topology DSL and a feature model representing different topologies. We started with two very similar topologies which are representable with both paradigms. In the following we have a third and fourth topology added to the domain model. In this way we simulate a typical evolution of the domain model and show the gradient of the complexity.

[1] http://www.iti.tugraz.at/hybcons

D. Case 4: PLC controlled inventory system

The domain in this case is a logistics system which is built of conveyors, rotary tables, cranes and high bay racking components. With the resulting product line it should be possible to generate automation system software, documentation, etc. for various assemblies of these components.

The arrangement of elements is also more or less variable, which results in a high number of variable relations. The number of elements is very low, since there are only the aforementioned components available. However, these elements may be instantiated several times. The selection of different functionality is not required.

VI. Results

A. Different modeling paradigms

For Case 1 (see Section V-A) we were only able to describe the domain with a DSL. It seems to be hardly possible to describe the fish farm automation domain in a feature model. The reason might be that there is no common architecture in the representation of the domain. This means, that fish farms can be completely different. Their only commonality are the parts, but the configuration or assembly of parts varies. For a correct representation it would be necessary to describe any number of ponds, each with a different configuration.

Czarnecki et. al [4] introduced an extension for feature-based modeling called cardinality-based feature modeling. With this extension it is possible to define a cardinality for features. The cardinality indicates the number of possible clones for a feature. Cloning a feature means to copy the feature and all of its subfeatures. The resulting subtrees can be configured individually. Basically, this enables the instantiation of features and in this case enables the description of the fish farm domain in a Feature model. The main disadvantage of this approach is the resulting high complexity. Defining constraints between features is much more complex, since there may exist several instances of the target feature. Another disadvantage is the poor tool support, since there is only a prototype implementation.

Figure 4 compares the complexity of the two modeling paradigms for the ERP configuration domain model. Representing Case 2 (see Section V-B) with a domain specific language results in a high number of elements. The reason is the fact that there are many different process steps. Representing them as individual elements in a language seems to be unreasonable. As can be seen in the comparison the DSL representation is much more complex. This means that there has to be an exorbitant high number of different derived products in order to get the DSL payed-off. For business processes there exist a number of business process modeling notations which are sometimes referred to as DSL. The same is true for SW functions as in Case 3. They can be described with means of model-based development, e.g.

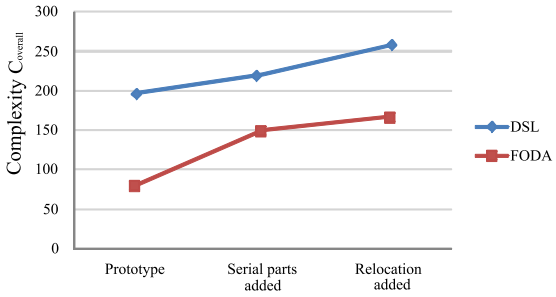


Figure 4. Comparison of the complexity of domain representations for ERP configuration (Case 2) with FODA and DSL during evolution.

Simulink. But, this is not what we call a DSL in the scope of our work, since it is not used to restrict the domain to a single company. Instead these languages can be used to describe a whole bunch of processes or software functions, respectively. In this work we use the DSL to describe a well scoped domain model. This means that the language should be restricted to a defined set of products.

Complexity is very high compared to other domains. A detailed investigation shows that this is mainly caused by element complexity. A high element complexity is an indication that FODM suits better here.

B. Characteristics of different DSLs

The characteristics of the drivetrain subdomain in Case 3 (see Section V-C) and Case 4 seem to be similar to Case 1. Again we have a small set of elements which can be assembled in different ways. In Figure 5 we compare different DSLs. The most important submetrics here are the element and the interface complexity. The number of properties is of course important for the overall complexity, but has hardy any additional information for the analysis of domain model characteristics. The element complexity of the three sample cases is very similar, but there is a big difference in the interface complexity. A detailed investigation of the domain resulted in the assumption that the PISCAS domain is much more detailed and, therefore, has a higher interface complexity. More detailed in this case means that the technical links are described in more detail. This leads to the assumption that a more mature DSL usually has a higher interface complexity.

C. Domain model evolution

Usually, the first version of a domain model represents only few different variants. Over time, the model evolutes and more variants are included. Following this, we started with two topologies and added other topologies one after the other. For this domain model evolution we measured the increase of complexity.

In Figure 6 we again take a look at the HybConS drivetrain subdomain (see Case 3). We show the evolution for

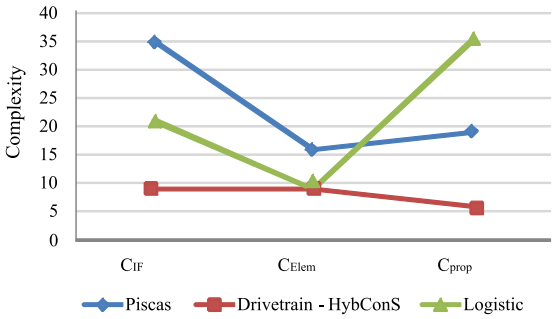


Figure 5. Comparing the complexity of the Piscas DSL (Case 1), the HybConS drivetrain DSL (Case 3) and a Logistics DSL (Case 4)

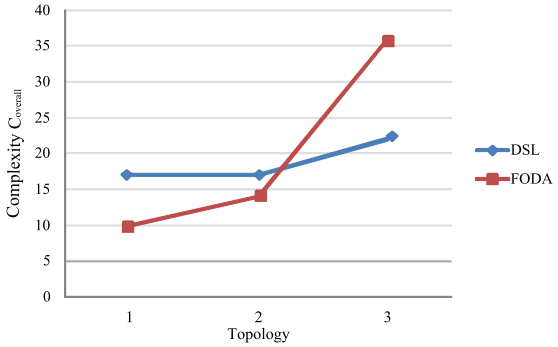


Figure 6. Comparison of the complexity of domain representations for drivetrain topologies for hybrid electrical vehicles with FODA and DSL

DSM and FODM. What we can see is that FODM performs better only for two different topologies. By including the third topology the complexity raises. This means, if we are able to restrict the scope to only a few products it is possible to describe the drivetrain topology with a feature model.

D. Multimodeling - a complexity reduction strategy

Figure 7 shows the HybConS domain model including the drivetrain topology and the software functions. The domain model (containing both subdomains) has been implemented as a DSL, as a feature model and with a multimodeling approach. A DSL in this case always results in high complexity. This is mainly caused by the software functionality, which can not be naturally described in this DSL. For each function there has to be a relation to the mechanical description indicating whether or not the function is possible with this configuration. The feature representation shows an almost linear increase in complexity. After only a few included topologies the complexity of the multimodel representation is lower than the single paradigm representation. One explanation for this behavior is the fact that the representation of the different subdomains can be optimized. The additional overhead for combining the two representations is insignificant.

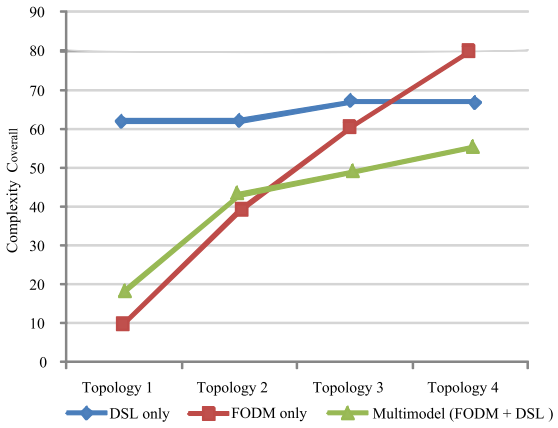


Figure 7. Comparison of HybConS description with FODA, DSL and Multimodeling

VII. Future work and conclusion

This work proposes simple metrics to estimate interface, element and property complexity of domain representations. The complexity of the domain representation can be seen as crucial because the domain model has a long life cycle and is used for the derivation of many products. Summarized, it can be said that, despite the fact that these metrics are simple, they are effective and provide useful results. This has been shown on the example of several case studies.

The findings in our case studies show that the complexity estimation for only one representation is not a valuable base for the selection of a paradigm. Complexities depend on the number of elements in the domain. This may result in a high complexity for both paradigms. Sometimes, this result can be used as an indication that a feature-oriented approach fits better. This is often the case especially for high element complexities.

On the other side, it has been shown that the selection of an appropriate representation also depends on the domain scope. The investigation of domain evolution shows that for a small scope the difference is often not that significant. Often during domain evolution the complexity of one modeling paradigm rises much faster as the other. The case studies show further that different domain representations result in different complexities and that the complexity is also dependent on the characteristics of the domain. In the last step we showed that multimodeling can be used as a strategy to reduce the complexity of heterogeneous domain representations.

The findings of this work can be used to analyze and possibly improve the domain representation.

In future work it could be possible to automate the evalu-

ation and comparison of different domains. If one succeeds in extracting useful patterns it would also be possible to automatically suggest improvements of the current domain model. Another possible future work is a systematic decision making support for the optimal domain modeling paradigm depending on the characteristics of the domain.

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

REFERENCES

[1] E. Bagheri and D. Gasevic, "Assessing the maintainability of software product line feature models using structural metrics," *Software Quality Journal*, vol. 19, pp. 579–612, 2011.

[2] P. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[3] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[4] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *Proc. of the Int. Workshop on Software Factories*, 2005, pp. 16–20.

[5] E. A. de Oliveira Junior, I. M. Gimenes, and J. C. Maldonado, "A Metric Suite to Support Software Product Line Architecture Evaluation," in *XXXIV Conferencia Latinaoamericana de Informatica(CLEI 2008)*, 2008, pp. 489–498.

[6] M. Ehsani, Y. Gao, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design*, 2nd ed. CRC Press, 2010.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[8] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.

[9] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[10] A. Leitner and C. Kreiner, "Managing ERP configuration variants: an experience report," in *Proc. of the 2010 Workshop on Knowledge-Oriented Product Line Engineering*, ser. KOPLE '10. New York, NY, USA: ACM, 2010, pp. 2:1–2:6.

[11] A. Leitner, C. Kreiner, R. Mader, C. Steger, and R. Weiß, "Towards multi-modeling for domain description," in *2nd Int. Workshop on Knowledge-Oriented Product Line Engineering: KOPLE 2011*, ser. SPLC'11, 2011, pp. 1–6.

[12] R. E. Lopez-Herrejon and S. Trujillo, "How complex is my Product Line? The case for Variation Point Metrics," in *VaMoS*, 2008, pp. 97–100.

[13] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, pp. 316–344, December 2005.

[14] L. Northrop and P. Clements, *Software Product Lines: Practices and Patterns*. Addison Wesley, 2002.

[15] M. Rossi and S. Brinkkemper, "Complexity Metrics for Systems Development Methods and Techniques," *Inf. Syst.*, vol. 21, no. 2, pp. 209–227, 1996.

[16] J. Sprinkle, "Analysis of a metamodel to estimate complexity of using a domain-specific language," in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ser. DSM '10. New York, NY, USA: ACM, 2010, pp. 13:1–13:6.

[17] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, 2000.

[18] T. Zhang, L. Deng, J. Wu, Q. Zhou, and C. Ma, "Some metrics for accessing quality of product line architecture," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 2, dec. 2008, pp. 500 –503.

A development methodology for variant-rich automotive software architectures
Eine Enwicklungsmethode für variantenreiche automotive Softwarearchitekturen

Andrea Leitner, Roland Mader, Christian Kreiner, Christian Steger, Reinhold Weiß
Institute for Technical Informatics
Graz University of Technology, Austria
{*andrea.leitner, roland.mader, christian.kreiner, steger, rweiss*}*@TUGraz.at*

Abstract

Ever accelerating product cycles together with multi-discipline engineering processes are typical for safety-critical automotive embedded systems development. This demands for both efficient and effective development and reuse strategies.

A development process following the V-model incorporating model-driven prototyping and development, safety engineering, and verification (unit testing, integration testing, cosimulation, etc.) is commonly found.

Product line engineering enables fast and efficient product configuration through systematic reuse. The V-model has been extended by an integrated product line engineering environment for automotive embedded systems. This ensures the consistent configuration across system architecture description (EAST-ADL2), model driven development (Matlab/Simulink), software component deployment on an ECU network (AUTOSAR based), Simulink based software unit testing, Simulink based software integration testing, and co-simulation model variants. Using the automotive architecture description language EAST-ADL2 enables the integration of safety engineering aspects.

Index Terms

Variability management, Automotive software development, Extended v-model

Abstrakt

Hohe Marktdynamik führt zu immer schneller werdenden Produktentwicklungszyklen automotiver eingebetteter Systeme. Der multidisziplinäre Charakter in der Entwicklung derartiger sicherheitsgerichteter Systeme stellt hohe Anforderungen an eine effiziente und effektive Wiederwendungsstrategie.

Das V-Modell ist ein weitverbreiteter Entwicklungsprozess in dieser Branche. Es beinhaltet typischerweise modellgetriebene Entwicklung, Sicherheitstechnik und Verifikation (Komponententest, Integrationstest, Cosimulation, etc.)

Produktlinienorientierte Entwicklung verspricht schnelle und effiziente Produktentwicklung durch systematische Wiederverwendung und gestattet konsistente Ansteuerung aller Varianten.

In dieser Arbeit wird das V-Modell durch eine Produktlinienumgebung für automotive eingebettete Systeme erweitert. Damit wird die konsistente Konfiguration der Systemarchitekturbeschreibung (EAST-ADL2), der modellgetriebenen Entwicklung (Matlab/Simulink), der Softwarekomponentenverteilung auf den Steuergeräten (AUTOSAR basierend), der Simulink basierenden Komponenten- und Integrationstests und der Cosimulationmodellvarianten sichergestellt.

Durch die Verwendung der Architekturbeschreibungssprache EAST-ADL2 ist es möglich auch sicherheitsrelevante Aspekte zu integrieren.

Index Terms

Variantenmanagement, Automotive Software Entwicklung, Erweitertes V-Model

1. Introduction

Software in the automotive domain is highly complex, multi-functional, distributed, real-time and safety-critical. Further the domain profile is traditionally vertically organized. Mechanical engineers can work more or less independently on their parts. With software this situation changes. Traditionally unrelated parts get related by software and start to interact [1]. Software is today the most crucial innovation driver in modern cars [2].

The basic goal of our work is the support of variant management for the development of a generic embedded automotive software architecture. For a generic architecture it is not useful to support all current and future variants in a certain domain. Thus it is necessary to specify the supported variants. This task is referred to as domain scoping. A too wide domain scope would increase complexity and reduce testability and thus reliability of specific software systems.

In the automotive domain it is not only necessary to ensure consistency, but also to ensure the reliability of the systems. The reliability of each system configuration has to be guaranteed at each time. This affects all tasks

in the software development process. One of the major questions is how one can ensure both, the generic nature of the architecture and the correctness of the specific systems towards a given specification.

One major requirement, therefore, is to handle the domain complexity. One possible mean is the support of different views with a common data base. Due to the complexity of the domain and the variability concerns, appropriate tool support is essential.

To increase the flexibility and support distributed development, the binding of a specific variant should be delayed as long as possible and reasonable.

We give an example that illustrates how a single system development approach can be changed into the systematic implementation of a generic architecture. For the realization of the generic architecture a Software Product Line (SPL) approach will be applied.

One of the major problems in the introduction phase of a SPL is the adaptation of existing processes. Often the technology is available, but there is a lack of process support. We suggest to overcome this problem with a minimal invasive method. This means that the currently used processes and programming habits should be changed as less as possible. Therefore, the current processes and methods have to be analyzed and extended with means of variability management.

Another important point is the reuse of safety artifacts. Safety artifacts are outcomes of safety-related activities in the development process such as safety analyses. The application of these activities is required by the safety standard ISO 26262 that is relevant to the automotive domain.

In Sec. 2.1 the basic concepts of variability and the management of variants are introduced. The consequences for the different phases in the development process are described in Sec. 3. This includes not only architectural aspects, implementation and verification, but also the systematic reuse of safety relevant artifacts.

2. Related work

There is no one-fits-all solution for product lines in general, since solutions are often very domain and project specific. Several works have proposed means and methods to enable variant management in the automotive domain. The architecture description language EAST-ADL2 provides mature safety and variability mechanisms. In the latest version of AUTOSAR, basic variability mechanisms have been integrated as well. Several projects describing integrated tool environments for the automotive domain can be found in current literature. One of them is the EDONA[1] [3] project, which has been launched as a collaboration between French automotive manufacturers and suppliers, research laboratories and software vendors. The aim of the project is an open integration platform for automotive systems development tools.

Requirements are the support of the AUTOSAR specifications and compliance to the safety standard ISO 26262. The project tries to overcome the problem arising from the multitude of different tools in the automotive system development process nowadays. There is no support for variability.

Two further projects have been carried out at the Fraunhofer ISST[2]. The first project was the MOSES [4] (German acronym: Modellbasierte Systementwicklung) project. The aim of this project is the development of a methodology for integrated model based development of E/E systems in vehicles. There is also little support for variability and software product line engineering in this project.

The follow-up project named VEIA[3] [5] (Distributed Development and Integration of Automotive Product Lines) is focused completely on product lines. Goal of the project is the development of a methodology to support variability in the system development process. The major drawback of the VEIA project is the focus solely on the requirements and architecture phase. One advantage of the approach is the support of AUTOSAR.

In [6] a metamodel for the definition of safety assets and a safety process for product lines has been introduced. Further a safety assessment process for product lines has been proposed.

2.1. Variant management with Software Product Lines

In the past software has been mostly static and it was acceptable that changes required much effort. As demands on software changes are getting faster and more dynamic, it is getting more important to delay design decisions to a later point in the development process. Variability in common language use is referred to as the ability or tendency to change. The variability we are interested in does not occur by chance but is brought about on purpose [7].

One approach is the use of Software Product Lines. SPLs are a viable software paradigm for systematic software reuse. The key idea is to build multiple products from a single infrastructure in a way that is aligned to stated business goals [8].

As identified in [9], it makes a difference if the regarded system is an embedded system or not. Variability in embedded systems is much more dependent on hardware than non-embedded systems.

In the following some of the most important concerns related to Software Product Lines are introduced.

1. http://www.edona.fr/
2. http://www.isst.fraunhofer.de/
3. http://veia.isst.fraunhofer.de/

2.1.1. Variability in time vs. Variability in space. Basically, two dimensions of variability can be distinguished: variability in time and variability in space. The space dimension describes different behaviors of products and the time dimension describes the evolution of an artifact over time [10]. In other words, variability in time describes the existence of different versions of an artifact that are valid at different times. This is kind of variability can be handled with version management systems. Variability in space on the other hand describes different shapes of the same artifact at the same time. This is a core issue in software product line engineering [7].

2.1.2. Internal Variability vs. External Variability. External variability is visible to customers, whereas internal variability is hidden. For external variability customers can decide for each feature or functionality whether they need it or not. So the specific needs of the customers are served best. Another possible reason for external variability are e.g. different markets, differences in law or specific standards. To reduce complexity, the customer must not be confronted with technical details. They are encapsulated in internal variability [11].

2.1.3. Binding and binding time. Variability is made explicit in variation points. A variation point can be regarded as a delayed design decision [10]. The binding time is defined as the point in time when the decision upon selection of a variant must be made [12].

In a software engineering process it describes the step where fully or partially instantiated products are created from software artifacts that contain variability [13].

The choice of the binding time has an important influence on the flexibility of the system. If the variation point is bound too early, flexibility of the product line artifacts is lost. On the other hand, late binding is costly and if the point of variant resolution is chosen too late this will unnecessarily increase costs.

There are many classifications for variability binding times in current literature. One classification especially for automotive embedded systems has been introduced by [14]. The authors distinguish 4 different binding times: Programming, Integration, Assembly and Run Time. These times are further divided.

In our project we decided to distinguish between 3 main binding times: Domain definition time, precompile time and parametrization time.

3. Integrating variability in the automotive SW development process

The integration of variability in the automotive SW development process will be proven on the example of the HybConS project. This project is a cooperation of the Virtual Vehicle Competence Center (ViF), AVL List GmbH, and the Institute for Technical Informatics, Graz University of Technology. Overall goal of the project is the development of a generic control software for hybrid vehicles. Therefore, different topologies for hybrid electrical vehicles [15] and different levels of hybridization should be supported by a common software base.

The goal for the variant management project part[4] is to provide a methodology to support variant management and safety considerations over the whole software development process. This means that development artifacts, including safety artifacts, are systematically reused in different products. An overview of the approach is given in Fig. 1. The individual steps are described in more detail in the following.

First we have to define the meaning of the term "*generic architecture*". A generic architecture has a fixed topology and fixed interfaces. In this fixed frame some alternatives and extensions can be plugged in [16]. In our case, generic does not mean to support all possible product variants, but only a set of clearly defined products.

Just imagine there is one new project requirement - "*the architecture should be generic*". What does this mean for the development process and the different development steps respectively? For the requirements, this means that they have to be formulated in a generic way [17], [18].

In the architectural and detailed design phase variation points have to be defined to introduce variability. How this is realized depends on the tools and methods used in the development process under consideration. In the implementation phase components have to be implemented in a way that allows their reuse in several situations. The reusable components are stored in some kind of repository and get reused systematically in predefined situations.

One of the most interesting parts of this approach is the testing part. To guarantee the generic nature of the architecture and the correctness of the software an automated test environment has to be provided. Every time a new component is added to the repository or a component is changed it has to be ensured that all existing or relevant variants are (re-)checked for correctness against the defined specification. With this variant management architecture it can be ensured that the software architecture is still generic and valid after changing a software component.

One important issue is to ensure the consistency of development artifacts throughout the whole development process. With consistency we mean that changes in one artifact have to be propagated to all related artifacts. E.g. changing a requirement means to change implementation, tests and documentation as well. This is even more difficult in a setting with variability. If, for example, a clone-and-own approach is used to develop different variants of a product, a bug identified in the common part of the implementation

4. http://www.iti.tugraz.at/hybcons

has to be fixed in all parts separately. This often leads to subsequent errors. To ensure the consistency we propose a single point of control for variability. In an SPL realization, this is realized through a common domain model. A domain model can be understood as a more abstract representation of the commonalities and variabilities of the planned product range.

3.1. Architecture description with EAST-ADL

EAST-ADL is an Architecture Description Language, which has been initially defined in the ITEA EAST-EEA project [19]. The ATESST2[5] (Advanced Traffic Efficiency and Safety through Software Technology, Phase2) project improves EAST-ADL with systematic approaches for information management, architecting, software product lines, requirements, verification and safety. In further projects it has been aligned with the AUTOSAR[6] (AUTomotive Open System ARchitecture) automotive standard (see Sec. 3.2). The concepts are similar, but there is no real standardized mapping.

EAST-ADL provides means for describing variability and safety considerations. In this project setting the architecture description language was introduced in a lightweight way. In day-to-day business, design and implementation are widely realized in Simulink. It is not reasonable to introduce another engineering tool. Anyway, it is important to keep the engineering and documentation artifacts consistent automatically. The approach we propose is to extract structural informations from AUTOSAR and use this information to automatically construct the EAST-ADL representation. In this way we introduce EAST-ADL in the engineering process, but use it only as a tool for documentation.

For our lightweight approach we first extract the structural informations from the AUTOSAR description. These structural informations are mapped to corresponding concepts in EAST-ADL. With this approach the EAST-ADL representation can be extracted from the AUTOSAR description automatically. The same can be achieved with variability informations. Variability in AUTOSAR is described on the component level. These informations can be used to construct the compositional variability model in EAST-ADL.

3.2. Reusable automotive software components

In a common automotive software development setting, Simulink is used as the main development environment. The integration of Simulink components in the Product Line is supported by a novel variability component model that is aligned with the AUTOSAR component model. A special variant block set [20] helps defining component-internal variation points.

5. http://www.atesst.org
6. http://www.autosar.org/

Another approach to support reusability of components is AUTOSAR. AUTOSAR has a much wider reuse scope than the variant block set. The objective of the AUTOSAR initiative is to establish an open industry standard for the automotive software architecture between suppliers and manufacturers. The main slogan of the consortium is defined as "*Cooperate on standards, compete on implementation*" [21].

One major goal is the separation from software and hardware in order to allow software reuse and smooth evolutions while limiting re-development and validation.

The focus in the presented approach is on the upper level of the AUTOSAR concept, the software component descriptions.

The standardized design for reuse in AUTOSAR can be used in the presented approach. One major drawback is the fact that systematic variant handling is supported only partly. AUTOSAR defines a mechanism to configure infrastructure components and a mechanism to calibrate application level components via interfaces. It does not define a mechanism to reflect variability in application level components i.e. model feature dependent behavior in a application level component. That is why we employ a special component model resolving this deficit.

3.3. Verification and variability

All the artifacts produced in the phases on the left side of the V-model have to be verified in the corresponding phases on the right side. As a result of multi-discipline engineering necessary for automotive systems, verification comes in several flavors. Even when only focusing on testing automotive software, all kinds of configurations like testing models (MiL), generated software (SiL), controllers running this software (HiL) are relevant. In every case, the environment behavior is simulated/emulated in order to generate the test case stimuli and verify the unit under test.

When introducing variability, test cases on all levels, i.e. unit tests, integration test, software system, etc. have to support variability, just like the software components do. Variation points have to be incorporated, that are consistently controlled from the common variability model. This is crucial as both the test and the component under test are assembled for a test run.

To go one step further, verification cannot only be provided for software, but for the whole embedded system. Therefore, new design and verification methodologies that can handle different subsystems in the same environment are necessary.

These subsystems are mainly designed for different purposes to operate in different physical domains such as in mechanical, electrical and thermodynamic domains. They build together a heterogeneous embedded system. Heterogeneity

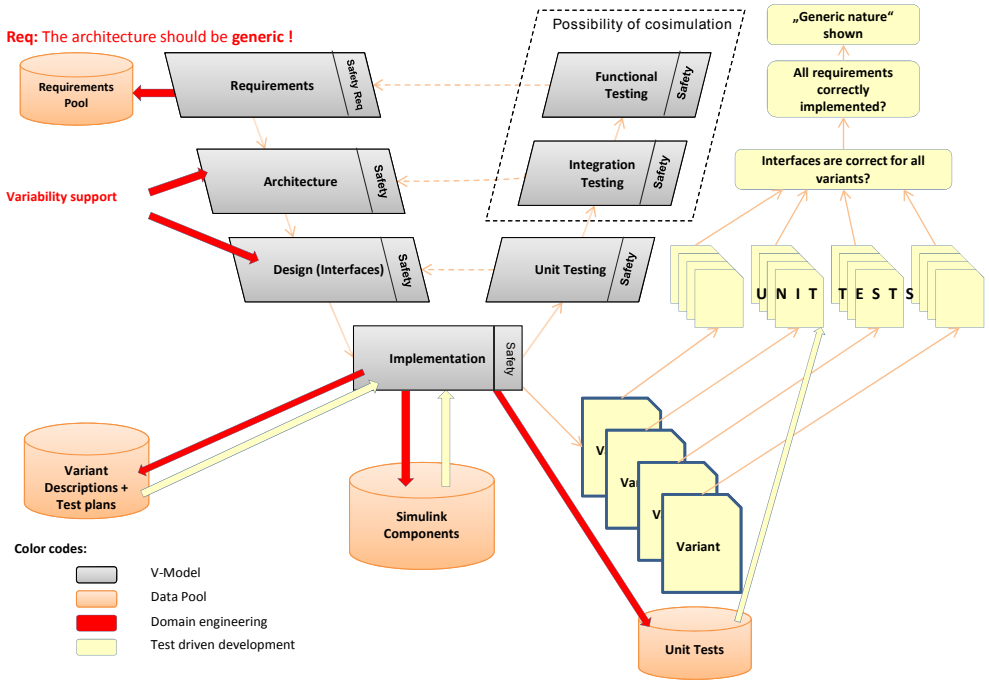


Figure 1. Improving the common V-model to support variability

is treated by means of multi-domain embedded systems described using different description languages.

Cosimulation is one approach for functional verification of heterogeneous system designs. In this solution, several simulators are used to verify subsystems in one common environment. Simulators are connected to each other by specified scheme building a cosimulation network.

The developed application framework [22] allows a schematic-based system design and automatic generation of a cosimulation platform. It is a distributed, webbased application, which allows system design using models provided by a library and automatic generation of verification platforms.

Having test cases - i.e. executable requirements available, it is possible to test several product variants by running all test cases for each variant. Practically, this is only feasible when using test automation. In this way however, an entire spectrum of variants can be verified regularly. These variants are built from a common generic architecture and component pool. The automatic composition of variants makes it possible to verify the generic nature of architecture and components and further ensure the correct functioning of all relevant variants.

In automotive industry practice, unit tests are used when developing software components. In the current state of the HybConS software, each of ≈80 components is associated with 10 executable unit tests on average. A full unit test run takes around 20 minutes on a typical development laptop.

This approach is certainly susceptible to an exponential scaling of test runs. However, in SPLE, only relevant variants are in focus, not all possible variants. In practice, we can expect a few tenths of variants (parameterization does not lead to a new software variant). In this way, a comprehensive re-test of all relevant variants is well within reach during a daily build procedure e.g.

3.4. Safety-related process artifacts

Automotive embedded systems do not only carry out comfort functions and entertainment functions but also highly safety-critical functions that control the behaviors of automotive actuators such as brakes, transmissions, electric motors or engines. It is obvious that a failure of the embedded system that controls these actuators can lead to accidents caused by hazards such as the omission of braking or the providing of positive torque by an electric motor or an engine without intention of the driver.

Due to their criticality these embedded systems are developed according to rigorous development processes such as defined by the automotive safety standard ISO 26262 [23]. This standard imposes special requirements to process steps such as requirements elicitation, design, implementation, verification, validation and documentation. Moreover the application of safety analyses such as preliminary hazard analysis, FTA (Fault Tree Analysis) and FMEA (Failure Modes and Effects Analysis) [24] is required. The application of safety-related process steps produces safety-related process artifacts such as safety requirements, system architecture models, implementation of safety integrity measures, fault injection test results, hazard tables, fault trees and FMEA tables.

It is obvious that these safety-related process artifacts are subject to variability depending on the features, functions, sensors and actuators of the vehicles in the product line as well. A particular vehicle variant clearly leads to a particular set of safety requirements, a particular system architecture, particular safety integrity measures, a particular fault injection test campaign, particular hazards, particular faults and failures.

The adequate integration of these safety-related process artifacts in a V-model that supports variability such as described in Section 3 is a challenging and necessary issue to achieve compliance to the automotive safety standard ISO 26262. Management of safety-related process artifacts in terms of variability is analog to that of other development artifacts illustrated in Figure 1.

4. Conclusion

An integrated product line oriented development environment for automotive embedded systems has been described. It extends the commonly used V-model by means of consistent variability handling. The approach does not only include the management of software artifacts, but also documentation, safety artifacts and various testing artifacts. With this approach we can support the systematic development of a generic software architecture. It was deliberately designed for lightweight and evolutionary introduction in a typical automotive development process and environment.

Acknowledgment: We gratefully thank pure systems GmbH for their support.

References

[1] B. Hardung, T. Kölzow, and A. Krüger, "Reuse of software in distributed embedded automotive systems," in *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*. ACM, 2004, pp. 203–210.

[2] M. Broy, "Challenges in automotive software engineering," in *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 33–42.

[3] F. Ougier and F. Terrier, "Edona: an open integration platform for automotive systems development tools," http://www.edona.org/home/liblocal/docs/ERTS2008_ougier-v5.pdf, visited 2010.

[4] E. Kleinod, "Modellbasierte Systementwicklung in der Automobilindustrie," http://www.isst.fraunhofer.de/Images/isst-bericht_77-06_online_tcm81-17204.pdf, visited 2010.

[5] M. Groe-Rhode, S. Euringer, E. Kleinod, and S. Mann, "Rough Draft of VEIA Reference Process," http://veia.isst.fraunhofer.de/medien/downloads/isst-bericht_80-07_online_en.pdf, visited 2010.

[6] I. M. Habli, "Model-based assurance of safety-critical product lines," Ph.D. dissertation, University of York, Department of Computer Science, 2009.

[7] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, 1st ed. Springer, 2005.

[8] C. W. Krueger, "Introduction to Software Product Lines," http://www.softwareproductlines.com/introduction/introduction.html, 2008.

[9] M. Jaring and J. Bosch, "A Taxonomy and Hierarchy of Variability Dependencies in Software Product Family Engineering," *Computer Software and Applications Conference, Annual International*, vol. 1, pp. 356–361, 2004.

[10] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. H. Obbink, and K. Pohl, "Variability Issues in Software Product Lines," in *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. London, UK: Springer-Verlag, 2002, pp. 13–21.

[11] G. Halmans and K. Pohl, "Communicating the variability of a software-product family to customers," *Inform., Forsch. Entwickl.*, vol. 18, no. 3-4, pp. 113–131, 2004.

[12] F. J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Berlin: Springer, 2007.

[13] C. W. Krueger, "Towards a Taxonomy for Software Product Lines," in *PFE*, 2003, pp. 323–331.

[14] C. Fritsch, A. Lehn, D. T. Strohm, and R. B. Gmbh, "Evaluating Variability Implementation Mechanisms," in *Proceedings of International Workshop on Product Line Engineering*, 2002, pp. 59–64.

[15] M. Ehsani, Y. Gao, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design*, 2nd ed. CRC Press, 2010.

[16] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[17] Heinänen, Jukka and Tukiainen, Markku, "Improving Requirement Reuse: Case Abloy," in *EuroSPI 2006 Industrial Proceedings*, 2006, pp. 3.21–3.30.

[18] D. Beuche, A. Birk, H. Dreier, A. Fleischmann, H. Galle, G. Heller, D. Janzen, I. John, R. Kolagari, T. von der Massen, and A. Wolfram, "Using requirements management tools in software product line engineering: The state of the practice," in *Software Product Line Conference, 2007. SPLC 2007. 11th International*, 2007, pp. 84 –96.

[19] "Embedded Electronic Vehicle Architecture," http://www.itea2.org/public/project_leaflets/EAST-EEA_results_oct-04.pdf, 2004.

[20] D. Beuche and J. Weiland, "Managing flexibility: Modeling binding-times in simulink," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, R. Paige, A. Hartman, and A. Rensink, Eds. Springer Berlin / Heidelberg, 2009, vol. 5562, pp. 289–300.

[21] H. Dörr, "The AUTOSAR Way of Model-Based Engineering of Automotive Systems," in *ICGT '08: Proceedings of the 4th international conference on Graph Transformations*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 38–38.

[22] S. Kajtazovic, C. Steger, A. Schuhai, and M. Pistauer, "Automatic generation of a verification platform for heterogeneous system designs," in *Proceedings of the Forum on Specification and Design Languages FDL'05*, 2005, pp. 153–164.

[23] International Organization for Standardization, "ISO/DIS 26262 Road vehicles - Functional safety," 2009.

[24] Nancy G. Leveson, *Safeware: system safety and computers*. Addison-Wesley Publishing Company, 1995.

Improving Domain Representation with Multi-Paradigm Modeling

Andrea Leitner
Inst. f. Tech. Informatics
Graz University of Techn., AT
andrea.leitner@tugraz.at

Reinhold Weiß
Inst. f. Tech. Informatics
Graz University of Techn., AT
rweiss@tugraz.at

Wolfgang Ebner
Virtual Vehicle Competence Center
Wolfgang.Ebner@v2c2.at

Christian Kreiner
Inst. f. Tech. Informatics
Graz University of Techn., AT
christian.kreiner@tugraz.at

ABSTRACT

Domain modeling is a key task in Software Product Line (SPL) development. We identified two popular modeling paradigms: Feature-Oriented Domain Modeling (FODM) and Domain-Specific Modeling (DSM). The representation of the domain model is crucial in SPL engineering, since domain models have a long lifecycle and represent the externalized organizational domain knowledge. For complex and heterogeneous domains, such as embedded systems, different representation techniques can be useful to describe different aspects of the system.

This paper describes a multi-paradigm modeling approach which enables the combined representation of Feature models and Domain-Specific Languages (DSL). The main idea is to reduce the complexity of the model and, thus, to improve its usability and maintainability. The technical realization of the multi-paradigm modeling approach uses 3 types of constraints to connect different modeling paradigms. The constraint checking mechanism reuses existing technology in order to not re-invent the wheel.

A case study describes the applicability of the approach in a real-life automotive project for hybrid electric vehicle control software (HybConS) and shows the improvement of this approach compared to single-paradigm modeling.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Domain engineering

General Terms

DESIGN , LANGUAGES

Keywords

Software product lines, Multi-paradigm modeling, Feature-oriented domain modeling, Domain-specific modeling

SPLC - Vol. II September 02 - 07 2012, Salvador, Brazil

1. MOTIVATION

The motivation for this work and the example which will be used throughout this paper is an automotive project called HybConS[1]. The goal of this project is to develop a generic software architecture for hybrid electric vehicle control units. Hybrid electric vehicles may vary in different drivetrain configurations, different mechanical components and different supported markets, which all influence the control software. An SPL approach is used to support all these variants in a systematic way. All aforementioned aspects should be part of the resulting problem description.

In this work, we focus on the problem space description. The problem space contents are dependent on the involved stakeholders. Various system factors affect the software and variability of embedded systems. This means that different stakeholder views need to be integrated in the resulting problem description. If the various domain aspects have different characteristics, we are talking of a heterogeneous domain. A heterogeneous domain is characterized by various dependent stakeholder views without one-to-one mapping. In the automotive domain, there is no direct correspondence between mechanical parts and software functionality resulting in two completely distinct architectures (mechanics and software). In contrast, for automation system software there is often a mapping to specific hardware elements (i.e. if a sensor is part of a product the corresponding software needs to be part as well). This domain is not considered heterogeneous.

Improved problem domain representations are important for various reasons. Most importantly, it is a collection of organizational domain knowledge and therefore a core asset for any organization. Furthermore, a well-defined problem space representation facilitates subsequent tasks (e.g. product derivation).

The main contribution of this work is a combined and improved representation of the problem domain by using the advantages of existing modeling paradigms and tools. This work is structured as follows: Section 2 summarizes the most important related work and Section 3 gives some background information. The following sections will highlight the multi-paradigm modeling framework from different perspectives. The concept perspective in Section 4 introduces the principle design of the multi-modeling framework.

[1]http://www.iti.tugraz.at/hybcons

Section 5 describes the details of the implementation perspective. Section 6 covers the process perspective on the multi-modeling framework. In Section 7 the practical perspective is shown with a concrete example. Section 8 finally concludes the paper.

2. RELATED WORK

Kelly et al. [10] highlight the importance of the appropriate representation in DSML specifications. It is stated that "*the correct representational paradigm depends on the audience, the data's structure, and how users will work with the data*". We extend this statement to the next level and argue that not only the representation is important, but also the paradigm to provide this representation in an effective way.

The need for heterogeneous representation mechanisms has previously been identified by different authors. Elsner et al. [3] propose a framework to formulate constraints between different modeling artifacts. This framework seems to be rather complex, because it is based on complex constraint definition languages. Bak et al. [1] on the other side suggest a completely new language, which combines the two approaches. The main contribution is the integration of capabilities of both modeling approaches, while providing first-class support for feature modeling. This is an improvement over previous works, where extensions to the original feature-oriented modeling approach have been suggested (e.g. cardinality-based feature modeling [2]). The goal of these extensions is to emulate domain-specific or class modeling properties in feature-oriented representations. In contrast to this new language, we suggest to use existing modeling techniques that have been proven useful over the years.

Voelter et al. [21] describe the limitations of feature-oriented domain modeling. They propose as a solution to combine the feature-oriented approach with domain-specific languages wherever the descriptive power of features is not enough.

Heidenreich et al. [6] propose a solution to map features to any Ecore-based representation. In contrast to the current approach, the mapping spans from the problem space to the solution space and is not meant to realize different representations in the problem space.

Haugen et al. [5] describe a separated language approach to specify variability in DSL models. They propose a Common Variability Language (CVL), and variability resolution mechanisms embedded in the OMG meta-model stack. This allows to describe variability in potentially all MOF-based languages, including UML, as well as MOF- and UML profile based DSLs. While being a general and clean approach to handle variability, it does not seem directly applicable to feature abstraction hierarchies and their complex constraints.

3. BACKGROUND

Basically, a domain model defines the functions, objects, data, and relationships in a domain [7].

Traditionally, domains are described either by a domain-specific language (DSL) or by a feature-oriented approach (e.g. FODA [7], FORM [8]). Of course there are other possible approaches (e.g. [4]), which seem to be point solutions with little practical relevance. In this work we concentrate on feature models and graphical DSLs and their advantages for the representation of a domain model. The two paradigms are described in more detail below:

3.1 Feature-oriented modeling

A feature can be defined as "*a prominent and distinctive user visible characteristic of a system*" [12]. The big advantage of this kind of abstraction is that it can be understood by both, customers and developers [9].

Kang et al. [7] first proposed to use features to represent the problem domain with the concept of *Feature-Oriented Domain Analysis* (FODA). A feature model consists of a hierarchical representation called feature diagram and composition rules, such as mutual exclusion (excludes) and mutual dependency (requires). Commonality can be described in terms of *mandatory* features and variability in terms of *optional* or *variant* features [18].

Over the years several extensions to this original approach have been introduced. Basically, they are used to simulate the modeling capabilities of domain-specific modeling. Cardinality-based feature modeling [2] e.g. has been introduced to support instantiation of features. Using this extension, a feature can be annotated with cardinality, which indicates the number of possible clones of this feature in a product. It is also possible to combine features into feature groups and define cardinalities for entire groups. Additionally, attributes with configurable values can be added to a feature. To ensure tool independence, such extensions are left out from our investigation.

3.2 Domain-specific modeling

The aim of domain-specific modeling (DSM) is to use a higher level of abstraction and the direct usage of concepts and rules from a specific problem domain. Domain-specific languages (DSL) are used to model a system within that domain. The key characteristic of DSLs is their focused expressive power [20]. Because of the narrow focus it is possible to generate products directly from these high level specifications [11]. A DSL with domain-specific notation is used to describe the problem. This notation is an important factor in the improvement of productivity [15]. In contrast to general-purpose languages, DSLs are used to solve a much smaller set of problems in a specialized, deliberately narrowed area (the domain).

The higher expressiveness can be used to represent the scope of a product family in a more natural way by using the notation of the specific domain. The analysis of the domain results in a DSL, that can be used to describe this domain [19].

3.3 Combined domain representation

For the preparation of a combined representation it is first important to be aware of the structural commonalities of FODM and DSM. Figure 1 shows that both approaches share the same structure from an abstract point of view. They only differ in the concrete representation of the domain.

The basic building blocks of both approaches are elements, different kinds of connections between these elements, and element properties. The structure of a feature-oriented representation is given by the use of features and a defined set of relations (connections) between these features. For a DSM tool implementation, restrictions on the structure have to be given in some form or another (e.g. GOPPRR[2]). These

[2]http://www.metacase.com/support/45/manuals/mwb/Mw-

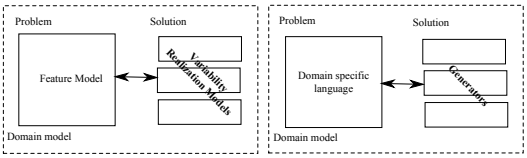


Figure 1: Conceptual differences between the two modeling approaches: feature-oriented and domain-specific modeling

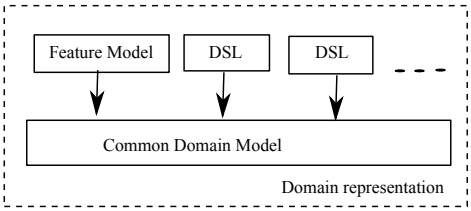


Figure 2: Conceptual realization of the multi-paradigm modeling framework

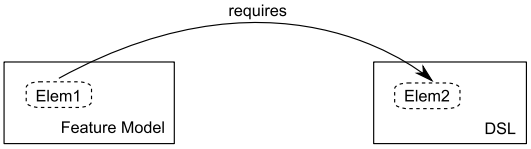


Figure 4: Illustrating the *hasElement* constraint

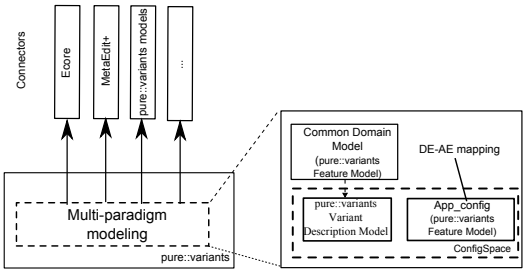


Figure 5: Overview of the technical realization of the multi-paradigm modeling framework

restrictions define which kinds of elements and connections are valid. Properties for a detailed specification of elements are usually available in both approaches.

4. MULTI-PARADIGM MODELING FRAMEWORK ARCHITECTURE

This work describes the detailed concept of our multi-paradigm modeling approach for problem descriptions in SPLs based on our previous work [13]. The details of the domain and application engineering process in the context of multi-paradigm modeling are described below.

4.1 Domain engineering

The major problem for the implementation of a combined solution is the definition of constraints between elements from different models as illustrated in Figure 4. This means that there has to be some kind of model interface allowing to describe these dependencies.

Figure 3 shows a meta-model defining the main building parts of the multi-paradigm modeling framework. Figure 2 illustrates the basic concept of the multi-paradigm modeling framework. The core part is a *Common Domain Model* (CDM), which is used to represent inter-model (different domain aspects or different representations) constraints. Intra-model constraints are described in the specific tool or language. This enables an efficient representation of the different domain aspects. The basic design concept is the use of element references in the CDM. The advantage of this design is the avoidance of transformations following a common meta-model which would probably result in a loss of information. We identified 3 basic inter-model constraints, based on the main building parts. This makes them applicable with different modeling languages and paradigms:

- **hasElement**
  The "*hasElement*" constraint is used to check whether or not an element exists in a model. This means that an element depends on the existence of an element in another model.

- **hasAttributeValue**
  The "*hasAttributeValue*" constraint defines the dependency of an element on a specific value of an attribute in another model.

- **hasConnection**
  The "*hasConnection*" constraint is used to check whether or not a connection between two elements exists. It can be used to make an element dependent on this connection.

These three types of constraints must be represented in the CDM and have to be evaluated by the constraint checking mechanism. They are intentionally kept simple, which has to be kept in mind during domain design.

4.2 Application engineering

A constraint checking mechanism should ensure consistency in the application engineering process. The mechanism must be able to check the existence of elements, connections and properties in various models.

5. MULTI-PARADIGM MODELING FRAMEWORK IMPLEMENTATION

Various tools could be used for the realization of the CDM, because it only needs to represent elements (references to original model element), connections (inter-model constraints), and properties. Our multi-paradigm modeling framework prototype is built on pure::variants[3], which provides a constraint checking engine. The existence of this constraint checking engine was the main reason to choose this tool environment. Furthermore, feature models provide an easy way to represent the required information.

1_1_1.html

[3]http://www.pure-systems.com/pure_variants.49.0.html

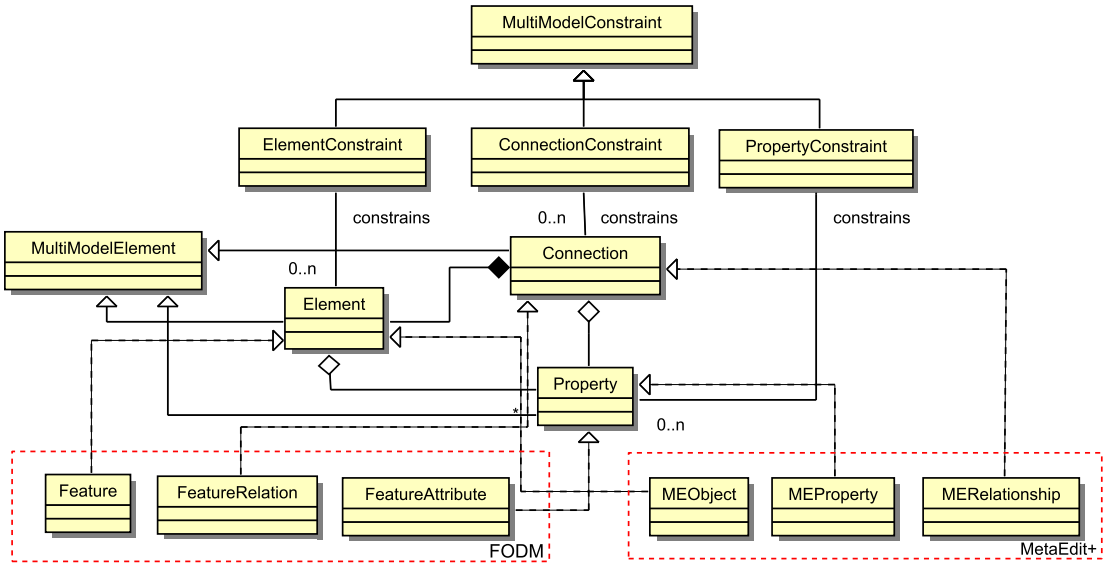


Figure 3: Metamodel for the multi-paradigm modeling concept and their relations to FODM and MetaEdit+

Table 1: Mapping between CDM concepts and different technical representations

CDM (Fig. 3)	**pure::variants** FODM	**MetaEdit+** DSL
Element Connection Property	Element (ps:feature) Element (ps:connection) Property	Object Relationship Property
ConfigSpace (Fig. 5)	Variant Description Model	Graph

5.1 The Common Domain Model

Internally, a pure::variants Feature model is used to represent the CDM. Elements from the different models are represented as features. The required information are: the element name in the source model, and the path to the source model. The latter is stored as a pure::variants *Property*.

5.2 Inter-model communication

Generally, there needs to be an interface to communicate with other models. For our prototype we implemented connectors for Ecore[4] and MetaEdit+[5]. Basically, these connectors are parsers which are able to read the respective domain and application models. The mapping of different concepts is described in Table 1.

A special case of tool connection is required by the MetaEdit+ tool. For correct communication, the meta-model has to be exported as a MetaEdit+ XML (.mxt) file. Examining the application model can then be done via a SOAP interface.

For Ecore, both models should be available as a file.

[4]http://www.eclipse.org/modeling/emf/?project=emf
[5]http://www.metacase.com/

5.3 The multi-paradigm modeling plugin

The multi-paradigm modeling framework is implemented as an Eclipse plugin, and consists of a UI and a logic part. The UI enables the user to define and check constraints. Required inputs are the current CDM, the element to define the restriction on (and the corresponding model), and the target element, property value or connection (and the corresponding model), respectively.

The plugin uses the tool connectors described in Section 5.2. Once the path to a domain model is given, possible values are retrieved and presented to the user. After finishing the input form the CDM gets updated. Elements are simply represented as pure::variants *Feature*. Properties are added as pure::variants *Property* to the respective element. Connections are features with type *ps:connection* and two properties: a connection source and a connection target. A pure::variants *restriction* corresponding to the constraint is added to the target element.

5.4 Application engineering

In application engineering, concrete products are derived using the various tools independently. Consistency between them is ensured by checking inter-model constraints via the CDM. Using a pure::variants Feature model to represent the CDM enables the use of the tools internal Prolog engine for constraint checking and solving. The right-hand side of Figure 5 illustrates the two important parts of the application engineering phase in the context of multi-paradigm modeling. They are described in detail below.

5.4.1 Configuration space

The first step in application engineering is the creation of a pure::variants *Configuration Space* [16] containing only the CDM. The *Configuration Space* is used to combine models for configuration purposes. This step results in a

pure::variants *Variant Description Model* (VDM), which is normally used to define a concrete product by feature selection. In the case of multi-paradigm modeling it is processed automatically by the constraint checker described below.

5.4.2 Configuration of multi-paradigm modeling application project

A central part for the configuration of a concrete application is the *app_config* file. It stores the mapping between application models and their respective domain models. This is straightforward for pure::variants and Ecore-based domain models. Pure::variants Feature models (*.xfm) are mapped to Variant Description Models (*.vdm), and Ecore models (*.ecore) are mapped to any type of Ecore application model. For MetaEdit+, the domain model is mapped to a concrete graph within a MetaEdit+ project. With this information, access via the SOAP interface can be established.

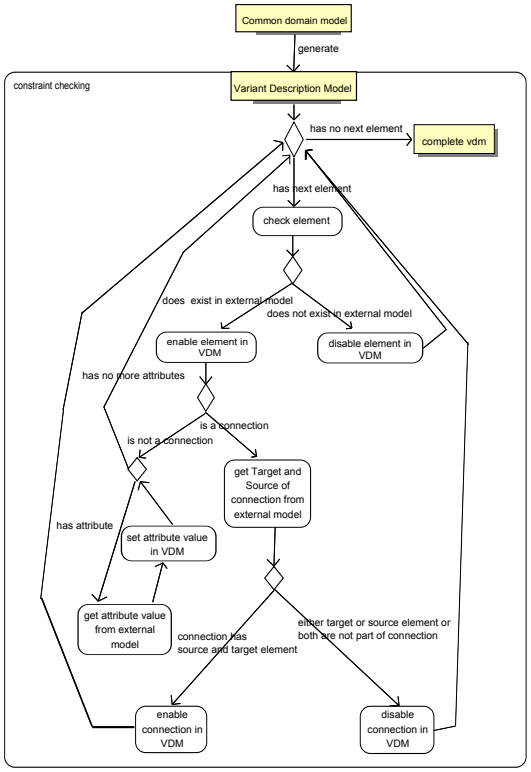


Figure 6: Constraint checking mechanism to ensure consistency in application engineering

5.4.3 Constraint checking process

Figure 6 shows the basic principle of the multi-paradigm modeling constraint checking mechanism. This mechanism is performed automatically. The constraint checking process iterates over each element in the VDM in the multi-paradigm modeling *configuration space* mentioned before (see Figure 5). This model is derived from the CDM and contains the path to the connected origin domain models. This information is used to retrieve the corresponding application model from the *app_config* file. Now, all the information required to perform the actual checking is available.

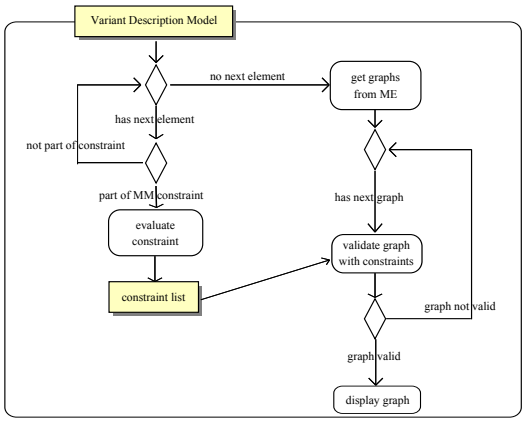


Figure 7: Process of getting all valid MetaEdit+ (ME) Graphs for a given Variant Description Model in pure::variants

For elements of type *ps:feature* it has to be checked if this element exists in the external application model. If it does, the selected state of the element in the VDM is set to true, otherwise to false. If the element (in the VDM) has an attribute, the value of this attribute has to be looked up in the corresponding external application model. The value is then set in the VDM.

For elements of type *ps:connection* it is checked if a connection of this type exists with the connection source and target denoted in the CDM. If such a connection exists, the selected state of the element in the CDM is set to true and to false otherwise. Once all the elements in the VDM are evaluated, there is a concrete product description in pure::variants. The pure::variants constraint checker finally checks the resulting model for validity.

5.5 Additional functionality

Two additional functions improve the usability of the multi-paradigm modeling approach. They are available for pure::variants and MetaEdit+ so far, but are not restricted to these tools. The implementation serves as a base for the description of the concepts here. The major advantage of these add-ons is the fact that one can start with either a graphical domain representation or a feature selection. Based on this first selection, the remaining possibilities of the other representations are calculated automatically. *"AutoSelection"* starts with the creation of a MetaEdit+ graph. The calculation of valid graphs provides the possibility to start with a feature selection.

5.5.1 AutoSelection

The *AutoSelection* mechanism takes a MetaEdit+ Graph as input. Based on the constraints defined in the CDM, valid feature selections for this Graph are automatically selected in the Feature model. Therefore, we first need to analyze the

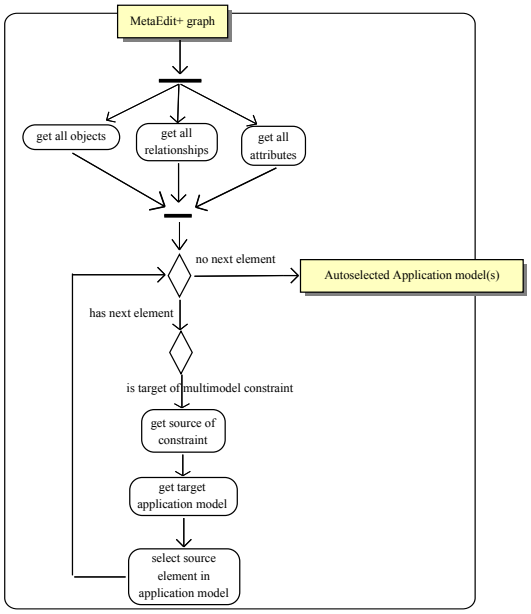


Figure 8: Process of automatically generating a reconfigured Variant Description Model (pure::variants) based on a given MetaEdit+ (ME) Graph

MetaEdit+ Graph. In the next step, we extract parts of the graph which are part of the CDM (this means they are part of an inter-model constraint). If such a constraint exists, it gets evaluated and the selection of the corresponding element in the particular model is done. The major advantage of this function is the possibility to start with the description of a graphical MetaEdit+ representation (e.g. the drivetrain configuration in our case), and do a first preselection of the remaining possibilities in the Feature model (the software functions in our case). This ensures consistency right from the beginning.

5.5.2 Calculate valid graphs

The second functionality based on multi-paradigm modeling can be used to determine all MetaEdit+ Graphs from an existing set of Graphs that are valid for a particular feature selection. Here, we want to start with the selection of features, the software functions in our example. Based on the constraints in the CDM, it is now possible to get all (remaining) valid graphs in MetaEdit+. In this case we would get a collection of valid drivetrain configurations (application models) for the currently selected software functions. In the prototype this is implemented as follows: First, the feature selection gets parsed, and the parts relevant for the multi-paradigm modeling context are extracted (selected features that are part of a constraint in the CDM). The result is a list of constraints that have to hold true in a graph in order to be considered as valid. All graphs from the current project are then imported from MetaEdit+ and checked against these constraints. Graphs for which all constraints hold true are displayed as the result set.

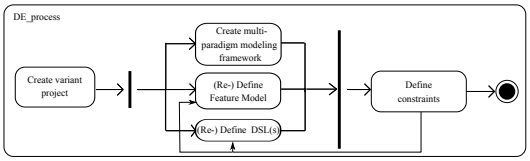


Figure 9: Multi-paradigm modeling activities in the domain engineering process

Figure 10: Multi-paradigm modeling activities diagram in the application engineering process

6. DOMAIN MULTI-PARADIGM MODELING PROCESS VIEW

The multi-paradigm modeling process follows the engineering processes defined for software product line engineering. In domain engineering, constraints between different representations have to be defined, and in application engineering the validity of the application models has to be ensured by checking the constraints.

6.1 Domain engineering activities

Figure 9 shows the basic tasks in domain engineering for multi-paradigm modeling. The first step is to create a pure::variants *variant project*. In the next step, the multi-modeling framework and the different domain representations are created. After these first steps the constraints can be defined.

6.2 Application engineering activities

Figure 10 illustrates the required steps to prepare the multi-paradigm modeling framework for the derivation of a concrete product. First, the application models are derived from the domain models in their respective environments (as in single-paradigm modeling). Next, each application model has to be linked to its corresponding domain model. Finally, a pure::variants *Configuration Space* containing only the CDM is created. After these preparation steps the constraints are checked as described in Section 5.4.3.

7. CASE STUDY

A practical case shows the applicability of our approach. We applied the multi-paradigm modeling framework within the scope of the HybConS project, described in Section 1. The domain has been divided into two domain aspects: control software and drivetrain configuration (system under control). Figure 11 shows a sample multi-paradigm model connecting the *MildHybrid* Feature to a *MechanicalEnergyFlow* connection in a drivetrain DSL. A direct mechanical connection (without clutch) excludes full hybrid functionality. This a very central concept, because many software aspects depend on the distinction between full and mild hybrid configuration.

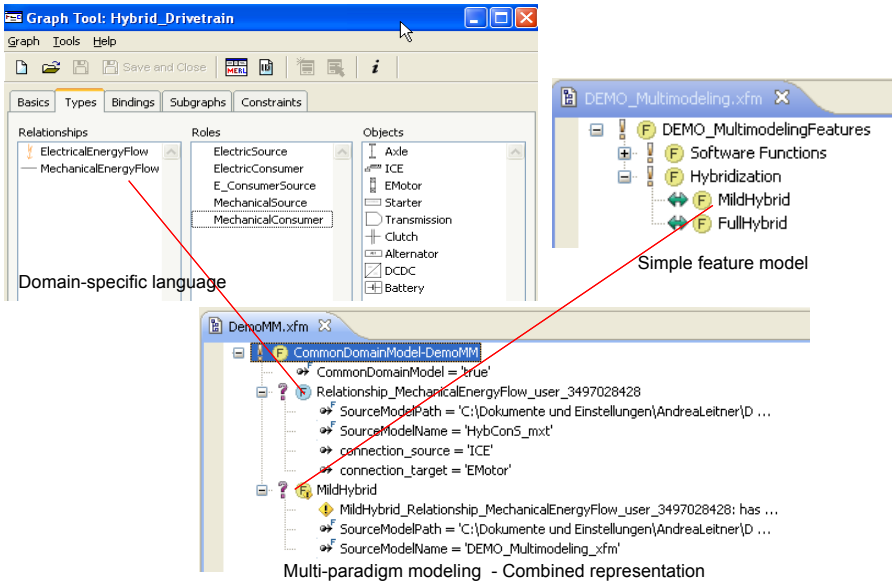


Figure 11: Multi-paradigm model combining a Feature model and a Domain-specific language on domain level. A full hybrid configuration has no direct mechanical energy flow between electric motor and ICE.

Representation complexity can be seen as an important quality metric for domain models. The improvement of domain models has several advantages: domain models have a long life-cycle in software product lines. They are used for the derivation of many products. Simpler representations reduce errors and improve productivity.

In a previous work [14] we introduced some simple metrics to compare the complexity of different domain representations. The metrics assess the interface, element, and property complexity – and, if summed up, the overall complexity. Figure 12 shows a comparison of the results when using the multi-paradigm modeling approach compared to the single-paradigm approaches. We modeled the domain, consisting of drivetrain configurations and corresponding control software functions with a FODM approach, a DSM approach and, the multi-paradigm modeling approach. The complexity of the DSL is very high compared to the other representations. This is due to the fact that vehicle software functionality is hard to describe in a DSL. The representation is very "unnatural", making it very complex. The FODM approach performs very well for a single drivetrain topology, but model complexity is increasing fast as the domain evolves. Already for a small number of topologies FODM complexity is higher than with the other approaches. The multi-paradigm modeling approach seems to unite the advantages of FODM and DSM in terms of complexity. In particular the additional modeling effort due to having two models in combination is not significant.

Summarized, this means that for four different drivetrain configurations which is a realistic practical scope the multi-paradigm modeling approach is the optimal solution.

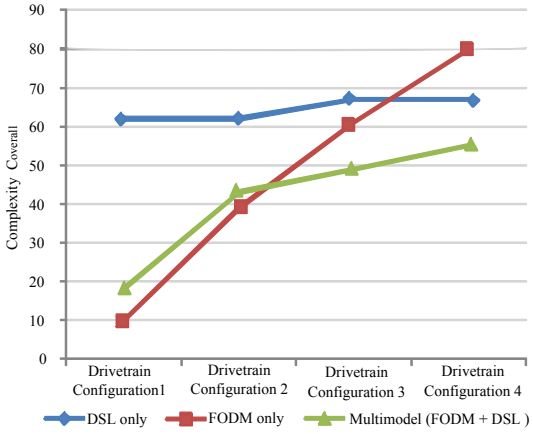


Figure 12: Complexities of HybConS domain description with FODM, DSL, and Multi-paradigm modeling compared

8. CONCLUSION

Embedded software is highly dependent on its peripheral system, which often results in heterogeneous domains. This work proposes a domain multi-paradigm modeling framework to allow the combined representation of heterogeneous domains using different paradigms. Our multi-paradigm modeling approach is also applicable to compositional variability models as defined in [17], because the different elements can be treated in the same way as traditional feature

model elements.

The domain multi-modeling framework concept has been described as well as a prototype implementation. Detailed information on how to link multiple subdomain models via a Common Domain Model is given. Constraint checking mechanisms and usability enhancements for application engineering are described. Further, the necessary domain multi-paradigm modeling activities during domain engineering and application engineering are explained.

Of course there are some drawbacks of this approach. Due to the simplicity of the constraints the domain has to be designed in a way that these constraints are sufficient. Nevertheless, constraints between domain aspects can often be described on a high level of abstraction.

As a proof of concept, a case study was presented. In a hybrid electric vehicle control unit project, the domain has been modeled with single-paradigm approaches, as well as with the described multi-paradigm modeling framework. Using simple complexity metrics, we could show that the proposed multi-paradigm modeling approach leads to reduced complexity of the domain representation.

Acknowledgment

The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG). We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

9. REFERENCES

[1] K. Bak, K. Czarnecki, and A. Wasowski. Feature and meta-models in clafer: mixed, specialized, and coupled. In *Proc. of the 3rd int. conf. on Software language eng.*, SLE'10, pages 102–122, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *Proc. of the Int. Workshop on Software Factories*, pages 16–20, 2005.

[3] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat. Consistent product line configuration across file type and product line boundaries. In *Proc. of the 14th int. conf. on SPL*.

[4] R. d. A. Falbo, G. Guizzardi, and K. C. Duarte. An ontological approach to domain engineering. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 351–358, New York, NY, USA, 2002. ACM.

[5] O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen. Adding Standardized Variability to Domain Specific Languages. In *Proc. of the 12th Int. Conf. on SPL*, pages 139 –148, 2008.

[6] F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: Mapping Features to Models. In *Companion of the 30th international conference on Software engineering*, ICSE Companion '08, pages 943–944. ACM, 2008.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[8] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[9] K. C. Kang, J. Lee, and P. Donohoe. Feature-Oriented Project Line Engineering. *IEEE Software*, 19(4):58–65, 2002.

[10] S. Kelly and R. Pohjonen. Worst Practices for Domain-Specific Modeling. *Software, IEEE*, 26(4):22 –29, 2009.

[11] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[12] K. Lee, K. C. Kang, and J. Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In *Software Reuse: Methods, Techniques, and Tools: Proceedings of the Seventh Reuse Conference (ICSR7*, pages 62–77. Springer-Verlag, 2002.

[13] A. Leitner, C. Kreiner, R. Mader, C. Steger, and R. Weiß. Towards multi-modeling for domain description. In *2nd Int. Workshop on Knowledge-Oriented Product Line Engineering: KOPLE 2011*, SPLC'11, pages 1–6, 2011.

[14] A. Leitner, R. Weiß, and C. Kreiner. Analyzing the complexity of domain models. In *19th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS12), Proc.*, 2012.

[15] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37:316–344, December 2005.

[16] pure-systems GmbH. *pure::variants User's Guide*, 2003.

[17] M.-O. Reiser, R. Kolagari, and M. Weber. Compositional variability - concepts and patterns. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1 –10, jan. 2009.

[18] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques: Research Articles. *Softw. Pract. Exper.*, 35(8):705–754, 2005.

[19] J.-P. Tolvanen. Domänenspezifische Modellierungssprachen für Produktfamilien. `http://www.sigs.de/publications/os/2001/05/OS_5_S_17_22.pdf`, visited 2010.

[20] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, 2000.

[21] M. Voelter and E. Visser. Product line engineering using domain-specific languages. In *14th Int. Conf. on Software Product Lines (SPLC 2011), Proc.* CPS, 2011.

Extending the multi-modeling domain representation from problem space to solution space

Andrea Leitner, Reinhold Weiß, Christian Kreiner
Institute for Technical Informatics
Graz University of Technology, Austria
{*andrea.leitner, rweiss, christian.kreiner*}*@TUGraz.at*

Abstract

The quality of the domain model is an important success factor in Software Product Line Engineering (SPLE). We identify Feature-Oriented Domain Modeling (FODM) and Domain-Specific Modeling (DSM) as the most important modeling approaches in practice. Especially for the representation of complex, heterogeneous domains the choice of an appropriate modeling paradigm is crucial.

Often, one modeling approach is not sufficient to develop an efficient domain model. Previously [1], we suggested an approach for the combined representation of the problem space with different modeling approaches. This combined solution has two main advantages: First, different groups of stakeholders can use their familiar terminology and modeling notation. Second, the complexity of the domain model representation can be reduced.

Here, we give ideas and pose research questions for the extension of this multi-modeling approach to the solution space. First, it has to be investigated whether or not it is possible to abstract different notations of the solution space in the same way as in the problem space. Second, the possibility to use this framework as a base for model integration has to be investigated and third, the impact of this extension on the representation complexity, an important quality metric, has to be evaluated.

We believe that this is desirable because different challenges of SPLE as well as MBD can be addressed with the proposed approach.

1. Heterogeneous domains

One major way to reduce complexity in Software Product Line Engineering (SPLE) is the separation of problem and solution space as suggested by Czarnecki and Eisenecker [2]. In their definition the problem space consists of the "*terminology used to specify family members*". The implementation forms the solution space. This is a very general definition. It only states that the problem space is a more or less abstract description of the products in the specified domain scope.

One challenge in SPLE is the understanding what a software system is. Most literature concerning SPLE deals with a software-only view not keeping in mind that nowadays nearly all software is somehow embedded. When talking about embedded software this is not only restricted to the development of traditional embedded systems. Almost everywhere software is also embedded in a process, which influences software design.

In our experience the two main domain modeling paradigms for SPLE are Feature-Oriented Domain Modeling (FODM) [3] and Domain Specific Modeling (DSM) [4]. FODM uses features, prominent user-visible characteristics of a system, to describe commonalities and variabilities of a system in a tree-structured feature model. DSM raises the level of abstraction to a domain specific language which represents domain objects. Different domain characteristics have different requirements on the modeling paradigm. Considering that the problem domain is a complex system it is very likely that different parts of the domain have different characteristics and, therefore, require different modeling paradigms. In this case we are talking of a heterogeneous domain.

For a better understanding we provide a practical example from the automotive domain here. The project is called HybConS[1] and it aims to develop a generic software architecture for control units of hybrid electrical vehicles (HEV). The characteristic of a HEV is the existence of different energy sources. Hybrid electrical vehicles may vary in drivetrain configuration,

1. http://www.iti.tugraz.at/hybcons

mechanical components, software functionalities and different supported markets. Since the drivetrain configuration influences the software, both aspects have to be described in the domain model. For the description of software aspects a feature model is suitable.

Representing the drivetrain in a feature-oriented way leads to several problems:

- One major problem is the dependence on the assembly of different mechanical components, which is hard to describe in a feature model. For example, a direct mechanical connection between the internal combustion engine and the electrical motor means that the electrical motor can not be actuated independently. If a separation clutch between the two energy sources exist, it is possible to propel the vehicle with electrical energy only. This is the basic difference between a mild and a full hybrid. Both are completely different vehicle types and, therefore, support very distinct possible software functionalities (e.g. full hybrid supports pure electrical drive, while mild hybrid does not).
- In an example topology with the electrical motor located at the wheels there are four instances of the component ”electrical motor”. This instantiation of several components is hard to describe in a feature model without specific extensions.
- Describing such compositions of mechanical components results in an extraordinary high number of relations and constraints, if possible at all. A correct and intuitive representation is hardly possible in a feature model.

On the other side, the representation of software functionality in a DSM approach is hardly feasible.

Summarized, this indicates a heterogeneous domain which requires different modeling paradigms.

2. Technical background

In case of a heterogeneous domain it can be useful to combine different representation mechanisms in order to optimize the overall representation of the problem domain. To create a multi-model representation we need to split the domain into several subdomains. For each of these subdomains an appropriate representation has to be found. In [1], we propose a solution to combine both modeling approaches based on established and well-known technology instead of introducing a new language.

The proposed solution is based on the fact that elements, different kinds of relations used to connect these elements, and properties to define elements or relations in more detail, are the basic building blocks of both paradigms.

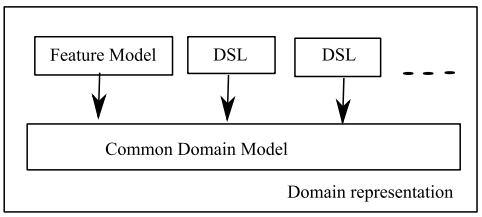


Figure 1. Conceptual realization of a multi-modeling environment

A major problem for the implementation of a combined solution is the definition of restrictions and dependencies from an element in one model to an element in another model. This means that there has to be some kind of model interface, that allows to describe these restrictions.

We propose a multi-modeling approach which uses a Common Domain Model (CDM) to represent inter-model restrictions. These restrictions can be defined on the basic building parts: elements, relations, and properties. Figure 1 illustrates the basic concept, described in [1], in more detail. Not all elements of all models need to be included in the CDM. Intra-tool constraints are described in the specific tool or language. Until now however, the multi-modeling approach is limited to the problem space.

2.1. Benefits of domain multi-modeling

In order to assess the benefits of our multi-modeling approach we tried to define metrics to evaluate the resulting domain model [5]. One useful quality metric is the representation complexity of the resulting domain model. Keeping complexity as low as possible is important in several aspects. First of all, it improves the usability and maintainability of the domain model. This is especially important, since the model is intended to have a long life cycle and is used for the derivation of products hopefully many times. Considering this, reduction of complexity results in an improvement of overall quality.

Another benefit is the support of individual representations and notations for different stakeholder views, thus improving stakeholder communication and individual understanding.

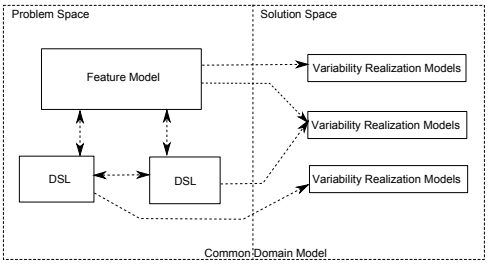


Figure 2. Domain multi-model concept

3. Problem statement

"Optimized domain model representation for problem and solution space"

Until now, the optimization of the domain model representation is restricted to the problem space [1]. For future research it would be interesting to investigate the questions posed below in order to evaluate the extension of this approach to the solution space. Figure 2 describes the basic concept. Table 1 summarizes the research questions and corresponding hypotheses.

3.1. Research Question 1:

"Is it possible to find a common abstraction for the solution space representation of FODM and DSM?"

Figure 3 illustrates the basic idea and compares the concepts to Model Driven Architecture (MDA). It separates the solution space into an abstract (middle-layer) and a concrete part. The concrete part can take several forms. Actually, the basic building parts of this middle-layer are elements, relations and properties used to abstractly describe the solution space.

Recalling that the combination of different modeling approaches for the representation of the problem space is possible because the investigated paradigms can be abstracted to elements, relations and properties. If an abstraction to these basic building parts is possible for the solution space as well, the CDM can be used for the combination of all parts of the domain (problem and solution space).

For example, pure::variants[2], a tool we use for FODM, uses an abstract model-based description of the solution space in order to configure different technical realizations. This description can be seen as a middle-layer between the problem space and the technical realization in the concrete tools or artifacts. In [6] we describe a concrete design using this layer of abstraction.

Basically, the same should be true for domain specific languages and code generators as well. Code generators are often implemented following template based patterns [7]. This description can be used to build the same kind of middle-layer representation.

As a result we can formulate the following hypothesis:

Hypothesis 1: Each code generator (DSM) and variability realization mechanism (FODM) can be abstracted to a representation that consists only of elements, relations and properties.

3.2. Research Question 2:

"Is it possible to use the Common Domain Model to improve traceability between different models in the development process?"

Here we have to investigate whether it is possible to link models in different development stages or different layers of abstraction via the Common Domain Model. Since the Common Domain Model is used to describe dependencies between different types of models it can potentially be used to support model integration.

Hypothesis 2: The CDM can at least support model integration.

3.3. Research Question 3:

"How does the combined representation of the entire domain affect the overall complexity?"

Section 2.1 discusses some implications of the multi-modeling approach for the complexity of the problem space representation. We show that the combined representation positively influences the complexity. For the extension of the combined representation on the solution space, the basic question is whether this extension has an additional positive influence or if it raises the complexity.

Hypothesis 3: The extension of the combined representation on the solution space has no special influence on the complexity.

2. http://www.pure-systems.com/

	Research Question	Hypothesis
1	*"Is it possible to find a common abstraction for the solution space representation of FODM and DSM?"*	Each code generator (DSM) and variability realization mechanism (FODM) can be abstracted to a representation that consists only of elements, relations and properties.
2	*"Is it possible to use the Common Domain Model to improve traceability between different models in the development process?"*	The CDM can at least support model integration.
3	*"How does the combined representation of the entire domain affect the overall complexity?"*	The extension of the combined representation on the solution space has no special influence on the complexity.

Table 1. Research questions and corresponding hypotheses

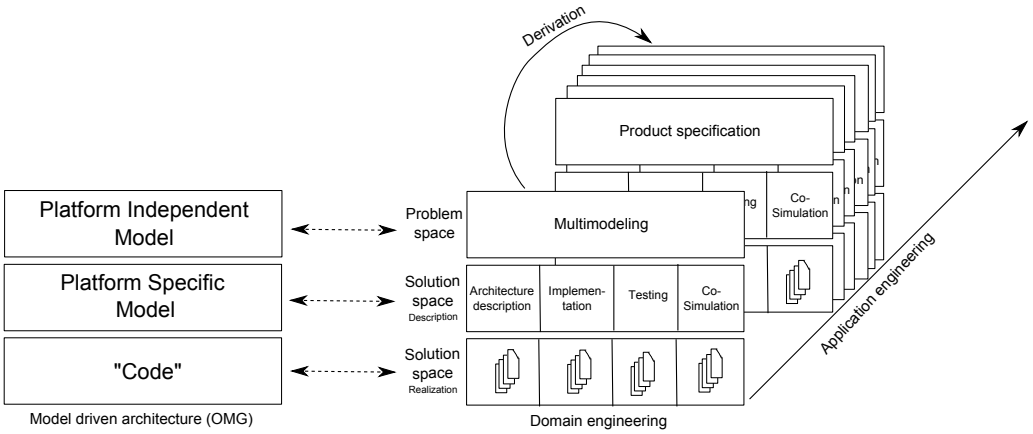


Figure 3. Comparison of MDA and SPLE using a multi-modeling approach

4. Related work

The need for heterogeneous representation mechanisms has previously been identified by different authors. In Elsner's work [8], a constraint-language has been introduced, that can be used to formulate constraints between different modeling artifacts. Bak [9], on the other side, proposes a completely new language, which integrates aspects from both approaches. The main contribution is the integration of the two modeling approaches, while providing first-class support for feature modeling. This is an improvement of previous works, where extensions to the original feature-oriented modeling approach have been suggested (e.g. cardinality-based feature modeling [10], staged configuration [11]). The goal of these extensions is the emulation of domain specific or class modeling properties in feature-oriented representations. In contrast, we suggest to use existing modeling techniques that have been proven useful over the years.

Voelter et al. [12] describe the limitations of feature-oriented domain modeling. They propose the combination of the feature-oriented approach with domain specific languages wherever the descriptive power of features are not enough.

Heidenreich et al. [13] propose a solution for mapping of features to any Ecore-based representation of the technical realization. In contrast to the approach proposed in this paper, the mapping spans from the problem space to the solution space and is not meant to realize different representations in the problem space.

Haugen et al. [14] describe a separated language approach to specify variability in DSL models. They propose a Common Variability Language (CVL) together with variability resolution mechanisms embedded in the OMG metamodel stack. This allows to describe variability in potentially all MOF-based languages, including UML, as well as MOF- and UML profile based DSLs. While being a general and clean approach to handle variability, it does not seem directly applicable to feature abstraction hierarchies and their complex constraints.

5. Future work

In future work, we need to implement and evaluate the proposed research directives. For RQ 1, different code generators have to be investigated first. In a next step, the code generator representation has to be converted to a more abstract representation, as shown in Figure 3. If this is possible, the current multi-modeling implementation can be extended to support the solution space as well. For the evaluation of RQ 2 the framework has to be used for model integration. In the last step the resulting domain model (consisting of problem and solution space) has to be evaluated using the metrics defined in [5].

6. Conclusion

Nowadays many domains are heterogeneous by nature. We identify the challenge that different subdomains require the use of different modeling paradigms. Previously, we proposed a multi-modeling solution for the problem space. We believe that the extension of the approach to the solution space is desirable, because it enables the combination of different realization mechanisms (code generator, ...). Additionally, it would be possible to define dependencies or constraints in different combinations (problem space - problem space, problem space - solution space, solution space - solution space).

Acknowledgment

The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

References

[1] A. Leitner, C. Kreiner, R. Mader, C. Steger, and R. Weiß, "Towards multi-modeling for domain description," in *2nd Int. Workshop on Knowledge-Oriented Product Line Engineering: KOPLE 2011*, ser. SPLC'11, 2011, pp. 1–6.

[2] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[4] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[5] A. Leitner, C. Kreiner, and R. Weiß, "Analyzing the complexity of domain models," in *19th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS12), Proc.*, 2012.

[6] A. Leitner and C. Kreiner, "Managing ERP Configuration Variants: An Experience Report," in *Proc. of the 1st Workshop on Knowledge oriented product line engeenering*, 2010.

[7] M. Voelter, "A Catalog of Patterns for Program Generation," http://www.voelter.de/data/pub/ProgramGeneration.pdf, voelter - ingenieurbro fr softwaretechnologie, Tech. Rep., 2003.

[8] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat, "Consistent product line configuration across file type and product line boundaries," in *Proc. of the 14th international conf. on SPL: going beyond*, ser. SPLC'10, 2010, pp. 181–195.

[9] K. Bak, K. Czarnecki, and A. Wasowski, "Feature and meta-models in clafer: mixed, specialized, and coupled," in *Proc. of the 3rd int. conf. on Software language eng.*, ser. SLE'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 102–122.

[10] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *Proc. of the Int. Workshop on Software Factories*, 2005, pp. 16–20.

[11] K. Czarnecki, "Overview of Generative Software Development," in *UPP*, 2004, pp. 326–341.

[12] M. Voelter and E. Visser, "Product line engineering using domain-specific languages," in *14th Int. Conf. on Software Product Lines (SPLC 2011), Proc.* CPS, 2011.

[13] F. Heidenreich, J. Kopcsek, and C. Wende, "FeatureMapper: mapping features to models," in *Companion of the 30th international conference on Software engineering*, ser. ICSE Companion '08. ACM, 2008, pp. 943–944.

[14] O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding Standardized Variability to Domain Specific Languages," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, 2008, pp. 139 –148.

Lightweight introduction of EAST-ADL2 in an automotive software product line

Andrea Leitner, Nermin Kajtazovic, Roland Mader, Christian Kreiner, Christian Steger, Reinhold Weiß
Institute for Technical Informatics
Graz University of Technology, Austria
{*andrea.leitner, nermin.kajtazovic, roland.mader, christian.kreiner, steger, rweiss*}*@TUGraz.at*

Abstract

This paper describes the technical aspects of the transition to a software product line approach in the automotive domain. One major challenge is the current existence of two different emerging standards for this domain, AUTOSAR and EAST-ADL2. These potential standards should be borne in mind during the software product line introduction because they may someday become mandatory. In addition, the existing development process should be changed as little as possible, and one final important requirement for the software product line is the implementation of a single point of control to ensure consistency between various development artifacts.

To this end, we propose a lightweight introduction of EAST-ADL2 as a documentation tool only as an initial step. This is achieved by extracting structural information from AUTOSAR models and automatically generating the corresponding EAST-ADL2 representation. The automatic generation ensures consistency between AUTOSAR and EAST-ADL2 models. As an important side effect, variability information can be extracted in this transformation step and used to build an EAST-ADL2 compositional variability model. This model can then be mapped to the central domain model and used to configure the EAST-ADL2 documentation to the other development artifacts consistently.

In this way, we can accomplish the lightweight introduction of EAST-ADL2 in the development process through the automatic generation and use the generated variability information for configuration from a single control point.

1. Introduction

This paper describes one important research outcome of our current project. The goal of the HybConS[1] project is to generate a generic software architecture for the electronic control unit (ECU) of Hybrid Electrical Vehicles (HEV) [1]. One part of this project is concerned with the introduction of a variability management environment in terms of a software product line (SPL) [2] for the development of automotive control software. A major requirement is to create a "single point of control" for various development artifacts. In this context, "single point of control" means one point from which all variability can be controlled consistently across the various development artifacts.

To better understand the challenges, it is important to get an overview of the development and context. In the automotive domain, model-based development with MATLAB Simulink is very common. The predominant development process is the V-model, as shown in Fig. 1 (left). To improve standardization and interoperability on the software level, the AUTOSAR consortium has developed the AUTOSAR standard [3]. Similar efforts have been made on the design level, which have resulted in EAST-ADL2 [4], an architecture description language for the automotive domain. The two concepts are described in more detail below.

The EAST-ADL2 metamodel provides a sophisticated variability package. With this package, it is possible to describe variability at different levels of detail and to derive concrete products. At the beginning of our project we evaluated the applicability of EAST-ADL2 as an SPL environment. We eventually decided to choose another approach, for reasons discussed below.

Our motivation is the need for a software product line approach in the current development process. As mentioned above, one goal is to develop a generic software architecture. Therefore, it is essential to describe variability systematically. To date, a single system development approach has been used. Variants are often built from existing products by copying and adapting existing code. This is an error-prone approach that should be changed.

Since we know that AUTOSAR is becoming a standard, and EAST-ADL2, intended as a systematic way to describe the domain, may also be adopted at some point, we want to include these concepts in our adoption process from the beginning. Currently, we are dealing with an established development process, so it is not possible to make major changes to the process during production. This is the main reason for the evolutionary and lightweight approach.

1.1. Motivation

In order to implement a single point of control it is not only necessary to describe variability, but also to control all kinds of development artifacts consistently. This means that variability mechanisms have to be integrated into all of the

1. http://www.iti.tugraz.at/hybcons

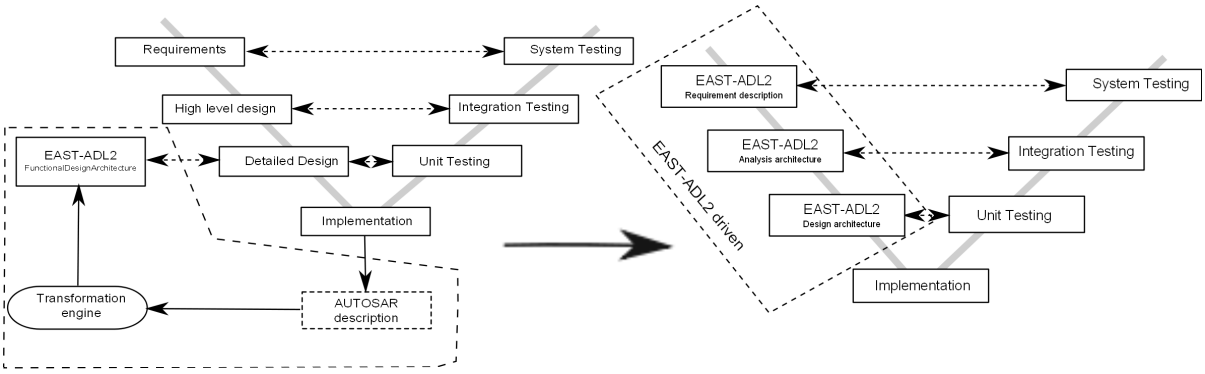


Figure 1. Lightweight and objective integration of EAST-ADL2 in the development process

relevant development stages and tools. This includes not only system description, but also support for implementation, tests, requirements and safety considerations. The lowest level of EAST-ADL2 has been aligned to the AUTOSAR metamodel. This makes it possible to describe AUTOSAR templates within the EAST-ADL2 model. UML and integrated behavior notation only partially enable the direct representation of behavior [5]. Other behavior descriptions, for example in MATLAB Simulink, can be linked to behavior representation elements of the EAST-ADL2 metamodel [4].

To use EAST-ADL2 and the corresponding tool environment as an integrated engineering environment, we have to link the currently used MATLAB Simulink models to the EAST-ADL2 description. This means that every time an interface in the Simulink description changes (e.g. adding a signal), the EAST-ADL2 description has to be changed as well. In a realistic development scenario, the consistency between the models would eventually be lost. Another problem that stems from the link to behavior is the loss of a single point of control. If the behavioral description is not part of the EAST-ADL2 model, it is also impossible to include the variability mechanisms provided by EAST-ADL2 in this respect. On the other side, variability support for the MATLAB Simulink tool chain exists. The variant management tool pure:variants[2] provides the means to describe variability in both MATLAB Simulink and TargetLink models [6].

In this paper, we propose a lightweight approach for the integration of EAST-ADL2 in the automotive software development process, as shown in Fig. 1. On the left side, the implementation of our lightweight introduction is illustrated in the dashed area. It is an extension to the current development process. The long-term objective is to completely integrate EAST-ADL2 in the development process, as shown on the right side. The initial strategy in this scenario is the use of EAST-ADL2 as a documentation tool. We achieve this by extracting both structural and variability information from an AUTOSAR model. Software components described with AUTOSAR are then transformed into an EAST-ADL2 Functional Design Architecture (FDA) model. This strategy can also be used as a starting point for the automatic extraction of variability information from legacy systems. Currently, this is only possible if the variability information is present in the AUTOSAR description. Variability mechanisms have been supported in AUTOSAR since version 4. This variability information can be used to integrate the EAST-ADL2 model into an existing product line project as one type of development artifact.

In summary, this proposal has two main contributions:

1) The lightweight introduction of EAST-ADL2 in the development process requires a transformation from AUTOSAR to EAST-ADL2. The detailed mapping is described in the following sections.
2) The implementation of a single point of control with respect to variability. Since EAST-ADL2 does not cover all of the steps of the current development process, we treat the EAST-ADL2 model as a development artifact. This means that we integrate variability mechanisms, but control them from one central model. Basic variability information can be automatically extracted in the transformation step.

Sec. 2 provides a short introduction to the concepts of EAST-ADL2 and AUTOSAR and summarizes related work. Sec. 3 describes the concepts of the two major parts of this proposal: Sec. 3.1 describes the basic mapping strategy used for the transformation, and Sec. 3.2 describes the integration of the model in an SPL. Finally, Sec. 4 describes the implementation of the two parts, and Sec. 5 evaluates the results of our transformation approach.

2. http://www.pure-systems.com/

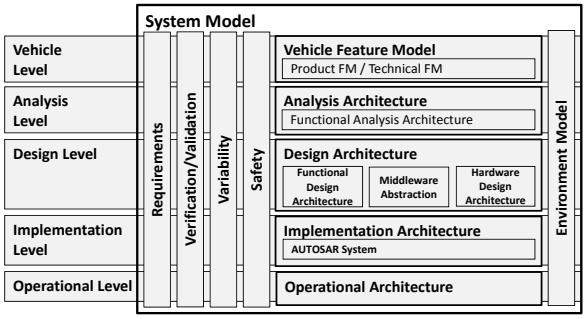


Figure 2. EAST-ADL2 domain structure [4]

2. Related Work

2.1. EAST-ADL2 overview

The Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) is an architecture description language for the automotive domain. It has been developed within the scope of the EAST-EEA (EAST - Embedded Electronic Architecture), ATESST and ATESST2[3] (Advancing Traffic Efficiency and Safety through Software Technology) projects. Its primary goal is to provide a detailed description of the entire system and to improve communication in the development environment. This is achieved through the representation of a system on different layers of abstraction (see Sec. 2.1.1) combined with additional modeling aspects, such as variability, requirements modeling, feature modeling, environment modeling, and system level analysis [4].

2.1.1. Structure. As illustrated in Fig. 2, an EAST-ADL2 system model consists of four layers.

On the vehicle level, an abstract description of the entire system in terms of features (as defined in [7]) is provided. This abstract representation serves as a basis for communication with stakeholders (e.g. customers). On the next level, the analysis level, abstract functions are defined by breaking down the requirements and features. This abstract functional description corresponds to a domain concept. For example, it can describe an environment model, devices interacting with an environment, or functions [8]. In the next level, components are further divided into either hardware (e.g. sensors, actuators) or software components. In addition, middleware is modeled to connect device-specific functions to design functions. Design functions form the *Functional Design Architecture* (FDA), which is the model that this paper focuses on. An abstract description of the hardware is captured in the *Hardware Design Architecture*. The implementation level is not explicitly defined in the EAST-ADL2 metamodel. Instead, some of the concepts have been aligned to the AUTOSAR specification.

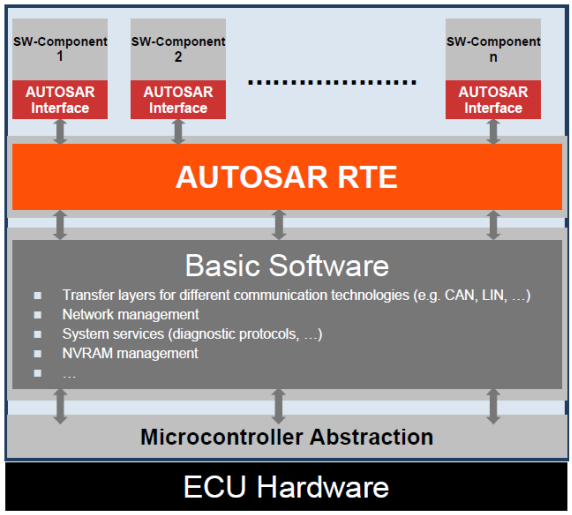


Figure 3. AUTOSAR architecture overview [10]

2.2. AUTOSAR overview

AUTomotive Open System ARchitecture (AUTOSAR[4]) represents a standardized, open automotive software architecture developed by a group of more than 150 companies, consisting of automotive manufacturers and suppliers. The demand stems from the growing complexity in automotive software development due to extensive innovations in E/E systems [9].

The main advantages of the AUTOSAR approach are the separation of hardware and software and the resulting reusability of components. The promotional slogan is *"Cooperate on standards - compete on implementation"*. This means that it should become possible for OEMs and other suppliers to exchange components without revealing any implementation details [10]. The basic AUTOSAR architecture is shown in Fig. 3.

2.2.1. ECU Software Architecture. One essential design concept of AUTOSAR is the separation between ECU-specific and ECU-independent software, i.e. basic software and application software, respectively. An intermediate layer, the virtual function bus, acts as an abstract representation of the communication infrastructure for all software components. Since the communication specification is ECU-independent, high levels of modularity, scalability, exchangeability and reusability are achieved [11]. All this is located at the system level of the AUTOSAR methodology. The

3. http://www.atesst.org

4. http://www.autosar.org

implementation of the virtual function bus is provided by the runtime environment (RTE), which is the core of the AUTOSAR architecture [3].

2.3. General literature

Since EAST-ADL2 and AUTOSAR are two relatively new concepts, few attempts for transformations and mappings have been described in the literature. Only one approach has described a possible mapping strategy at the implementation level. In [12] and [13], the project EDONA is introduced. The aim of this project is to integrate heterogeneous tools into one platform in order to support the cooperative development of embedded automotive systems. The most interesting part of this project in the present context is the transformation block called *ARGateway*. It transforms the software design described in EAST-ADL2 into the AUTOSAR software component description. Since the target in this case is the implementation level, both structural and behavioral information are necessary. This requires a detailed mapping strategy.

Two design patterns for tool integration are presented in [14]. One pattern describes the integration via a data model, whereas the other integrates the process flow. Our approach is based on the ideas of the second pattern. We implemented tool adapters for Papyrus and AUTOSAR, as well as a semantic translator (Mapping API). Since there are only two tools and two languages involved in the transformation, a tool backplane is not necessary here.

In [15], a method for transformations between different model formats is presented. One of the two use cases describes the transformation from MATLAB Simulink to EAST-ADL2. The mapping is described in a so-called structural bridge, which preserves the semantics of the original model. A technical space bridge is used to access the original model data. In contrast to our approach, it is not possible to realize variability representations.

In [16], the transformation between different AUTOSAR metamodel versions is described. This is necessary to ensure interoperability.

A summary of common Model-to-Text and Model-to-Model transformation approaches is provided in [17]. The transformation implementation in this work is achieved as a Direct-Manipulation Approach.

In [18], three fundamental aspects of transformation mechanisms are introduced: *scope*, *direction*, and *stages*. The scope restricts which parts of the program are affected by the transformation. In our case, these are the model elements which should be contained in the target model. The direction of a transformation determines whether the structure of the source or the target program drives the transformation process. In our case, the target model (EAST-ADL2) is the driving part, since only structural and variability information is required there. The stage aspect of a transformation determines the necessary iterations. In our case, two iterations are necessary. This is due to multiple structural dependencies.

A good overview of the terminologies and technologies used for model transformations is given in [19]. Our approach can be classified as both *Reverse engineering* and *Migration*.

Domain modeling based on legacy systems is a common approach. Manual modeling using guidelines (e.g. [20]), or semi-automated knowledge extraction (e.g. [21]) can be used. Another interesting approach can be found in [22]. Here, an existing software framework serves as a basis for domain description in the solution space. The metamodel for a consistent problem-space DSL is extracted from this description. This in turn guarantees that there is a generator that can execute all concrete product descriptions using the original framework.

Haugen et. al. [23] describe a separated language approach for specifying variability in domain-specific language models. They propose a Common Variability Language (CVL) and corresponding variability resolution mechanisms embedded in the OMG metamodel stack. This allows for the description of variability in potentially all MOF-based languages, including UML, as well as MOF- and UML profile-based domain-specific languages. Although this represents a general purpose, clean approach for handling variability and providing a single point of control, it is not applicable for the representation of variability in Simulink-based implementations.

3. Design considerations

As mentioned before, our approach consists of two major parts. The first step is the transformation from the AUTOSAR component description to higher level description in EAST-ADL2. Sec. 3.1 provides a detailed description of this transformation step. Second, the EAST-ADL2 representation should be used as a development artifact, which can be controlled from one single point. Sec. 3.2.1 describes the extraction of variability information and the connection to the common domain model.

3.1. AUTOSAR to EAST-ADL2 Mapping

Our mapping strategy defines how elements from AUTOSAR are transformed into the EAST-ADL2 Functional Design Architecture. The FDA does not describe the software architecture from an implementation point of view, but rather from the design perspective. Since the two models describe software on different levels of abstraction, it is necessary to analyze different mapping strategies in order to generate a correct model (documentation) and to reduce losses in the transformation process.

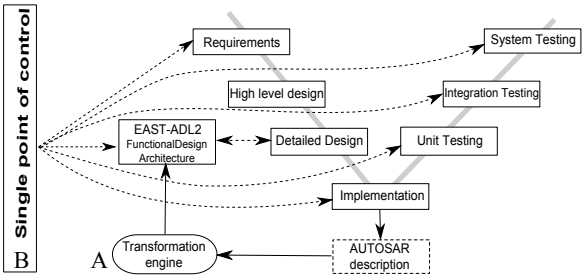


Figure 4. Graphic representation of the two major project parts: A) Extraction of an initial EAST-ADL2 model from existing software B) Introduction of a single point of control for variability

The EAST-ADL2 documentation suggests two types of mappings:

Detailed mapping is used if behavioral information from the target model is required. In this case, the function corresponds to AUTOSAR *runnable*.

Black-box mapping is adequate if only structural information is required. In this case, the function is mapped to an atomic or composite software component in AUTOSAR.

3.1.1. Mapping strategy definition. A mapping proposal in [8] defines the FDA as a functionality of the application software architecture in AUTOSAR. It also gives very important and helpful suggestions about how to execute structure and behavior mapping in detail. However, it is necessary to first decide which of the two mapping concepts proposed in EAST-ADL2 to use. Since, in this case, structural information is more important than behavioral, the black-box solution is chosen. This level of detail excludes the mapping of runnable entities and reduces the effort required to implement the transformation. To the best of our knowledge, no detailed mapping strategy between EAST-ADL2 and AUTOSAR has been described. Nevertheless, there is sufficient information about similarities between models and possible mapping solutions, which can be used to define a detailed mapping strategy.

Next, we need a detailed specification that describes how elements and their properties are mapped. Starting with the description of a complete information flow from a hardware sensor to its software representation, the most important (or all) groups of different software components are present and can be analyzed.

Fig. 5 shows a sample information flow which captures the vehicle's velocity. The lower layer shows the software representation, and the upper layer represents the corresponding hardware description.

The physical value of the velocity is captured by the sensor and converted here to *current*. Typically, this signal

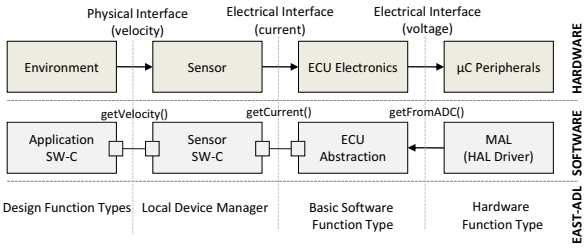


Figure 5. Information flow in AUTOSAR, modified [24]

is converted to a microcontroller input type (e.g. voltage) by the *ECU Electronics* component. The signal is then handed over to the standardized *Hardware Abstraction Layer Driver* and is thereafter available in the software. The next steps describe the reverse conversions on the software layer. The *ECU Abstraction* transforms the value received from the *Hardware Abstraction Layer Driver* to *current* and delivers it to the software representation of the sensor. The sensor software component then transforms *current* to the software representation of the physical value of the velocity [24].

We used the documentation of metamodel elements in combination with use cases, such as the one described above, to find analogies to EAST-ADL2 model elements. The detailed mapping specification is also based on material collected from [8], [24], [12], and [13].

3.2. Single point of variability control

As mentioned before, not all parts of the development process can be represented in EAST-ADL2. In particular, the representation of variability in the implementation is currently not possible in a scenario with EAST-ADL2 as the software product line core. In our scenario, the EAST-ADL2 model is used as a development artifact similar to implementation and test, which are controlled from one single point, as shown in Fig. 4B. This single point of control is a feature model represented in pure::variants. To control variability from this common domain model, a connector between Papyrus (EAST-ADL2) and pure::variants has to be implemented. The implementation is described in further detail in Sec. 4.2.1. EAST-ADL2 variability concepts are described below.

3.2.1. Extracting variability information. We first assume that it is not important to document why variability occurs. Therefore, all variants are handled as internal variants. For further propagation towards the analysis block, the developer must decide what will be visible to the customer.

The variability part of the EAST-ADL2 metamodel contains two types of traces. First, domain assets are traced to variation points on the artifact level, which is known as artifact-level variability. Different options of these variation points are handled by special metamodel elements,

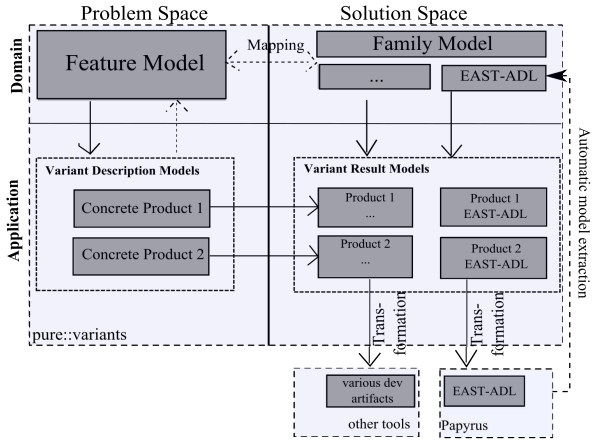


Figure 6. Single point of variability control concept

called variable elements, which have direct traces to domain assets. Second, domain assets are traced to features and feature models. These feature models are orthogonal to the assets model and only have references to them. In EAST-ADL2, feature models are composed in a hierarchical order, whereby the root feature model is called the technical feature model. It is configured by a single-vehicle-level feature model. The combination of both levels of variability makes it possible to define the product line and to configure the system in EAST-ADL2.

To extract variability information, the variable elements in the target model must first be identified. There is a predefined set of prototypes in the FDA which may vary [25]:

- FunctionPrototype
- FunctionPort
- FunctionConnector
- HardwareComponentPrototype
- HardwarePort
- ClampConnector

Combining these prototypes makes it possible to define variability at various levels of granularity, such as subsystems, software components, cross-cutting characteristics, specializations and functionality. In an initial proof-of-concept only the first two options have been implemented.

3.2.2. Connecting variability representations. Fig. 6 illustrates the separation of variability concerns. The figure is aligned to the concept of pure::variants, which consists of four major parts. In the problem space, the domain is abstractly defined in a feature model. There, the commonalities and variabilities of different products are described, and the domain knowledge is made explicit. Concrete product descriptions are derived from this feature model. In the solution space, technical realizations are described in so-called family models. There is one family model for each type of artifact or tool under variability control. These models are consistently controlled from the feature model. In our scenario, we generate a vehicle-level feature model in EAST-ADL2, which is used as the family model for the configuration of the software architecture described in EAST-ADL2. Since the two representations have a similar structure, the import functionality is straightforward. The remaining manual task is the mapping between the problem and the solution space in pure::variants, which introduces the links for the configuration.

4. Implementation

4.1. Implementation of the transformation process

Fig. 7 depicts the transformation process.

The implementation consists of three basic components: a *SAX*[5] *Parser*, an *AUTOSAR Model Builder* and a *Transformation engine*. The following sections describe these components in greater detail.

4.1.1. Parser. The transformation process starts with the selection of a parser. For each AUTOSAR schema version there is a SAX parser, which is internally selected. The model parsing process results in an element table, which is used as a temporary storage for the calculated absolute path for each component and the references of this component. First, path information is necessary to identify the references between elements.

Resolving Paths. Component paths are stored as relative paths in the AUTOSAR model. For further processing, absolute paths have to be recovered.

The absolute path in AUTOSAR is built by concatenating all package names from the top to the corresponding element. This is the first information stored in the element table.

Resolving References. In the next step, references for each component have to be resolved. References in AUTOSAR can be expressed in two ways: (1) with absolute paths and (2) with relative paths. Absolute paths always start with a `'/'` followed by the root package. They are used as a unique identifier for a package. Capturing absolute references does not require any post-processing effort. Relative references in AUTOSAR are the same as those found in typical file systems. They represent the “outer right excerpt” of an absolute reference and are related to the containing package. To identify the target element, this path needs to be converted into an absolute path. For this calculation, the schema element *ReferenceBase* is used. There are four possible contents of the *ReferenceBase*. In the best case, it contains the left part of the absolute path. In the worst case, it contains a reference to some other reference base.

5. http://sax.sourceforge.net/

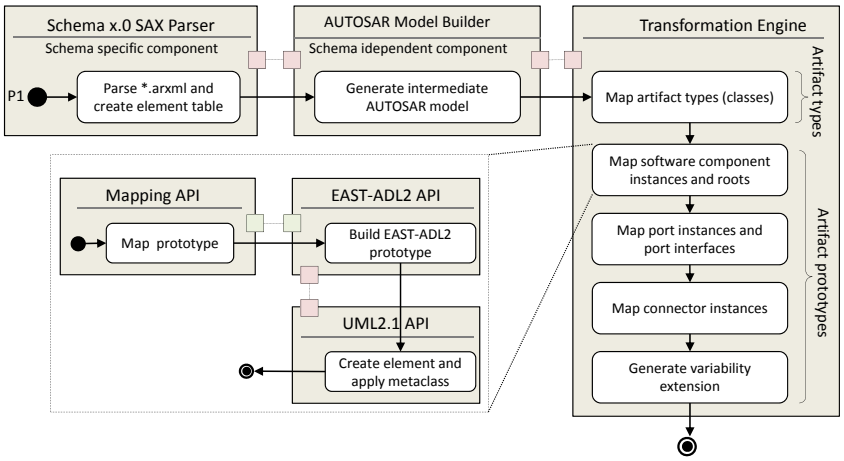


Figure 7. Mapping process from AUTOSAR to EAST-ADL2

4.1.2. AUTOSAR Model Builder. The AUTOSAR Model Builder takes the element table as its input and builds a version-independent intermediate model. In this way, there is more flexibility in the implementation of the source model, although it causes minor, insignificant performance losses.

4.1.3. Transformation Engine. In the last step, the *Transformation Engine* builds the EAST-ADL2 model according to a defined mapping strategy. For this purpose, a Mapping API has been implemented in Java to represent the mapping strategy.

The mapping has to follow a specific sequence, since some mappings depend on previous information. Types are mapped first. An example of a type is a composite software component containing prototypes. It is typically referenced by a prototype which is typed by this composite. Therefore, types have to be readily available at the time prototypes are mapped.

In the next step, prototypes are mapped. Since AUTOSAR provides constructs to express references to the root software components explicitly, these references are mapped as root prototypes within the Functional Design Architecture, i.e. they are first-level functions in the FDA. Ports and port interfaces are mapped when all software components already exist in EAST-ADL2. Connectors must also be mapped at the end of the mapping process, since they refer to ports.

4.2. On single point of control implementation

4.2.1. Variability extraction. The first step towards a single point of control is the extraction of variability information. AUTOSAR provides four different mechanisms to describe variation points [3]: aggregations, associations, attribute values and property sets.

For our prototype implementation, the aggregation pattern suffices to enable variability in sub-systems and components. The main challenge in this sub-process is the generation of variability logic for the captured common and variable model assets. Metamodel elements from the EAST-ADL2 variability package are generated and interrelated to each other. The process generates artifact-level and vehicle-level elements. The vehicle level provides the interface for the system configuration. Artifact-level variability, on the other hand, is used for the internal configuration. EAST-ADL2 uses a so-called Compositional Variability Model [25]. After completing the model transformation, the root component is identified, and the variability extension is created by traversing the component tree in top-down order.

The first activity in the process takes an element and checks whether it is an elementary or a composite software component. If it is elementary, it has to be checked to see if it "varies". If it does, a construct *VariableElement* is created within a variability extension and referenced to this model asset. Variability is not represented directly in the model, but rather in an orthogonal way. This makes the model and the variability representation independent from each other.

As shown in Fig. 8, for each composite, the following parts are created:

- a **public feature model**, which contains features that reference the content of a composite,
- an **internal binding**, which specifies the rules for how a variable content element is affected by selecting features from the public feature model, i.e. configuration decisions.

The public feature model is only visible for the composite container, i.e. for the composite in the next highest level of the hierarchy. The composite's internal binding defines how this public feature model is configured. This is how the

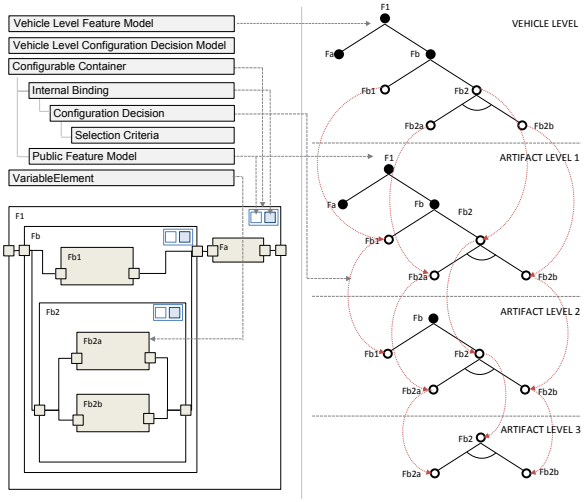


Figure 8. Compositional variability concept of EAST-ADL2 [25]

configuration is propagated over the whole hierarchy. For each feature of a public feature model, one configuration decision corresponding to the concept of transfer functions is created. A collection of these configuration decisions related to one public feature model is thereby packaged into one internal binding. The internal binding and public feature model are the most important constructs, since they contain the whole logic for the system configuration for a specific composite software component.

If a composite contains other composite software components, they need to be processed recursively. The processing of a composite software component is done after the generation of the variability extension for all its elementary and composite parts. Finally, both the internal binding and public feature model must be packaged into a configurable container. This construct is created and referenced to the composite software component.

The last two activities of the process are part of vehicle-level variability. Here, the vehicle-level feature model and a bridge connecting the vehicle level and artifact level in EAST-ADL2 are generated.

In our case, the vehicle-level feature model is generated by simply cloning the root technical feature model. With this simplification, the step can be automated. Further conceptual abstraction would require manual intervention. Still, as a pragmatic approach, it propagates variability not only to the technical feature model, but also up to the vehicle-level feature model. The only difference between the two abstraction layers in our case is that features inside the vehicle-level feature model are represented by an element *VehicleFeature*, which is a specialization of an element *Feature*. It is extended for attributes such as *cardinality*,

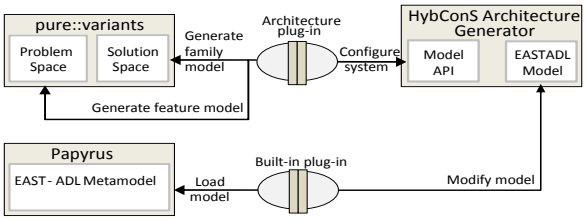


Figure 9. HybConS variability connector plugin structure

isOptional, etc. The bridge to the artifact level is created by mapping these vehicle-level features to the features of the root feature model. In fact, they are identical, but the feature model on artifact level is not visible for the configuration.

4.2.2. Variability connector. The single point of control is achieved with Eclipse plugins. This is easily done, since both pure::variants and Papyrus are Eclipse-based tools. Fig. 9 shows the structure of the variability connector architecture. The *HybConS Architecture Generator* represents the transformation engine described above. The *Architecture plug-in* is used to import the EAST-ADL2 vehicle-level feature model into the pure::variants solution space and to configure the system from the common domain model in pure::variants. Reading from and writing to EAST-ADL2 models in Papyrus is performed with a built-in plugin that provides this functionality.

The EAST-ADL2 vehicle-level feature model is configured from the pure::variants feature model. The detailed configuration of the composite variability representation in EAST-ADL2 is performed using the built-in configuration capabilities of Papyrus.

5. Evaluation

The prototype does not cover the whole set of AUTOSAR and EAST-ADL2 metamodel elements. Approximately 14% of the AUTOSAR metamodel and 64% of the EAST-ADL2 metamodel have been implemented for this prototype. The relatively limited coverage of the AUTOSAR metamodel in this first version can be explained by the fact that only basic structural elements are necessary in our project scenario.

First, we wanted to evaluate the accuracy of the transformation. It is not easy to automate this test, since this would require a second independent mapping process. Therefore, the accuracy of the generated EAST-ADL2 representation was assessed manually. Some UML tools can generate diagram information for a graphical representation of given model information. This feature was used to visualize the generated model. These graphical representations were compared manually.

As a second evaluation criterion, we measured the performance of transformations. Tab. 1 gives an overview of the

	Model sizes [# model elements]						
Types	9	34	59	109	209	409	310
Prototypes	6	56	106	206	406	806	518
Ports	12	112	212	412	812	1612	5410
Connectors	14	114	214	414	814	1614	4806
File s. [KB]	29	136	242	456	882	1736	12783
Process	**Run time [s]**						
SAX Parser	0.14	0.21	0.27	0.40	0.67	1.29	3.95
Model Build	0.03	0.03	0.06	0.06	0.07	0.10	0.87
Transform.	1.11	1.74	2.16	3.50	6.13	10.97	29.34
Total	**1.29**	**2.00**	**2.50**	**3.97**	**6.88**	**12.37**	**34.16**
Variability	0.26	1.56	2.30	5.43	16.82	76.54	69.53

Table 1. Transformation performance metrics

results for seven different use cases. For runtime estimation the following configuration was used: CPU Intel RT2400, 1.83GHz, 2.00 GB RAM, Windows 7 32-bit, JDK 1.6.

The use cases differ in size, as shown in the first part of the table. It shows both the number of model elements and model sizes. The parts of the transformations are types, prototypes, ports and connectors. The last parameter describes the file size.

The lower part of Tab. 1 shows the runtimes for the 3 main processing steps. The total shows that the transformation process is completed within an acceptable amount of time for an average sized model. The third part of the table shows the time required to extract variability information. This process is the slowest, but it has to be perfomed only once.

The last part of the evaluation shows the automation capabilities provided by this approach. Fig. 10 gives an overview of the automatic and manual tasks. The implementation of automotive software is still a manual task. Some tools can generate AUTOSAR description automatically. The description serves as input for our transformation engine. The transformation step is performed automatically.

For the implementation of a single point of control, the user has to add variability information to the AUTOSAR description. If variability information exists, it can be extracted automatically to build a compositional variability model in EAST-ADL2. The import of the generated EAST-ADL2 vehicle-level feature model as a pure::variants family model can be automated as well. The last step, the mapping between the pure::variants feature model and the family model that represents the EAST-ADL2 configuration, must be performed manually.

With this approach, the variability of existing products can be extracted and propagated to a higher level, thereby serving as a base for the domain modeling process. Of course, a higher level of abstraction requires manual work.

In the application engineering process, the configuration for a concrete product is described in pure::variants. The bridge between pure::variants and EAST-ADL2/Papyrus enables the automatic propagation of the configuration through all levels.

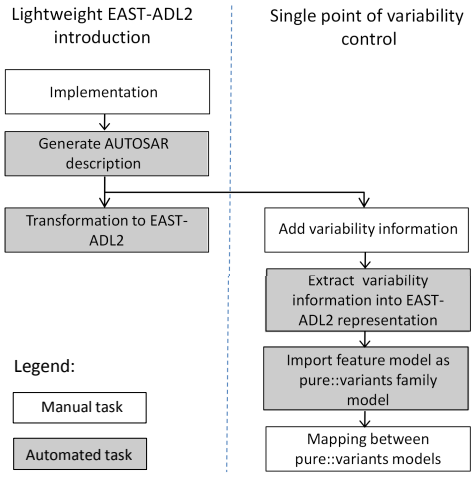


Figure 10. Evaluation flow

6. Conclusion

This paper focuses strongly on the automotive domain. For this setting, we suggest adopting a software product line approach, while bearing in mind that a new standard for architecture description is evolving in this domain. This future standard has its own variability modeling capabilities, which are not used here due to the lack of single point of control mechanisms.

We suggest an initial transformation from AUTOSAR to an EAST-ADL2 Functional Design Architecture, which enables a lightweight introduction of EAST-ADL2 in a standard automotive software development process. With this approach, the existing development process does not change. Software implemented with MATLAB Simulink can be transformed into an AUTOSAR description. This information is used by our transformation engine to automatically generate a documentation in EAST-ADL2. We have defined a mapping strategy and implemented this strategy in a transformation engine.

The second part of this paper described the use of this transformation to extract variability information. We used the gathered variability information to automatically build a compositional variability model in EAST-ADL2 notation. The extracted information can be used in the same way as other development artifacts, thereby enabling a single point of control.

To date, a prototype showing the basic functionality has been implemented. In future development, this prototype should be enhanced to support the full range of elements. In our opinion, it is not possible to automate the variability extraction process further because some kind of variability information has to be provided at some point in time. We believe that the current implementation is a good starting

point. A more advanced approach would be to compare existing implementations and analysis to extract potential variability. This logic could easily be integrated into our approach.

Acknowledgment: The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

References

[1] M. Ehsani, Y. Gao, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design*, 2nd ed. CRC Press, 2010.

[2] P. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[3] AUTOSAR, "Technical overview," http://www.autosar.org/, 2008, release 3.0, Document version 2.2.1.

[4] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. T. Kolagari, M. Törngren, and M. Weber, "The EAST-ADL architecture description language for automotive embedded software," in *Proc. of the 2007 Int. Dagstuhl conf. on Model-based engineering of embedded real-time systems*, ser. MBEERTS'07. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 297–307.

[5] C.-J. Sjöstedt, D.-J. Chen, P. Cuenot, P. Frey, H. Lönn, D. Servat, and M. Törngren, "Developing Dependable Automotive Embedded Systems using the EAST-ADL; representing continuous time systems in SysML," Royal Institute of Technology Stockholm, Tech. Rep., 2007.

[6] D. Beuche and J. Weiland, "Managing Flexibility: Modeling Binding-Times in Simulink," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, R. Paige, A. Hartman, and A. Rensink, Eds. Springer Berlin / Heidelberg, 2009, vol. 5562, pp. 289–300.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[8] P. Cuenot, P. Frey, R. Johansson, H. Lönn, M.-O. Reiser, D. Servata, R. T. Koligari, and D. Chen., "Developing automotive products using the EAST-ADL2, and Autosar compliant architecture description language," *European Congress on Embedded Real-Time Software. Toulouse, France*, 2008.

[9] M. Broy, "Challenges in automotive software engineering," in *ICSE '06: Proc. of the 28th Int. conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 33–42.

[10] H. Dörr, "The AUTOSAR Way of Model-Based Engineering of Automotive Systems," in *ICGT '08: Proceedings of the 4th international conference on Graph Transformations*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 38–38.

[11] O. Kindel and M. Friedrich, *Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis*. Heidelberg: dpunkt, 2009.

[12] F. Ougier and F. Terrier, "Eclipse based architecture of the EDONA platform for automotive system development," *Embedded Real Time Software and Systems*, 2010.

[13] A. Albinet, L. Quéran, B. Sanchez, and Y. Tanguy, "Requirement management from System modeling to AUTOSAR SW Components," *Emb. Real Time Software and Systems*, 2010.

[14] G. Karsai, A. Lang, and S. Neema, "Design patterns for open tool integration," *Software and System Modeling*, vol. 4, no. 2, pp. 157–170, 2005.

[15] M. Biehl, C.-J. Sjöstedt, and M. Törngren, "A Modular Tool Integration Approach - Experiences from two Case Studies," in *3rd Workshop on Model-Driven Tool & Process Integration (MDTPI 2010) at the European Conference on Modeling Foundations and Applications (ECMFA 2010)*, Jun. 2010.

[16] U. Honekamp, "The Autosar XML Schema and Its Relevance for Autosar Tools," *IEEE Software*, vol. 26, pp. 73–76, 2009.

[17] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003.

[18] J. van Wijngaarden and E. Visser, "Program Transformation Mechanics," http://igitur-archive.library.uu.nl/math/2007-1123-200750/wijngaarden_03_programtransformation.pdf, Utrecht University, Tech. Rep., 2003.

[19] T. Mens and P. V. Gorp, "A Taxonomy of Model Transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, March 2006.

[20] K. Kim, H. Kim, and W. Kim, "Building Software Product Line from the Legacy Systems "Experience in the Digital Audio and Video Domain"," in *SPLC '07: Proc. of the 11th International Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 171–180.

[21] A. Leitner and C. Kreiner, "Managing ERP Configuration Variants: An Experience Report," in *Proc. of the 1st Workshop on Knowledge oriented product line engeenering*, 2010.

[22] A. L. Santos, K. Koskimies, and A. Lopes, "Automated domain-specific modeling languages for generating framework-based applications," in *SPLC*, 2008, pp. 149–158.

[23] O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding Standardized Variability to Domain Specific Languages," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, 2008, pp. 139 –148.

[24] AUTOSAR, *AUTOSAR Software Component Template*, 2009.

[25] ATESST2, "EAST-ADL Domain Model Specification," ATESST2 Consortium, Tech. Rep., 2010.

Requirement Identification for Variability Management in a Co-simulation Environment

Andrea Leitner
Inst. f. Tech. Informatics
Graz University of Techn., AT
andrea.leitner@tugraz.at

Philipp Toeglhofer
Inst. f. Tech. Informatics
Graz University of Techn., AT
philipp.toeglhofer@student.tugraz.at

Josef Zehetner
Virtual Vehicle Competence Center, Austria
josef.zehetner@v2c2.com

Daniel Watzenig
Virtual Vehicle Competence Center, Austria
daniel.watzenig@v2c2.com

ABSTRACT

Co-simulation is a powerful approach to verify a system design and to support concept decisions early in the automotive development process. Due to the heterogeneous nature of the co-simulation framework there is a lot of potential for variability requiring the systematic handling of it.

We identified two main scenarios for variability management techniques in a co-simulation environment. Variability management capabilities can be included in the co-simulation tool itself or provide variability mechanisms to configure the co-simulation externally from a software product line. Depending on the context, one or even both scenarios can be applied.

This work addresses different types of variability in an independent co-simulation framework (ICOS) and defines requirements for a realization concept.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Domain engineering

General Terms

DESIGN

Keywords

Software product line engineering, Co-Simulation, Variability management,

1. MOTIVATION

Software in the automotive domain is highly complex, multi-functional, distributed, real-time and safety-critical. Further, the domain profile is traditionally vertically organized, resulting in individually developed parts. To integrate different parts in an early development stage, different models should be simulated within a holistic model of the system. The need for co-simulation is, for example, stated in [1].

Hardung et al. [2] highlight the importance for reuse in the automotive development process. Dager [3] shows a successful experience for the development of a software product line architecture for real-time embedded diesel engine controls.

Following this, co-simulation and the support of reusability seem to be two integral aspects in the automotive development process. Therefore, we try to combine both strategies.

We motivate the explicit use of variability management techniques with different application scenarios identified in practice.

1.1 Application scenarios

The main goals of introducing variability management in co-simulation environments is to elevate systematic reuse of existing models and co-simulations and to make variability explicit.

We identified two different application scenarios:

1.1.1 Standalone variability management in co-simulation

Variability management capabilities can be standalone in the co-simulation environment in order to provide different scenarios.

- *Co-Simulation of different vehicle variants*
  The following examples show different sources of variability in a co-simulation project. First, variability can be caused by alternative mechanical components. If, for instance, a hybrid electric vehicle is sold with two different types of electric motors, both motors are represented in two distinct models. This means, that depending on the vehicle variant the proper choice of electric motor model has to be used for simulation. Another issue resulting in the same solution is the provision of models for the same component but with different

SPLC '2012, September 02 - 07, 2012, Salvador, Brazil

levels of granularity.

Second, the environment varies due to different markets. For instance, the simulation of the heat flow of a car that is sold on the Russian market might need to be simulated with completely different ambient temperatures than one that is only sold in Australia. This means that different models in the co-simulation project have to be initiated with different values. Additionally, a vehicle's thermo dynamical energy flow heavily depends on the engine [4]. Consequences of different engines (like in the example above) can be evaluated using co-simulation.

- *Optimization of co-simulation settings*
  Finding a step size that is a good trade-off between accuracy and simulation time is crucial for the success of co-simulation. If no appropriate step size is known in advance, trial-error or optimization can be done by varying the step size and evaluating the results.

1.1.2 Integration in an automotive software product line (SPL)

Co-simulation can also be used as part of an automotive software product line. A possible scenario is shown in [5]. Variability has to be handled in different development phases and artifacts (requirements, implementation, tests, etc.), respectively. The co-simulation project and all corresponding models can be seen as one such development artifact.

Depending on the context one of the application scenarios or maybe both are desirable. This paper motivates the introduction of variability management capabilities in a co-simulation environment. The remainder of the paper is organized as follows: Section 2 describes the technical background, especially the concept of co-simulation. Section 3 describes different types of variability occurring in such an environment. Section 4 defines the most important requirements for variability support in co-simulation. Section 5 lists current related literature and Section 7 finally concludes the paper.

2. TECHNICAL BACKGROUND

Simulation is a well-known approach to support the development of complex systems. Development time and costs can be saved if a model of the system is simulated before the real system is actually built. Simulation *"enables the study, analysis and evaluation of situations that would not be otherwise possible"* [6].

The increased interdependence of engineering disciplines led to the demand for the integration of several domain-specific models into a single simulation. This demand can be satisfied by the application of *co-simulation* [1].

The main task of co-simulation is the holistic simulation of an overall system to determine the global characteristics of the system. The overall system consists of several subsystems which are simulated in their domain-specific simulation tools. Thus, the co-simulation platform is responsible for assembling the subsystems by connecting their inputs and output and for ensuring the interaction of the subsystems [7]. Another application of co-simulation is the geographically distributed simulation as described in [8].

The different models in the co-simulation project are treated as black boxes. The only information known about the models are their interfaces (the names and data types of the input and output parameters).

2.1 ICOS

ICOS[1] is an independent co-simulation environment developed at the virtual vehicle competence center[2]. It enables cross-domain co-simulation for a wide range of engineering disciplines in the field of automotive engineering.

In the automotive industry many specific simulation tools have been used in the past. Most of these tools specialize on a single area or discipline of automotive engineering. Hence, there is very little support for a heterogeneous simulation environment which is inherent to the automotive industry.

Typical co-simulation platforms try to overcome this limitation by supporting coupling of various simulation tools.

ICOS supports the coupling of existing domain/area-specific simulation tools and models that were developed using these tools [7].

One of ICOS' main design goals was to create a design which separates the co-simulation platform and its coupling algorithms from the simulation tools that are part of the co-simulation environment. In other words, the co-simulation platform must be *independent* from the simulation tools it uses [4].

2.1.1 Parameter connections

Parameter connection information is required to exchange data between models. This information has to be provided by the user by connecting input and output parameters. Obviously two connected parameters need to have the same data type. Further is it possible to connect one output parameter to more than one input parameter [9].

2.1.2 The boundary condition server

The *boundary condition server* (BCS) is a component of the ICOS co-simulation platform that provides boundary conditions and initialization values for other simulation models [9]. For example a temperature model in KULI[3] might require the ambient temperature as input. The initial temperature as well as the change of the temperature in time can be provided using the BCS.

The BCS does only have output parameters. Thus, the values of these output parameters have to be configured before the co-simulation is started and the BCS cannot react on the output of any other model.

[1] http://vif.tugraz.at/en/products/icos/
[2] http://www.v2c2.at
[3] http://www.kuli.at/

Per definition every ICOS project contains exactly one boundary condition server. This does not imply any constraints as the number of output parameters for this unique instance is unlimited [9].

3. VARIABILITY IN A CO-SIMULATION ENVIRONMENT

For a better overview, we classify the types of variability of a co-simulation environment into:

- *Model-Related Variability*
- *Linking Variability*
- *Environment Variability*
- *Coupling Variability*

3.1 Model-Related Variability

As already stated in Section 1.1 there are two cases which require model-related variability. First, we mentioned the case of two different electric motors. Both can be part of some distinct variant of a hybrid electrical vehicle. Variability is introduced in order to be enable to simulate every variant of the hybrid electrical vehicle with the appropriate model of the electric motor. The second motivating example consists of different levels of modeling granularity. For some co-simulation scenarios it might be sufficient to have a rather high level model (faster) of a subsystem, while others need a more detailed (but slower) model of the same subsystem.

How can this kind of variability be established in a co-simulation environment:

- *Model substitution:* If both electric motors are represented in two distinct models, these models must be made substitutable. Obviously, in order for two or more models to be substitutable, their interfaces need to be compatible, i.e. the number of input and output parameter and their data types must match.
- *Model and simulation tool substitution:* In case the two electric motors are modeled in different simulation tools, the simulation tool has to be substitutable too. Independent co-simulation abstracts the use of different co-simulation tools, enabling the substitution of models including its simulation tool in a similar way as the substitution of models described before.
- *Variable models:* Models might exploit some kind of variability. This variability can be used to change the behavior of a model in the co-simulation. As we stated before, the models are black boxes but may provide some kind of configuration port to bind internal variability.

3.2 Parameter Connection Variability

Model input and output parameters have to be connected in some way. This connection between input and output models can be variable as well. This is particularly true if substitutable models do not provide equal interfaces (same number of parameters and matching data types). With parameter connection variability, the changes in the parameters of substitutable models can be handled.

Even in the absence of model substitution, variable links can be used to make the co-simulation variable. Take a single-input single-output (SISO) as an example: The SISO performs a calculation based on a single input value and provides the result of the calculation as output. There might be configurations of the co-simulation where the conversion is not desired. In this case the linking can be changed to skip this conversion/-calculation.

3.3 Environment Variability

An example for environment variability has also been provided in Section 1.1. This example requires the use of different ambient temperatures for different co-simulation scenarios.

For a realization, initialization values or boundary conditions can be used as input parameters for models. This kind of variability does not change the model itself.

3.4 Coupling Variability

Coupling is the process of simulation tool integration. It handles the data exchange between simulation tools, or more precisely between the subsystems that are simulated in these tools [10]. Apparently this is not a trivial task, especially if little or no knowledge about the coupled subsystems is available. When introducing variability to a co-simulation, it might be desirable to make settings, like the macro time steps or the coupling mode, variable. We identified two motivations for this case:

1. variability of the coupling settings can be used to adapt the resulting co-simulation (the product). For instance, some simulation tools might require sequential simulation. Therefore, all resulting co-simulations that include models for these simulation tools, need to run sequentially.
2. variability of coupling settings can be used to optimize these settings or to find an appropriate value.

4. REQUIREMENTS

Although most of the requirements are independent of any co-simulation platform, we also include ICOS specific requirements. This is motivated by the fact that some variability types relate to specifics of this tool. In the following the main requirements for the introduction of variability management in the ICOS co-simulation platform are discussed.

4.1 Independent Co-Simulation

ICOS is an independent co-simulation tool, i.e. the co-simulation platform is independent from the used simulation tools. We require variability management in ICOS to be independent too. The co-simulation and

its models can exist on its own without the presence of the variability model. This makes the approach usable for other co-simulation platforms as well.

REQUIREMENT 1. *(Independence): The variability model has to be independent from the simulation tools that are used within the co-simulation.*

4.2 Decoupled Variability Model

The variability model describes the variability of a co-simulation. Obviously, this description depends on the co-simulation and its models. Contrarily, the co-simulation and its subsystem models are required to be decoupled from the variability model. Therefore, references from the co-simulation project to the variability model must not exist. Furthermore, the co-simulation and its models may exist on their own without the presence of the variability model.

REQUIREMENT 2. *(Decoupled Variability Model): The co-simulation project and all its affiliated models are independent from the co-simulation variability model.*

4.3 Modifiers

A variant may have several impacts on a co-simulation project. To describe the different impacts we introduce the concept of modifiers. A variant consists of one or more modifiers. Each modifier describes exactly how a core co-simulation project is modified when a variant is selected. Thereby a modifier defines a single modification of the co-simulation project.

For instance, a variant "substitute model A" may consist of two modifiers. While the first modifier actually substitutes a model, as described in Section 4.3.1, the second modifier adapts a boundary condition value according to the specification of the substituted model.

In the following all types of modifiers are described in detail. Furthermore their relationship to the variability types of Section 3 (model-related, linking, coupling and environment variability) will be explored.

4.3.1 Model Substitution Modifier

Models which are part of a variable co-simulation should be substitutable. Thus, a model can be replaced by another compatible model (see Model-Related Variability Sec. 3.1).

REQUIREMENT 3. *(Model substitution): A model that is part of the variable co-simulation project can be substituted with another model that is either simulated in the same simulation tool or in another simulation tool.*

4.3.2 Parameter Connection Modifiers

An ICOS co-simulation project is only valid, if all input parameters are linked to exactly one output parameter. However, an output parameter can be linked to several input parameters. This constraint has to be fulfilled for each generated product.

REQUIREMENT 4. *(Variable Linking): Links between input and output parameters can be variable. Their variability has to be bound in a way that in every product each input parameter is linked to exactly one output parameter.*

4.3.3 Parameter Modifiers

Parameter names in the simulation tool and in ICOS are distinct. Considering the substitution of models, we need means to map differing model-internal parameter names to a single parameter name in ICOS.

REQUIREMENT 5. *(Mapping parameter names): Internal model parameter names need to be variable in order to be mapped to different names in substitutable models.*

4.3.4 Boundary Condition Modifiers

The BCS offers boundary conditions (e.g. the ambient temperature) over output parameters. An easy way to make boundary conditions variable is to offer different values on different output parameters and change the linking of those output parameters. However, this does not comply with *Requirement 2*, as the co-simulation project has be adapted to the variability model.

As a solution boundary condition modifiers are introduced. With the help of boundary condition modifiers the value of a boundary condition can vary in two ways:

- *Single value:* the boundary condition parameter takes a single, constant value
- *Multiple values:* the boundary condition parameter takes multiple, variable values
- *Range of values:* This is a special case of multiple values, where a start and an end value as well as a step size is defined. E.g. from 1 to 3 (step 0.5) results in 1, 1.5, 2, 2.5 and 3.

REQUIREMENT 6. *(Variable boundary conditions): Boundary conditions can be variable by changing its value to a single or multiple values without change of the parameter linking.*

The boundary condition modifier is connected to the model-related as well as the environment variation points.

4.4 Extensibility

There is a huge amount of possible parameters, properties or items that one might want to make variable. Any group of variation points might include some scenarios that are not covered by the requirements stated so far. Some examples:

- different model initialization files for different variants (model-related variation point)
- variable step sizes (coupling variation points)
- varying coupling mechanism (coupling variation points)

REQUIREMENT 7. *(Extensibility): The effort to implement new types of modifiers has to be relatively small and the skills needed should not require extensive training.*

5. RELATED WORK

Thiel et al. [11] list different challenges in automotive systems engineering and how software product lines can help to overcome these challenges. Another example of automotive software product lines is given in [12]. Co-simulation is another important means to handle a serious of challenges in automotive systems development. To the best of our knowledge there is no related literature introducing explicit variability management in co-simulation environments. Pretschner et al. [13] to not explicitly highlight the need for an integration of co-simulation and variability. Nevertheless, they state that there is a high number of tools involved in an automotive development process due to the heterogeneity of the domain, but there is a lack of continuous tool integration. Co-Simulation can be seen as one possible solution for this problem. On the other hand the authors also mention the demand for explicit variability management due to the need for mass customization and the different lifecycles of software, mechanical components and ECUs.

Karner et al. [14] describe runtime switching for models with various levels of detail in a HW/SW co-simulation environment. Nevertheless, these works only focus on switching alternative implementations, which is only one small aspect of our work. Furthermore, this solution cannot be used independent from the co-simulation environment. Zeller et al. [15] describe an approach for using co-simulation to simulate dynamic reconfiguration, namely reallocation of software functions to ECUs and activation/deactivation of specific functionalities. This can be used for example for optimization of allocation.

6. FUTURE WORK

In future work we will provide an implementation for the variability mechanisms identified in this work. They should be designed in a way that enables two scenarios. First, the binding of variability in a standalone version directly in the co-simulation tool environment and second, the connection to a feature model in a software product line context.

For the standalone implementation the variability of a co-simulation project can be described by means of a variability model in domain engineering. The variability information is used in application engineering to derive concrete co-simulation projects.

For the integration in a product line context, a connector for pure::variants[4] will be provided. In concrete, it should be possible to import the co-simulation variability model as a pure::variants Family Model which can be connected and controlled with a Feature Model.

The variability mechanisms will be validated on the example of a practical application.

[4]http://www.pure-systems.com/pure_variants.49.0.html

7. CONCLUSION

This work motivates the introduction of variability management capabilities in a co-simulation environment. There are two main scenarios justifying this work. First, variability management techniques can be used standalone in the co-simulation environment in order to simulate different scenarios or to optimize various configuration settings. Second, the co-simulation environment can be integrated in an automotive software product line. The main contribution of this work are variability scenarios and requirements for the implementation of variability mechanisms in a co-simulation environment.

Acknowledgment

The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

8. REFERENCES

[1] M. Geimer, T. Krüger, and P. Linsel, "Co-Simulation, gekoppelte Simulation oder Simulatorkopplung? Ein Versuch der Begriffsvereinheitlichung ," in *O+P Zeitschrift für Fluidtechnik (50)*, Wiesbaden, Germany, 2006, pp. 572–576.

[2] B. Hardung, T. Kölzow, and A. Krüger, "Reuse of software in distributed embedded automotive systems," in *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*. New York, NY, USA: ACM, 2004, pp. 203–210.

[3] J. Dager, "Cummins's experience in developing a software product line architecture for real-time embedded diesel engine controls," in *Proceedings of SPLC 2000*, 2000, pp. 23–46.

[4] W. Puntigam, "Coupled Simulation: key for a successful energy management," in *Virtual Vehicle Creation*, Stuttgart, Germany, Jun. 2007.

[5] A. Leitner, R. Mader, C. Kreiner, C. Steger, and R. Weiß, "A development methodology for variant-rich automotive software architectures," *Elektrotechnik und Informationstechnik*, vol. 128, no. 6, pp. 222–227, 2011.

[6] R. Shannon, "Introduction to the art and science of simulation," in *Simulation Conference Proceedings, 1998. Winter*, vol. 1, dec 1998, pp. 7 –14 vol.1.

[7] M. Benedikt, J. Zehetner, D. Watzenig, and J. Bernasch, "Modern coupling strategies - is co-simulation controllable? ," in *NAFEMS Seminar: The Role of CAE in System Simulation*, Wiesbaden, Germany, 2011.

[8] M. Faruque, M. Sloderbeck, M. Steurer, and V. Dinavahi, "Thermo-electric co-simulation on geographically distributed real-time simulators," in *Power Energy Society General Meeting, 2009. PES '09. IEEE*, july 2009, pp. 1 –7.

[9] V. V. C. Center, *ICOS User Manual version 2.0*, Virtual Vehicle Competence Center, Graz, Austria, Jan. 2012.

[10] M. Benedikt, H. Stippel, and D. Watzenig, "An adaptive coupling methodology for fast time-domain distributed heterogeneous co-simulation," in *SAE Technical Paper*, vol. 2010-01-0649, 2010.

[11] S. Thiel, M. A. Babar, and G. Botterweck, "Software product lines in automotive systems engineering," in *SAE International Journal of Passenger Cars- Electronic and Electrical Systems*, vol. 1, no. 1, 2009, pp. 531–543.

[12] C. Tischer, A. Müller, M. Ketterer, and L. Geyer, "Why does it take that long? establishing product lines in the automotive domain," in *Software Product Line Conference, 2007. SPLC 2007. 11th International*, sept. 2007, pp. 269 –274.

[13] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 55–71.

[14] M. Karner, E. Armengaud, C. Steger, and R. Weiss, "Holistic simulation of flexray networks by using run-time model switching," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 544 –549.

[15] M. Zeller, G. Weiss, D. Eilers, and R. Knorr, "Co-Simulation of Self-Adaptive Automotive Embedded Systems ," in *Proceedings of the 8th International Conference on Embedded and Ubiquitous Computing (EUC)*, 2010, pp. 73 – 80.

Bibliography

[1] R. Kremlicka, S. Mayer, G. Bittner, and G. Reich, "Megatrends in der Automobilindustrie und ihre Auswirkungen auf den AC Centrope ," tech. rep., Mobilitätscluster der Wirtschaftsagentur Wien, 2011.

[2] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering*. Springer, 2005.

[3] L. Northrop and P. Clements, *Software Product Lines: Practices and Patterns*. Addison Wesley, 2002.

[4] H. Dörr, "The AUTOSAR Way of Model-Based Engineering of Automotive Systems," in *ICGT '08: Proceedings of the 4th International Conference on Graph Transformations*, (Berlin, Heidelberg), pp. 38–38, Springer-Verlag, 2008.

[5] F. Ougier and F. Terrier, "EDONA: an Open Integration Platform for Automotive Systems Development Tools," tech. rep., Renault and CEA LIST, visited 2012.

[6] L. M. Northrop, "Software Product Line Adoption Roadmap," tech. rep., Software Engineering Institute, Carnegie Mellon University, 2004.

[7] N. Navet and F. Simonot-Lion, *Automotive Embedded Systems Handbook*. Boca Raton, FL, USA: CRC Press, Inc., 2008.

[8] B. Baeker, "Wiederverwendung von automotiver Software: Reifegradmodell, Technologie und Praxisbericht," in *Moderne Elektronik im Kraftfahrzeug IV*, pp. 204–219, 2009.

[9] K. Berg, J. Bishop, and D. Muthig, "Tracing software product line variability: from problem to solution space," in *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, SAICSIT '05, pp. 182–191, 2005.

[10] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques: Research Articles," *Softw. Pract. Exper.*, vol. 35, no. 8, pp. 705–754, 2005.

[11] C. W. Krueger, "Easing the transition to software mass customization," in *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, (London, UK), pp. 282–293, Springer-Verlag, 2002.

[12] J. Lee, K. C. Kang, and S. Kim, "A Feature-Based Approach to Product Line Production Planning," in *Proceedings of the third Software Product Line Conference*, pp. 183–196, 2004.

[13] K. Schmid, "A comprehensive product line scoping approach and its validation," in *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, (New York, NY, USA), pp. 593–603, ACM, 2002.

[14] F. J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Berlin: Springer, 2007.

[15] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," tech. rep., Carnegie-Mellon University Software Engineering Institute, November 1990.

[16] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.

[17] I. Jacobson, M. Griss, and P. Jonsson, *Software reuse: architecture process and organization for business success*. ACM Press, 1997.

[18] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. H. Obbink, and K. Pohl, "Variability Issues in Software Product Lines," in *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, (London, UK), pp. 13–21, Springer-Verlag, 2002.

[19] M. Jaring and J. Bosch, "A Taxonomy and Hierarchy of Variability Dependencies in Software Product Family Engineering," *Computer Software and Applications Conference, Annual International*, vol. 1, pp. 356–361, 2004.

[20] J. V. Gurp, J. Bosch, and M. Svahnberg, "On the Notion of Variability in Software Product Lines," in *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, (Washington, DC, USA), p. 45, IEEE Computer Society, 2001.

[21] C. W. Krueger, "Towards a Taxonomy for Software Product Lines," in *PFE '04: Revised Papers from the 5th International Workshop on Software Product-Family Engineering*, pp. 323–331, 2003.

[22] C. Fritsch, A. Lehn, D. T. Strohm, and R. B. Gmbh, "Evaluating Variability Implementation Mechanisms," in *in Proceedings of International Workshop on Product Line Engineering*, pp. 59–64, 2002.

[23] D. Beuche and J. Weiland, "Managing Flexibility: Modeling Binding-Times in Simulink," in *ECMDA-FA*, pp. 289–300, 2009.

[24] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Project Line Engineering," *IEEE Software*, vol. 19, no. 4, pp. 58–65, 2002.

[25] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *Proceedings of the International Workshop on Software Factories*, pp. 16–20, 2005.

[26] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, 2000.

[27] K. Czarnecki, "Overview of Generative Software Development," in *Proceedings of Unconventional Programming Paradigms (UPP)*, pp. 326–341, 2004.

[28] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[29] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computer Surveys*, vol. 37, pp. 316–344, December 2005.

[30] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, no. 2, pp. 199–220, 1993.

[31] X. Peng, W. Zhao, Y. Xue, and Y. Wu, "Ontology-Based Feature Modeling and Application-Oriented Tailoring," in *Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components*, pp. 87–100, 2006.

[32] R. A. Falbo, G. Guizzardi, and K. C. Duarte, "An ontological approach to domain engineering," in *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, (New York, NY, USA), pp. 351–358, ACM, 2002.

[33] K. Czarnecki, C. H. Peter Kim, and K. T. Kalleberg, "Feature Models as Views on Ontologies," in *SPLC '06: Proc. of the 10th Int. on Software Product Line Conference*, (Washington, DC, USA), pp. 41–51, IEEE Computer Society, 2006.

[34] V. B. Matcha, P. Reddy, C. Hari, G. Srinivas, and N. SanjeevaRao, "Software Reuse: Ontological Approach to Feature Modeling," in *IJCSNS International Journal of Computer Science and Network Security*, pp. 262–268, 2009.

[35] O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding Standardized Variability to Domain Specific Languages," in *Proc. of the 12th International Software Product Line Conference*, pp. 139 –148, 2008.

[36] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski, "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, (New York, NY, USA), pp. 173–182, ACM, 2012.

[37] D. M. Weiss, J. J. Li, H. Slye, T. Dinh-Trong, and H. Sun, "Decision-Model-Based Code Generation for SPLE," in *Proceedings of the 12th International Software Product Line Conference*, pp. 129 –138, 2012.

[38] D. Dhungana, P. Grünbacher, and R. Rabiser, "The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study," *Automated Software Engineering*, vol. 18, pp. 77–114, Mar. 2011.

[39] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat, "Consistent product line configuration across file type and product line boundaries," in *Proc. of the 14th international conf. on SPL: going beyond*, SPLC'10, (Berlin, Heidelberg), pp. 181–195, Springer-Verlag, 2010.

[40] G. Holl, D. Thaller, P. Grünbacher, and C. Elsner, "Managing emerging configuration dependencies in multi product lines," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, (New York, NY, USA), pp. 3–10, ACM, 2012.

[41] M. Rosenmüller, N. Siegmund, T. Thüm, and G. Saake, "Multi-dimensional variability modeling," in *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, Fifth International Workshop on Variability Modelling of Software-Intensive Systems, (New York, NY, USA), pp. 11–20, ACM, 2011.

[42] M. Rosenmüller and N. Siegmund, "Automating the configuration of multi software product lines," in *Fourth International Workshop on Variability Modelling of Software-Intensive Systems*, pp. 123–130, 2010.

[43] M. Rosenmüller, N. Siegmund, C. Kästner, and S. S. ur Rahman, "Modeling Dependent Software Product Lines," in *Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE)*, pp. 13–18, Department of Informatics and Mathematics, University of Passau, Oct. 2008.

[44] W. Friess, J. Sincero, and W. Schroeder-Preikschat, "Modelling compositions of modular embedded software product lines," in *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*, (Anaheim, CA, USA), pp. 224–228, ACTA Press, 2007.

[45] F. Heidenreich and C. Wende, "Bridging the Gap Between Features and Models," in *2nd Workshop on Aspect-Oriented Product Line Engineering (AOPLE'07) co-located with the International Conference on Generative Programming and Component Engineering (GPCE'07)*, 2007.

[46] F. Heidenreich, P. Sanchez, J. Santos, S. Zschaler, M. Alferez, J. Araujo, L. Fuentes, U. K. amd Ana Moreira, and A. Rashid, "Relating feature models to other models of a software product line: A comparative study of featuremapper and vml*," *Transactions on Aspect-Oriented Software Development VII, Special Issue on A Common Case Study for Aspect-Oriented Modeling*, vol. 6210, pp. 69–114, 2010.

[47] H. Dubois, V. Ibanez, C. Lopez, J. Machrouh, N. Meledo, P. Mouy, and A. Silva, "The product line engineering approach in a model-driven process," in *Embedded real time software and systems 2012*, 2012.

[48] E. Kleinod, "Modellbasierte Systementwicklung in der Automobilindustrie - Das MOSES-Projekt," tech. rep., Fraunhofer Institut Software- und Systemtechnik, visited 2010.

[49] M. Große-Rhode, S. Euringer, E. Kleinod, and S. Mann, "Rough Draft of VEIA Reference Process," tech. rep., Fraunhofer Institut Software- und Systemtechnik, 2007.

[50] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2003.

[51] M. Ehsani, Y. Gao, and A. Emadi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design*. CRC Press, 2 ed., 2010.

[52] N. Kajtazovic, "Evaluation of variant management capabilities of automotive software engineering tools," Master's thesis, Graz University of Technology, 2011.

[53] N. Pavlidis, "Design and Implementation of a Variant Rich Component Model for Model Driven Development," Master's thesis, Graz University of Technology, 2011.

[54] W. Raschke, "Handling Variability in Unit Testing of Automotive Control Software," Master's thesis, Graz University of Technology, 2012.

[55] M. Voelter, "A Catalog of Patterns for Program Generation," tech. rep., Voelter - Ingenieurbüro für Softwaretechnologie, 2003.