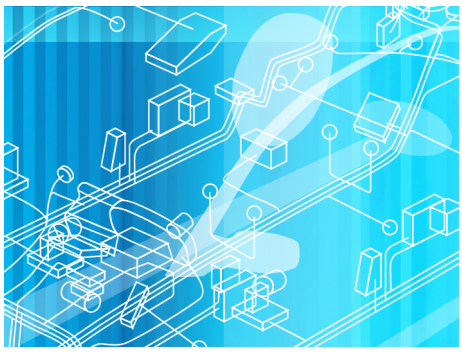
Fraunhofer Institut
Software- und
Systemtechnik

Architekturzentriertes Variantenmanagement für eingebettete Systeme

Ergebnisse des Projekts »Verteilte Entwicklung und Integration von Automotive-Produktlinien«

Martin Große-Rhode



ISST-Bericht 89/08
Oktober 2008

Herausgeber: Fraunhofer-Gesellschaft e. V.
Institut für Software- und Systemtechnik
Leitung: Prof. Dr. Jakob Rehof
Institutsteil Berlin: Mollstraße 1
10178 Berlin
Institutsteil Dortmund: Joseph-von-Fraunhofer-Straße 20
44227 Dortmund

ISSN 0943-1624

Das Projekt VEIA

Das Projekt »Verteilte Entwicklung und Integration von Automotive-Produktlinien« wurde vom Bundesministerium für Bildung und Forschung im Rahmen der Forschungsoffensive »Software Engineering 2006« unter dem Förderkennzeichen »01ISF15A« gefördert.

Am Projekt waren vier Partner beteiligt:

- BMW Group
- Fraunhofer ISST
- PROSTEP IMP GmbH
- Technische Universität München

Webseite: http://veia.isst.fraunhofer.de/

Autoren

Dieses Dokument wurde geschrieben von:

Martin Große-Rhode (Fraunhofer ISST).

Inhalt

Zusammenfassung

Automotive-Systeme müssen eine Vielzahl funktionaler und technischer Varianten unterstützen. Dazu muss im Entwicklungsprozess frühzeitig entschieden werden, ob unterschiedliche Varianten durch eine generische oder durch mehrere spezifische Softwarelösungen implementiert werden sollen. Im Projekt VEIA wurden Beschreibungsmittel, Methoden und der Werkzeugprototyp *v.control* für die Beantwortung dieser Frage erstellt.

In *v.control* werden funktionale Anforderungen und Softwarelösungen durch Architekturmodelle dargestellt. Deren Elemente können variabel sein, d. h. die Beschreibungssprache erlaubt die Kennzeichnung von Optionen, Alternativen und Parametern. Durch die Verbindung mit Modellen der charakteristischen Eigenschaften der Systemvarianten (Feature-Modellen) können ausführbare und AUTOSAR-konforme Modelle generiert werden. Zur Bewertung von Entwurfsalternativen für die unterschiedlichen Softwarelösungen wurden Metriken definiert und implementiert. Damit liefert der Prototyp nicht nur den Machbarkeitsnachweis für die im Projekt entwickelte Methode, sondern zeigt auch den objektiven Nutzen der Erstellung von VEIA-Modellen für den Entwicklungsprozess.

1 Einleitung

Keine zwei Fahrzeuge sind gleich. Jeder Käufer erwartet umfassende Auswahlmöglichkeiten, um sein Fahrzeug selbst konfigurieren zu können, und für die unterschiedlichen Absatzmärkte sind jeweils spezifische Gesetze und Vorschriften zu erfüllen. Zu den externen kommen die internen, in der Prozesskette liegenden Gründe für weitere Varianten hinzu: Komponenten unterschiedlicher Zulieferer sind zu unterstützen und verschiedene technische Infrastrukturen zu bedienen. Innerhalb des Lebenszyklus einer Fahrzeugbaureihe ändern sich die Randbedingungen, Komponenten sind in der ursprünglichen Form nicht mehr verfügbar oder kommen von neuen Zulieferern, interne Richtlinien und technische Infrastrukturen ändern sich. Darüber hinaus sollen weitere Fahrzeugvarianten später produziert und in den Markt eingeführt werden, die aber bereits bei der Entwicklung der ersten Generation als weitere Varianten mitberücksichtigt werden müssen.

So individuell wie die Fahrzeuge sind auch die Elektrik-, Elektronik- und Softwaresysteme, die innerhalb des Produktentstehungsprozesses für die Fahrzeuge entwickelt werden. Es ist offensichtlich, dass nicht für jedes inidviduelle E/E-System ein eigener Entwicklungsprozess aufgesetzt werden kann. Vielmehr muss die Variantenvielfalt innerhalb einer Baureihe oder Produktlinie, oder auch über Baureihen hinweg, in *einem* Prozess geplant und umgesetzt werden.

Zumindest in der industriell etablierten Softwaretechnik gehört die durchgängige Entwicklung variantenreicher Systeme innnerhalb eines Prozesses – von der ersten Konzeption und Anforderungserfassung bis zur Produktion, Wartung und Weiterentwicklung einer Systemfamilie – aber noch nicht zum Standardrepertoire. Die Software-Produktlinientechnik liefert konzeptionelle Ansätze und zum Teil auch technische Umsetzungen in Form von Methoden und Werkzeugen. Für ihren umfassenden praktischen Einsatz im Bereich der Entwicklung von Automotive-Systemen (d. h. der Entwicklung von eingebetteten Systemen im Automobil) ist aber immer noch eine Reihe von Fragen zu beantworten. Sie reichen von den zum Teil zu stark idealisierenden Annahmen in der Konzeption über Details der werkzeugtechnischen Unterstützung bis zur organisatorischen Umsetzbarkeit und – nicht zuletzt – dem nachweisbaren Nutzen der Methode.

In diesem Aufgabenbereich hat sich das Projekt VEIA das Ziel gesetzt, *auf der Grundlage der Konzepte der Produktlinientechnik eine Methode für die verteilte Entwicklung und Integration von Automotive-Systemen zu erarbeiten, die sich an den konkreten Anforderungen industrieller Entwicklungsprozesse orientiert und*

praktisch anwendbar ist.[1] Zum Ende des Projekts kann das erreichte Ziel konkreter als Antwort auf die folgenden Fragen formuliert werden.

> *Wie lässt sich die Entwicklung variantenreicher Softwaresysteme auf der Architekturebene adressieren und in die Anforderungs- und Entwurfsphasen eines funktionsorientierten Prozesses integrieren?*
>
> *Wie lässt sich dieses Variantenmanagement einer AUTOSAR-konformen Entwicklung vorschalten?*

Die Beschreibungsmittel, Methoden und Werkzeuge, die eine konstruktive Antwort auf diese Frage geben, wurden im Projekt erstellt.

1.1 Projektergebnisse

Zur Beantwortung der Fragen wurden folgende Resultate erarbeitet:

1. ein Referenzprozess, der angibt, welche Modelle vor den Modellen der Lösungen bzw. AUTOSAR-Beschreibungen entwickelt werden sollen und wie sie miteinander in Beziehung gesetzt werden [GEKM07],
2. eine Architekturbeschreibungssprache für unterschiedliche Sichten auf variantenreiche eingebettete Systeme mit einer AUTOSAR-Schnittstelle [GKM07c],
3. ein Leitfaden zur Entwicklung der Modelle [Gro08, Kle08], sowie
4. Metriken zur Bewertung der Modelle bzw. der durch sie repräsentierten Systementwürfe [FLÖR07, GKM07b, Man08a].

Zur Erstellung, Verwaltung, Analyse und Bewertung der Modelle wurde ein Werkzeug implementiert, mit dem ein vollständiger Durchgang durch den Referenzprozess demonstriert werden kann.

[1] Zitat aus dem Projektantrag

1.2 Fallstudie

Um die Projektergebnisse bewerten zu können, wurde eine durch Software zu unterstützende Fahrzeugfunktion aus der aktuellen Serienentwicklung bei der BMW Group als Fallstudie definiert. Der *Condition Based Service* (CBS) empfiehlt dem Fahrer auf der Grundlage des Fahrzeugzustands Servicetermine. Diese Funktion ersetzt die statisch vorgegebenen Serviceintervalle und bietet damit dem Fahrzeughalter einen erhöhten Komfort.

Die Fallstudie CBS weist folgende, für eine Übertragung der Projektergebnisse charakteristischen Merkmale auf:

1 CBS ist eine Mehrwertfunktion, d. h. sie verwendet Informationen, die im Fahrzeug bereits vorhanden sind – die Informationen über den Zustand verschiedener Verschleißteile im Fahrzeug – für die Ableitung eines zusätzlichen Nutzens. Das bedeutet, dass die Methode insbesondere auf Entwicklungen angewendet werden kann, die ein bestehendes System erweitern.

2 Da die Zustände verschiedener Verschleißteile in die Berechnung eingehen, ist die Funktion CBS auch auf die Steuergeräte *verteilt*, in denen die Zustandsinformationen erfasst werden.

3 CBS soll in allen Fahrzeugen eingesetzt werden. Dementsprechend sind alle möglichen Kombinationen von Verschleißteilen, Ausstattungsvarianten und Hardwareplattformen zu unterstützen, d. h. CBS ist eine *variantenreiche* Funktion.

4 CBS soll als *Software-Baustein* erstellt werden, d. h. die Software ist so zu strukturieren, dass ein CBS-Baustein in jedes Fahrzeugsystem eingesetzt werden kann und sowohl mit der verteilten Zustandserfassung als auch der Anzeige des jeweiligen Fahrzeugs zusammenspielt.

Mit der Fallstudie soll insbesondere der Nutzen des im Projekt entwickelten Referenzprozesses bzw. der (zusätzlich) zu erledigenden Modellierungsaufgaben belegt werden. Um diesen Beleg zu erbringen, wurde folgendes Szenario definiert:

Für die Entwicklung der CBS-Software bieten sich zwei Alternativen an:

1 Es wird eine generische Lösung erstellt, die alle Varianten abdeckt. Zur Ausführung im Fahrzeug kommen dann immer nur die Anteile, die in diesem Fahrzeug benötigt werden. (150%-Lösung: die Software kann mehr als ihre individuelle Verwendung verlangt, da sie die Summe aller Verwendungen implementiert.)

2 *Es wird ein Mechanismus implementiert, mit dem für jedes Fahrzeug die Software generiert werden kann, die spezifisch für dieses Fahrzeug benötigt wird. (Spezifische Lösung: die Software kann genau das, was in der individuellen Verwendung verlangt wird. Es ist aber ein Generierungsmechanismus zu implementieren.)*

Welche der beiden Lösungsalternativen ist besser?

Mit den im Projekt erarbeiteten Mitteln kann diese Frage objektiv, nachvollziehbar und wiederholbar beantwortet werden.

Die Darstellung der Projektergebnisse in diesem Bericht ist an einem Durchlauf durch den Referenzprozess orientiert, der auf die Beantwortung der Zielfrage des Szenarios hinausläuft. In Kapitel 2 wird erläutert, wie Varianz in den Architekturmodellen, die die funktionalen Anforderungen und die softwaretechnischen Lösungsalternativen repräsentieren, dargestellt wird. Der Mechanismus, mit dem aus einem Modell eines variantenreichen Systems ein (produkt-)spezifisches Modell ausgeleitet wird, das keine Varianz mehr enthält, wird in Kapitel 3 vorgestellt. Die Bewertung von Lösungsalternativen durch die Anwendung von Metriken auf die Architekturmodelle wird in Kapitel 4 diskutiert. In Kapitel 5 werden die Ergebnisse zusammengefasst sowie Anschlussmöglichkeiten und weitere Entwicklungen skizziert.

2 Darstellung von Varianz in Architekturmodellen

Sowohl die funktionalen Anforderungen als auch die Software- und Systemlösungen werden im VEIA-Referenzprozess durch Architekturmodelle dargestellt. Die dazu verwendeten Konzepte orientieren sich an denen bekannter Architekturbeschreibungssprachen; in Bezug auf AUTOSAR stellen sie eine vereinfachte Version der AUTOSAR-Softwarebeschreibungen bzw. der AUTOSAR-Steuergeräte- und -Systembeschreibungen dar.

2.1 Architekturbeschreibung

Funktionale Einheiten werden als Komponenten dargestellt, die hierarchisch strukturiert werden können. Komponenten haben Ports; jeder Port ist mit einer Schnittstelle assoziiert, in der die Datenelemente, d. h. Signale oder Operationsaufrufe deklariert sind, die über diesen Port kommuniziert werden können. An einem Eingangsport können Datenelemente empfangen, an einem Ausgangsport verschickt werden. Zur Repräsentation von Komponentenzuständen werden (abstrakte) Zustandsvariablen verwendet.

Innerhalb einer gemeinsamen Oberkomponente können Komponenten miteinander durch Konnektoren verbunden werden. Jeder Konnektor verbindet einen Ausgangsport als Quelle eines Datenelements mit einem Eingangsport als Ziel des Datenelements. Dabei können verschiedene Konformitäts- bzw. Vollständigkeitsregeln überprüft werden:

1 Die Schnittstellen des Ausgangs- und Eingangsports stimmen überein.

2 i Zu jedem Signal, das an einem Eingangsport erwartet wird, gibt es einen Konnektor zu einem Ausgangsport, der dieses Signal liefert, d. h. alle erwarteten Informationen werden auch geliefert.

ii Zu jedem Operationsaufruf, der an einem Ausgangsport versendet werden kann, gibt es einen Konnektor zu einem Eingangsport, der diesen Operationsaufruf empfangen kann, d. h. alle erwarteten Dienste werden auch angeboten.

3 (Dual zu 2)

i Zu jedem Signal, das an einem Ausgangsport versendet werden kann, gibt es einen Konnektor zu einem Eingangsport, der dieses Signal empfangen kann. d. h. alle gelieferten Informationen werden auch verwendet,

ii Zu jedem Operationsaufruf, der an einem Eingangsport empfangen werden kann, gibt es eine Verbindung zu einem Ausgangsport, an dem dieser Operationsaufruf versendet werden kann, d. h. alle angebotenen Dienste werden auch verwendet.

Stimmen die Schnittstellen des Ausgangs- und Eingangsports, die durch einen Konnektor verbunden sind, nicht miteinander überein, so wirkt der Konnektor als Filter. Das heißt, es werden genau diejenigen Datenelemente übertragen, die am Ausgangsport *und* am Eingangsport deklariert sind.

Die Verbindungen zwischen einer Oberkomponente und deren Unterkomponenten werden durch Delegationen dargestellt. Wie Konnektoren verbinden sie Ports miteinander; allerdings verbindet eine Delegation stets einen Eingangsport der Oberkomponente mit einem Eingangsport der Unterkomponente *oder* einen Ausgangsport der Unterkomponente mit einem Ausgangsport der Oberkomponente. Die Konformitäts- und Vollständigkeitsregeln für Delegationen ergeben sich aus denen für Konnektoren: Bei einer Delegation zwischen Eingangsports ist der Eingangsport der Oberkomponente die Quelle und der Eingangsport der Unterkomponente das Ziel; bei einer Delegation zwischen Ausgangsports ist es umgekehrt.

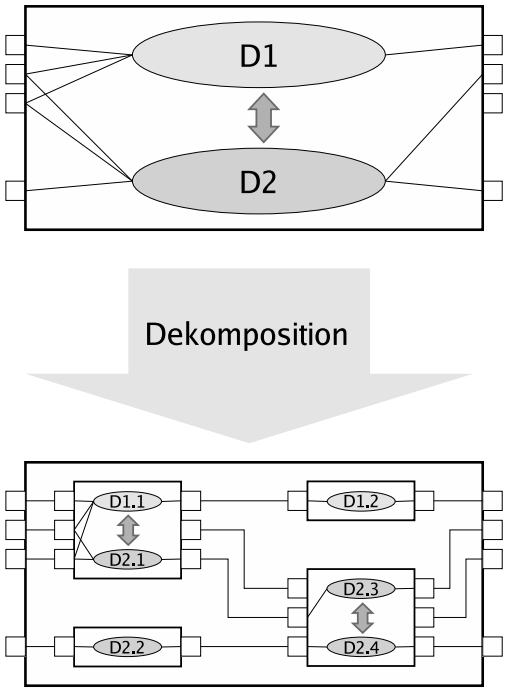
Durch die Komponenten, Ports und Schnittstellen wird die Struktur des Systems dargestellt. Das Verhalten, d. h. die (kausalen und temporalen) Beziehung zwischen den Eingangs- und Ausgangsdatenelementen einer Komponente bzw. eines Systems, wird durch Dienste repräsentiert. Ein Dienst ist mit einer Menge von Eingangs- und Ausgangsports assoziiert sowie mit einer Verhaltensbeschreibung versehen. Darüber hinaus wird für jeden Dienst angegeben, wodurch er ausgelöst wird (Eingang eines Datenelements oder Erreichen eines Zeitpunkts) und auf welche Zustandsvariablen er lesend oder schreibend zugreift. Die Verhaltensbeschreibung kann informell sein, z. B. eine natürlichsprachliche Use-Case-Beschreibung, oder formal, z. B. ein Statechart oder Code. Da die Verhaltensbeschreibungen auf die in der Strukturbeschreibung deklarierten Elemente zurückgreifen, kann die Konsistenz der beiden Ebenen überprüft werden. Dies trifft auch auf informelle Beschreibungen zu, die zumindest syntaktisch überprüft werden können.[2]

Zwischen den Diensten einer Komponente oder eines Systems bestehen oft Abhängigkeiten. Zum Beispiel kann ein Dienst einen anderen abbrechen, zeitweilig unterbrechen oder sein Ergebnis beeinflussen. Solche Abhängigkeiten werden sowohl auf der Anforderungs- als auch der Lösungsebene erfasst (siehe [GHH07b, HH08]).

[2] Zur Verhaltensmodellierung mit Diensten siehe [Gro08].

Verhaltensbeschreibungen können analog zur hierarchischen Struktur der Komponenten dekomponiert werden, d. h. zu einem (abstrakten) Dienst der Oberkomponente gehört eine Menge von Diensten der Unterkomponenten, die den Dienst der Oberkomponenten gemeinsam implementieren. Dabei müssen die Deklarationen des übergeordneten Diensts (assoziierte Ports der Oberkomponente) konsistent zu denjenigen der Teildienste sein. Sowohl zwischen den Teildiensten eines Diensts als auch zwischen den Teildiensten unterschiedlicher Dienste kann es wiederum Abhängigkeiten geben (siehe Abbildung 1).

Abbildung 1 Dekomposition eines abstrakten Diensts.



Da ein Dienst eine Ursache-Wirkungs-Beziehung zwischen eingehenden und ausgehenden Datenelementen auf einer höheren Abstraktionsebene beschreibt, stellt die Dienstdekomposition die dazugehörigen Wirkketten auf den darunter liegenden Abstraktionsebenen dar. Im VEIA-Referenzprozess reichen die Abstraktionsebenen von der logischen Sicht auf ganze Fahrzeugfunktionen (von den Sensoren und Bedienelementen zu den Aktoren und Anzeigelementen) bis zu programmiersprachlichen Funktionen auf der Anwendungsebene, d. h. AUTOSAR *runnables* und deren *implementations* in atomaren Softwarekomponenten.

2.2 Schichten und Abstraktionsebenen

Analog zur Aufteilung in AUTOSAR und anderen Architekturbeschreibungssprachen für eingebettete Systeme werden in der Beschreibung eines E/E-Systems im VEIA-Referenzprozess zwei Schichten unterschieden: die Anwendung und die Infrastruktur. Die Infrastruktur wird durch die Systemtopologie, d. h. die Verbindung der Steuergeräte über Kommunikationstechnologien (Busse), die Schnittstellen und Ressourcen der Steuergeräte selbst, sowie die Basissoftware und die Laufzeitumgebungen (RTE) beschrieben. Die Anwendungsschicht enthält die Beschreibung der Softwarekomponenten, die, unabhängig von ihrer Verteilung auf die Infrastruktur, die angeforderte Funktionalität des Systems implementieren.

Die Entwicklung der Infrastrukturschicht wurde im Projekt VEIA nicht betrachtet. In erster Näherung kann eine Infrastruktur mit den oben erwähnten Architekturkonzepten dargestellt werden. Konkrete Beschreibungen können z. B. mit den entsprechenden AUTOSAR-Templates, d. h. den *ECU Descriptions* und *System Constraints* erstellt werden.

Innerhalb der Anwendungsschicht werden zwei Abstraktionsebenen unterschieden. Die

logische Architektur, eine informationslogische Beschreibung der angeforderten Funktionalität, und die

Softwarearchitektur, eine softwaretechnische Beschreibung der Lösungsarchitektur.

Für beide Beschreibungen (Arten von Modellen) wird die gleiche Architekturbeschreibungssprache verwendet; in der Lösungsbeschreibung werden gegebenfalls mehr oder feinere Details angegeben, z. B. konkrete Datentypen anstelle der abstrakten Datentypen des Anforderungsmodells. Da sämtliche Architekturbeschreibungskonzepte auch im *AUTOSAR Software Component Template* vorhanden sind, können die Modelle der Anwendungsschicht nach AUTOSAR transformiert werden.

Die Verbindungen zwischen dem Anforderungsmodell, d. h. der logischen Architektur, sowie dem Lösungsmodell, d. h. der Softwarearchitektur auf der einen Seite und dem Infrastrukturmodell des Systems auf der anderen Seite werden durch weitere Modelle beschrieben: die Verteilungs- (oder Allokations-)modelle. Sie definieren für die Lösungsebene, auf welchen Steuergeräten die Anwendungssoftwarekomponenten installiert und ausgeführt werden und über welche Kommunikationsmedien (Busse) die Kommunikation zwischen den Komponenten

abläuft. Für die Anforderungsebene werden entsprechende Verteilungsanforderungen (Constraints) beschrieben. Das Format (die Modellierungskonzepte) für die Darstellung von Verteilungsmodellen wurde(n) in VEIA nicht ausformuliert. Im Vorgängerprojekt MOSES [Kle06] wurden solche Konzepte detailliert; in AUTOSAR entsprechen sie den *System Constraints*.

2.3 Varianzkonzepte

Die Architekturkonzepte werden nun für die Darstellung funktionaler Varianten erweitert. Prinzipiell können Varianten in einer Architektur durch folgende Konzepte repräsentiert werden.

Alternative (xor) Ein Architekturelement ist in genau einer Ausprägung vorhanden, wobei die möglichen Ausprägungen durch eine Menge von Alternativen angegeben sind.

Auswahl (or) Ein Architekturelement ist in einer beliebigen Anzahl von Ausprägungen aus einer Auswahlmenge vorhanden.

Option (ja/nein) Ein Architekturelement ist vorhanden oder nicht.

Parameter Ein Architekturelement enthält einen Parameter, der einen von mehreren (vielen) möglichen Werten hat.

Das Auswahlkonzept (or) lässt sich auf das Optionskonzept zurückführen; Parameter sind ein Standardkonzept von Architekturbeschreibungssprachen (vgl. die AUTOSAR-Konfigurationsparameter). Beide Konzepte werden daher in diesem Bericht nicht weiter diskutiert.[3]

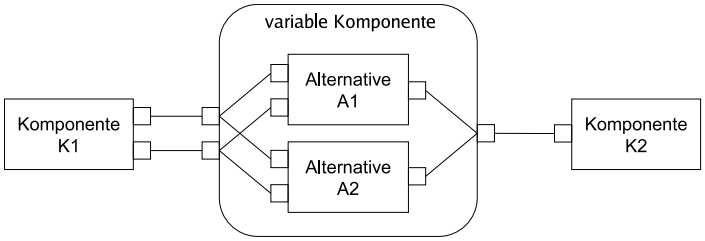
Die Varianzkonzepte können für Komponenten, Ports und Datenelemente verwendet werden. Für Komponenten können alle Konzepte verwendet werden (Alternative, Option, Parameter), für Ports und Datenelemente nur Optionalität und Parameter.

[3] Details sind in [Man08b] ausgeführt.

2.3.1 Integration der Varianzpunkte in die Architektur

Zur Darstellung von Komponentenalternativen wird die hierarchische Struktur von Architekturmodellen erweitert: Die Alternativen, aus denen die gewünschte Ausprägung ausgewählt wird, werden formal als Unterkomponenten einer gemeinsamen, abstrakten Oberkomponente dargestellt (siehe Abbildung 2). Die variable Komponente repräsentiert die Stelle in der Architektur, an der die ausgewählte Alternative eingesetzt wird. Sie wird auf ihrer Hierarchieebene durch Konnektoren in die Architektur eingebunden; Delegationen von den Ports der variablen Komponente zu denen der Alternativen geben an, wie eine Alternative eingebunden ist, wenn sie ausgewählt wird. Das Ersetzen einer variablen Komponente durch eine der zugehörigen Alternativen wird im nächsten Kapitel beschrieben.

Abbildung 2 Komponentenalternativen als Unterkomponenten einer variablen Komponente.



Optionale Komponenten gekennzeichnet. Innerhalb des Architekturmodells unterscheiden sie sich darüber hinaus nicht von anderen, nicht-variablen Komponenten. Die Bedeutung der Optionalität kommt erst bei der Konfiguration sowie der Komponentenstruktur zum Tragen. Optionale Ports und Datenelemente werden ebenso durch das Metaattribut *isOptional* gekennzeichnet.

2.3.2 Erweiterte Komponentenstruktur

Alternative Komponenten, d. h. Unterkomponenten einer variablen Oberkomponente, müssen nicht die gleichen Ports haben.[4] Die Ports der variablen Oberkomponente ergeben sich aus denjenigen der Alternativen. Sie repräsentieren die Spanne

[4] Dies ist einer der Hauptunterschiede der VEIA-Konzeption zu anderen Produktlinienansätzen; eine aus Anwendungssicht zwingend notwendige Erweiterung.

zwischen der minimalen und der maximalen Menge der Ports der Alternativen (siehe Abbildung 3):

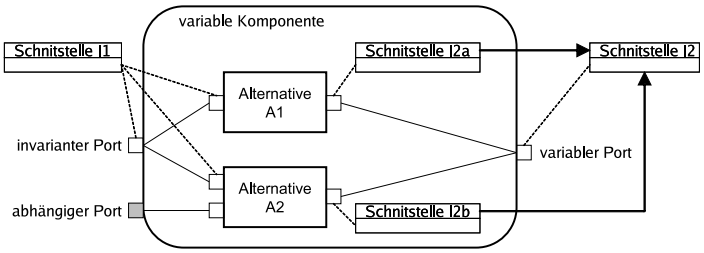
Invariante Ports Hat jede Alternative einen Port mit derselben Schnittstelle, so erhält die variable Komponente einen zugehörigen Port mit derselben Schnittstelle. Dieser invariante Port wird über Delegationen mit den zugehörigen Ports der Unterkomponenten (Alternativen) verbunden.

Variable Ports Hat jede Alternative einen Port mit ähnlicher Schnittstelle, so erhält die variable Komponente einen zugehörigen Port mit einer zu berechnenden Schnittstelle (siehe unten). Dieser variable Port wird ebenfalls über Delegationen mit den Ports der Unterkomponenten verbunden.

Abhängige Ports Hat eine oder haben mehrere Alternativen einen Port, zu dem es in mindestens einer anderen Alternative keine Entsprechung gibt, so erhält die variable Komponente einen abhängigen Port. Dessen Schnittstelle ist invariant oder variabel, je nachdem, ob die entsprechenden Ports dieselbe oder unterschiedliche Schnittstellen haben. Der abhängige Port der variablen Komponenten wird ebenfalls über Delegationen mit den entsprechenden Ports der Unterkomponenten verbunden.

Die invarianten Ports einer variablen Komponente können berechnet werden. Die Einrichtung eines variablen Ports hängt davon ab, ob die Schnittstellen der entsprechenden Ports an den Unterkomponenten als ähnlich erachtet werden; es ist also eine Entwurfsentscheidung. Im Extremfall können alle Ports von alternativen Komponenten, zu denen es nicht in allen anderen Alternativen entsprechende Ports mit gleichen Schnittstellen gibt, als abhängige Ports eingerichtet werden. Die Schnittstelle eines variablen Ports erhält als Schnittstelle die Vereinigung aller Schnittstellen der zugehörigen Ports der Unterkomponenten. Wenn ein Datenelement nicht für alle zugehörigen Ports deklariert ist, so wird es in der Schnittstelle des variablen Ports als abhängig gekennzeichnet.

Abbildung 3 Ports einer variablen Komponente.

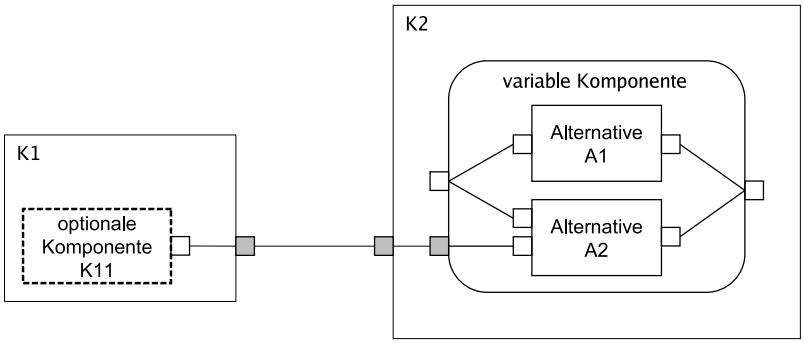


Abhängige Ports können auch durch optionale Unterkomponenten entstehen: Ein Port einer Oberkomponenten, der nur der Delegation an eine optionale Unterkomponente dient, ist abhängig von dieser Unterkomponente.

Mit diesem Konzept der invarianten, variablen und abhängigen Ports sowie der abhängigen Datenelemente bleibt die Information erhalten, dass auf einer unteren Hierarchieebene Varianz vorliegt; sie wird aber auf der höheren Hierarchiebene komprimiert dargestellt. Die volle Varianzinformation kann ausgeblendet und bei Bedarf sichtbar gemacht werden. Die genannten Portarten setzen sich durch die Hierarchie nach oben fort, d. h. die Delegation eines abhängigen Ports an eine Oberkomponente ist wiederum ein abhängiger Port, auch wenn die Oberkomponente selbst eine invariante Komponente ist.

Durch diese Darstellung können Abhängigkeiten zwischen Alternativen bzw. Optionen in einer Architektur analysiert werden (siehe Abbildung 4). Wenn eine alternative Komponente zum Beispiel einen Eingangsport enthält, der keine Entsprechung an einer anderen Alternative hat, dann können die Delegationen und Konnektoren, die zu diesem Eingangsport führen, zurückverfolgt werden. Stößt dieser Pfad an einer anderen Stelle in der Architektur wieder auf eine Alternative oder Option, so kann die Auswahl dieser Alternative oder Option durch die Auswahl an der Ausgangskomponente bereits festgelegt sein: Diejenige Alternative oder Option, die das benötigte Datenelement liefert, muss gewählt werden.

Abbildung 4 Abhängigkeit variabler Architekturelemente: Wenn die Alternative A1 gewählt wird, muss auch die Option K11 gewählt werden.



3 Konfiguration von Modellen

Ein Architekturmodell, das variable Elemente enthält, stellt nicht ein einzelnes Produkt (System), sondern eine Produktlinie (Systemfamilie) dar. Durch die Konfiguration der variablen Elemente kann aus dem Produktlinienmodell ein Produktmodell generiert werden, dass keine Varianz mehr enthält. Mit dieser Modellkonfiguration wird die Auflösung von Varianz zur Entwurfszeit im Prozess unterstützt. So können zum Beispiel aus einer Softwarearchitektur mit variablen Elementen produkspezifische Softwarearchitekturen generiert werden. Andere Varianzauflösungszeitpunkte (Compile-Zeit, Post-build, Bandende, Laufzeit usw.) wurden im Projekt VEIA nicht behandelt.

Varianzfreie Produktmodelle werden zum Beispiel benötigt, wenn Testfälle mit Testorakeln generiert werden sollen, oder wenn ein Modell ausgeführt, d. h. das dargestellte System simuliert werden soll. Auch für den AUTOSAR-Anschluss werden varianzfreie Modelle benötigt, da die entsprechenden Varianzkonzepte – alternative und optionale Komponenten sowie die Darstellung der Abhängigkeiten – im *AUTOSAR Software Component Template* nicht vorhanden sind.

3.1 Produktmerkmale

Die spezifischen Eigenschaften eines Produkts können sich an verschiedenen Stellen in der Architektur niederschlagen, auch innerhalb einer Architektursicht.

Um die Variantenvielfalt und die Modellkonfiguration handhaben zu können, werden neben den Architektursichten daher die charakteristischen Merkmale der Produkte – Gemeinsamkeiten, Unterschiede, Abhängigkeiten – in einem Merkmalmodell (*Feature Model*) erfasst. Dazu werden die Merkmale benannt, hierarchisch strukturiert und Alternativen und Optionen gekennzeichnet. Bezüglich der hierarchischen Strukturierung wurden keine Einschränkungen oder Vorgaben definiert. Sie dient in erster Linie der Übersichtlichkeit des Modells. Ein optionales Merkmal wird im Modell gekennzeichnet; es kann im Konfigurationsprozess gewählt oder abgewählt werden. Wenn es nicht gewählt wird, sind auch alle Merkmale, die sich in der Hierarchie darunter befinden, nicht gewählt. Zueinander alternative Merkmale werden unter einem gemeinsamen variablen Obermerkmal zusammengefasst; d. h. sie werden analog zur Darstellung von alternativen Komponenten in die hierarchische Struktur eingegliedert. Beim Konfigurationsprozess wird genau eines der alternativen Merkmale unter einem variablen Obermerkmal ausgewählt.

Mit jeder nicht gewählten Alternative entfallen auch alle in der Hierarchie darunter liegenden Merkmale. Obwohl die Auswahl mehrerer Merkmale aus einer Gruppe wie bei den Komponenten auf optionale Merkmale zurückgeführt werden kann, werden in den Merkmalmodellen *Kardinalitäten* zur Darstellung sowohl der Auswahl als auch der Alternative verwendet. Für die Auswahl werden die minimal und die maximal zu wählende Zahl von Merkmalen angegeben *[min,max]*. Für *min=1* und *max=1* ergibt sich die Alternative (xor): genau ein Merkmal muss ausgewählt werden.

Abhängigkeiten zwischen Merkmalen, die sich an beliebigen Stellen in der Hierarchie befinden können, werden durch die beiden Relationen *needs* (Implikation) und *excludes* (Exklusion) dargestellt. Darüber hinaus können logische Formeln zur Beschreibung komplexerer Abhängigkeiten verwendet werden.

3.2 Auswahl von Merkmalen

Um ein einzelnes Produkt innerhalb der Produktlinie zu charakterisieren, muss für jedes optionale Merkmal angegeben werden, ob es ausgewählt ist oder nicht, und für jedes variable Merkmal muss entsprechend der angegebenen Kardinalitäten eine Menge von Merkmalen bzw. genau ein Merkmal ausgewählt werden. Dieser Selektionsprozess kann von der Wurzel des Merkmalmodells absteigend bis zu den Blättern durchgeführt werden. Die Relationen schränken die Auswahlmöglichkeiten ein: Wenn ein Merkmal m_1 gewählt wurde, das ein Merkmal m_2 impliziert (m_1 *needs* m_2), dann muss auch m_2 gewählt werden. Wenn ein Merkmal m_1 gewählt wurde, das ein Merkmal m_2 ausschließt (m_1 *excludes* m_2), dann darf m_2 nicht gewählt werden.

Konsistente Merkmalselektionen können als Produktbeschreibungen (Produktkonfigurationen) gespeichert werden. Es können auch unvollständige Selektionen gespeichert werden, d. h. Selektionen, die noch optionale oder variable Merkmale enthalten. Diese beschreiben Teilproduktlinien.

3.3 Bindung von Architektur-Varianzpunkten

Mit einem Bindungsmodell wird dargestellt, welche Merkmale für die Ausprägung eines variablen Architekturelements ausschlaggebend sind. Wird ein Merkmal mit einem Architekturelement durch einen Link verbunden, dann bedeutet die Auswahl des Merkmals bei der Konfiguration, dass auch das Architekturelement

ausgewählt wird. Ein Merkmal kann mit mehreren Architekturelementen verbunden werden; ebenso kann ein Architekturelement mit mehreren Merkmalen verbunden werden.

Folgende Eigenschaften eines Bindungsmodells können zum Beispiel geprüft werden.

Vollständigkeit Jedes variable Architekturelement ist an mindestens ein Merkmal gebunden.

Angemessenheit Jedes variable Merkmal bindet mindestens ein Architekturelement.

Bei der Bindung müssen nur *primäre* Architekturelemente gebunden werden. Wenn zum Beispiel eine optionale Funktion an ein Merkmal gebunden ist und das Merkmal bei der Konfiguration nicht ausgewählt wird, dann entfallen mit der Funktion auch ihre Ports, Unterfunktionen, Variablen und Dienste. Diese müssen nicht mit dem Merkmal verbunden werden, weil sie Teile der Funktion sind. Entfällt ein Port, so fallen auch alle Konnektoren weg, die an dem Port hängen.

3.4 Extraktion eines Produktmodells

Das Merkmalmodell dient der Komplexitätsreduktion bei der Extraktion von Architekturmodellen für einzelne Produkte aus der Produktlinie. Die Auswahl findet im Merkmalmodell statt. Über das Bindungsmodell und die Abhängigkeitsregeln innerhalb des Architekturmodells wird die Merkmalselektion in das Architekturmodell übertragen und definiert dort die Selektion der produktspezifischen Architekturelemente. Da es im Architekturmodell in der Regel sehr viel mehr Elemente und Abhängigkeiten als im Merkmalmodell gibt, wird der Konfigurationsprozess für den Anwender einfacher.

Die Extraktion des produktspezifischen Architekturmodells wird im Folgenden negativ beschrieben, d. h. es wird dargestellt, welche Architekturelemente aus dem Produktlinienmodell entfernt werden. Ausgangspunkt der Ausleitung ist eine Produktkonfiguration, d. h. eine konsistente Teilmenge der Merkmale. Das Architekturmodell wird von der Wurzel aus durchlaufen, wobei Elemente gemäß der folgenden rekursiven Regeln entfernt werden.

– Wenn es zu einem optionalen oder alternativen Architekturelement kein Merkmal in der Produktkonfiguration gibt, das mit diesem Element verbunden ist, dann wird das Architekturelement entfernt.

– Wenn eine Komponente im Architekturmodell entfernt wird, dann werden alle ihre Kindelemente entfernt: Ports, Unterkomponenten, Konnektoren und Delegationen, Dienste, interne Variablen.

– Wenn ein Port entfernt wird, dann werden alle Konnektoren und alle Delegationen zur Oberkomponente an diesem Port entfernt.

– Wenn die letzte Delegation in einer Komponente zu einem Port dieser Komponente entfernt wurde, dann wird auch dieser Port entfernt.

Wenn dieser Prozess beendet ist, findet noch ein Bereinigungsschritt statt: Jede variable Komponente wird durch ihre (einzige) verbliebene Alternative (Unterkomponente) ersetzt. Dabei werden die Ports der Oberkomponente durch die entsprechenden Ports der Unterkomponente ersetzt und die dazwischen liegenden Delegationen entfernt.[5]

Mit Hilfe der Extraktion können weitere Eigenschaften der Varianzbindung definiert und geprüft werden.

Konsistenz In keinem Produktmodell ist mehr als eine Alternative einer variablen Komponente ausgewählt.[6]

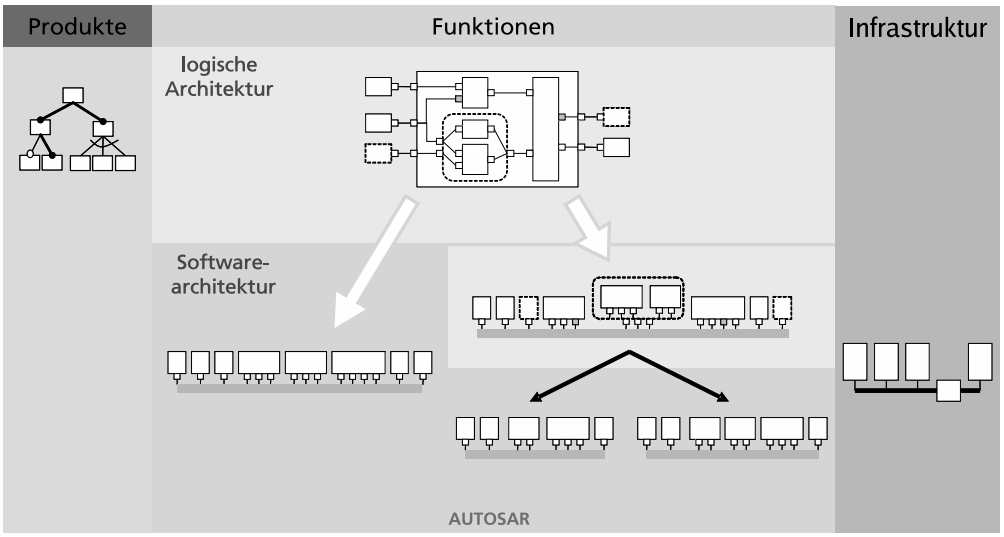
Verbindungsvollständigkeit In keinem Produktmodell gibt es unverbundene Ports. (Da bei der Extraktion eines Produktmodells evt. Konnektoren entfernt werden, kann es passieren, dass ein Port bestehen bleibt, aber nicht mehr verbunden ist.)

[5] Zum Konfigurationsprozess siehe auch [GHH07a].
[6] Zur Konsistenzprüfung auf der Produktlinienebene siehe auch [GLS08].

4 Bewertung von Entwurfsalternativen

Die im VEIA-Referenzprozess zu erstellenden Modelle sollen insbesondere dazu dienen, die Frage zu beantworten, welche System- und Prozesseigenschaften aufgrund des Architekturentwurfs zu erwarten sind, um alternative Entwürfe bewerten und miteinander vergleichen zu können.[7] Die Zielfrage der Fallstudie nach der besseren Lösung (siehe Abschnitt 1.2) ist in Abbildung 5 noch einmal im Kontext des Referenzprozesses dargestellt.

Abbildung 5 Bewertung unterschiedlicher Softwarelösungen.



Als Bewertungskriterien wurden im Projekt vornehmlich die Größe der Software, die für den Speicherbedarf und die Flashzeit relevant ist, und der Entwicklungsaufwand festgelegt. Als Ausgangspunkt wurde die für Software-Produktlinien erweiterte Funktionspunkt-Metrik von Kiebusch [Kie06, KBS06] gewählt. Nach entsprechenden Anpassungen der Metrik wurden die VEIA-Modelle um Attribute erweitert, in denen die für die Messung notwendigen weiteren Informationen wie die Echtzeitkategorie (hart, weich) und der Organisationsgrad bzgl. Produktlinientechnik erfasst werden. Andere Eigenschaften, wie die horizontale (gekapselte)

[7] Zur allgemeinen Herangehensweise siehe [GKM07a].

und vertikale (komponentenübergreifende) Variabilität von Komponenten, können unmittelbar aus den Modellen berechnet werden.

Da diese Metrik, wie die meisten anderen Architekturmetriken auch, nicht für hierarchische Modelle definiert ist, wurde eine weitere Modelloperation definiert und implementiert, die es erlaubt, ein Architekturmodell auf einen vom Nutzer zu bestimmenden Schnitt durch die Hieriarchie zu reduzieren. Darüber hinaus wurde die Definition der Metrik so erweitert, dass sie auch mit dem in VEIA entwickelten Konzept der Mehrfachverwendung von Komponenten vereinbar ist (siehe [GKM07b]).

Die Fallstudie wurde so weit entwickelt, dass sich ein erstes Ergebnis ablesen lässt: Für die Entwicklung des CBS-Clients ist es günstiger, eine generische Softwarelösung zu entwickeln. Dies liegt daran, dass die angeforderte Variabilität gut auf die anderen Komponenten verlagert werden kann. Betrachtet man die komplette Funktion CBS, von den Sensoren bis zur Anzeige, und gewichtet man die Software-Größe höher als den Entwicklungsaufwand, überwiegen die Vorteile eines produktspezifischen Lösungsmechanismus, d. h. einer Software-Produktlinie, deren Varianz zur Entwurfszeit aufgelöst wird.

5 Umsetzung und weitere Entwicklung

Die konzeptionellen Ergebnisse des Projekts wurden kontinuierlich in dem prototypischen Werkzeug *v.control* umgesetzt. Die Modelle werden in *v.control* als Bäume dargestellt; auf eine diagrammatische Darstellung wurde verzichtet, da es sich in erster Linie um eine Machbarkeitsstudie handelt.

In der Projektnavigationssicht (links in Abbildung 6) werden Merkmalmodelle, logische (Funktions-) und Softwarearchitekturen erzeugt und dargestellt. Umgesetzt wurde die Strukturansicht auf die Modelle; Verhaltensbeschreibungen von logischen Komponenten (Funktionen) und Softwarekomponenten sind noch nicht integriert. Die Eigenschaften der Modelle und Modellelemente können jeweils in speziellen Editoren dargestellt und bearbeitet werden.

Die Verbindungen zwischen den Modellen werden ebenfalls in der Projektnavigationssicht bzw. den Editoren angelegt. Mit dem Link-Viewer (siehe Abbildung 6) werden die Verbindungen angezeigt und ausgewertet: Zu jedem Merkmal können alle Funktionen und Softwarekomponenten, die an das Merkmal gebunden sind, angezeigt werden.

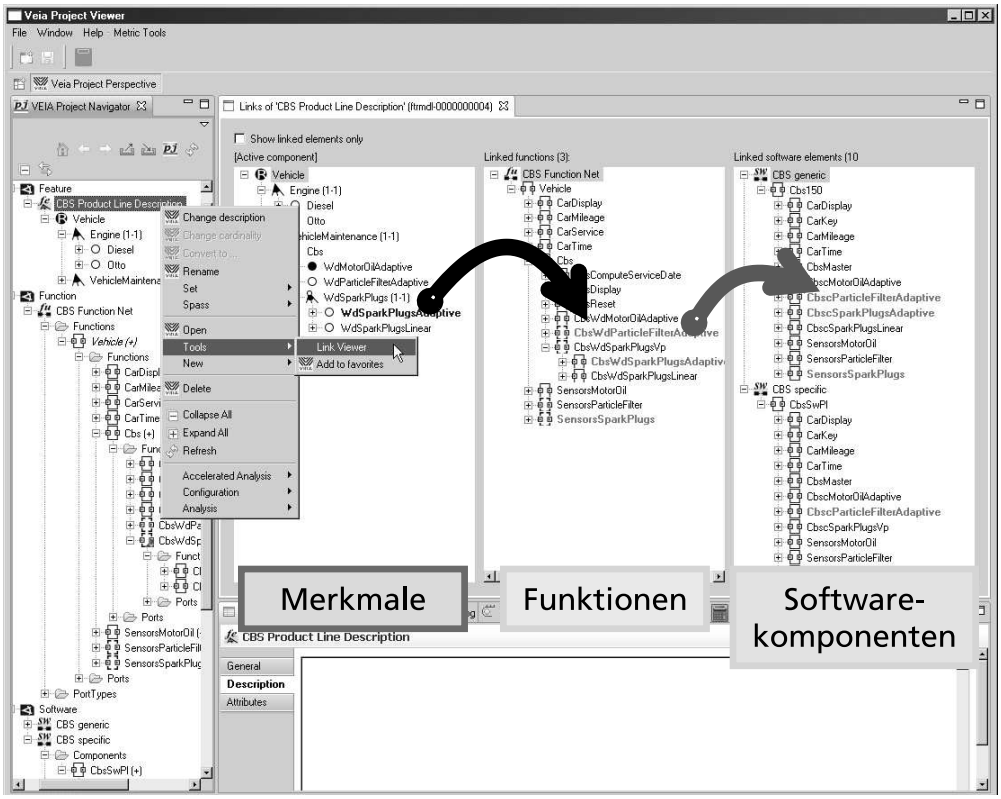
Innerhalb des Merkmalmodells können Produktkonfigurationen (konsistente Merkmalselektionen) erzeugt und gespeichert werden. Für diese Produktkonfigurationen können dann die entsprechenden Extrakte der Architekturmodelle (Ausleitungen des zugehörigen Funktionsnetzes und der Softwarearchitektur) erzeugt werden. Darüber hinaus sind verschiedene Analysemöglichkeiten für Merkmalmodelle implementiert, wie das Auffinden nicht selektierbarer Merkmale und die Berechnung der Anzahl aller möglichen Produktkonfigurationen.

Die in Kapitel 4 genannte Metrik zur Schätzung des Software-Größe und des Entwicklungsaufwands wurde implementiert. Darüber bietet *v.control* die Möglichkeit, weitere Attribute zu definieren, um andere Metriken anschließen zu können (siehe [Man08a]).

5.1 Weitere Entwicklung

Die im VEIA-Referenzprozess zu erstellenden Modelle können dazu verwendet werden, das Variantenmanagement in einem funktionsorientierten Prozess auf der Architekturebene zu unterstützen und in AUTOSAR-konforme Lösungsmodelle

Abbildung 6 Verbindung der Merkmale und Architektursichten im VEIA-Demonstrator.



abzubilden. Um den Übergang zu AUTOSAR auch werkzeugtechnisch besser zu integrieren, soll zusätzlich zu der bereits bestehenden AUTOSAR-Schnittstelle der Werkzeugprototyp *v.control* als AUTOSAR-Aufsatz weiterentwickelt werden. Um Interoperabilität mit anderen Werkzeugen zu ermöglichen, soll dazu die AUTOSAR-Tool-Plattform *Artop* verwendet werden. Eine Anbindung an MATLAB ist unabhängig davon bereits in Arbeit.

Die Modellierung von Varianten in der Infrastrukturschicht wurde im Projekt und im Werkzeug nicht explizit behandelt. Da die Konzepte für das Variantenmanagement und deren Integration in Architekturbeschreibungssprachen sehr generisch definiert wurden (siehe [Man08b]), können sie für die Infrastrukturschicht spezialisiert werden. Detaillierte Untersuchungen hierzu stehen allerdings noch aus. Insbesondere ist zu berücksichtigen, dass die Darstellung von Varianten über (Konfigurations-)Parameter in diesem Bereich eine sehr viel größere Rolle spielt als in der Anwendungsschicht.

Die methodische Kopplung von Struktur- und Verhaltensmodellen und die Verwendung unterschiedlicher Beschreibungsformate für das Verhalten wurden in [Gro08] exemplarisch gezeigt. Sowohl die Entwicklungsmethodik als auch die Analyse von Verhaltensvarianten und deren Abhängigkeit von Strukturvarianten müssen weiter systematisch untersucht werden, um zu tragfähigen Ergebnissen zu gelangen.

Insbesondere mit dem Werkzeug *v.control* ist aber bereits in der jetzigen Ausbaustufe gezeigt, dass Funktionsorientierung und Variantenmanagement auf der Architekturebene vereint und für den Systementwicklungsprozess nutzbar gemacht werden können.

Literatur

[FLÖR07] *Feldo*, Marek ; *Lindow*, Sven ; *Özhan*, Mesut ; *Rosenmüller*, Rainer: Bewertung des Echtzeit-Verhaltens von E/E-Systemen im Automobil. ISST-Bericht 81/07. http://veia.isst.fraunhofer.de/.

[GEKM07] *Große-Rhode*, Martin ; *Euringer*, Simon ; *Kleinod*, Ekkart ; *Mann*, Stefan: Grobentwurf des VEIA-Referenzprozesses. ISST-Bericht 80/07. http://veia.isst.fraunhofer.de/.

[GHH07a] *Gruler*, A. ; *Harhurin*, A. ; *Hartmann*, J.: Development and Configuration of Service-based Product Lines. In: *Proc. 11th International Software Product Line Conference (SPLC 2007)*, 2007

[GHH07b] *Gruler*, A. ; *Harhurin*, A. ; *Hartmann*, J.: Modeling the Functionality of Multi-functional Software Systems. In: *Proc. 14th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS)*, 2007

[GKM07a] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Architekturbewertung. ISST-Bericht 82/07. http://veia.isst.fraunhofer.de/.

[GKM07b] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Entscheidungsgrundlagen für die Entwicklung von Softwareproduktlinien. ISST-Bericht 83/07. http://veia.isst.fraunhofer.de/.

[GKM07c] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Fallstudie »Condition-Based Service«: Modelle für die Bewertung von logischen Architekturen und Softwarearchitekturen. ISST-Bericht 84/07. http://veia.isst.fraunhofer.de/.

[GLS08] *Gruler*, Alexander ; *Leucker*, Martin ; *Scheidemann*, Kathrin: Modeling and Model Checking Software Product Lines. In: *Proceedings of the 10th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS08)*, 2008 (LNCS).

[Gro08] *Große-Rhode*, Martin: Methods for the Development of Architecture Models in the VEIA Reference Process. ISST-Bericht 85/08. http://veia.isst.fraunhofer.de/.

[HH08] *Harhurin*, Alexander ; *Hartmann*, Judith: Towards Consistent Specifications of Product Families. In: *FM: 15th International Symposium on Formal Methods* Bd. 5014, Springer Verlag, May 2008 (LNCS)

[KBS06] *Kiebusch*, Sebastian ; *Bogdan*, Franczyk ; *Speck*, Andreas: An Unadjusted Size Measurement of Embedded Software System Families and its Validation. In: *Software Process Improvement And Practice* 11 (2006), Juni, 435–446. http://dx.doi.org/10.1002/spip.285. – DOI 10.1002/spip.285

[Kie06] *Kiebusch*, Sebastian: *Metriken für prozessorientierte Software-System-Familien: Umfangskalkulation sowie Aufwandsprognose im Electronic Business und Automobilbereich*. Leipzig, Germany, Institut für Wirtschaftsinformatik, Universität Leipzig, Dissertation, 2006. http://www.kiebusch.de/

[Kle06] *Kleinod*, Ekkart: Modellbasierte Systementwicklung in der Automobilindustrie – Das MOSES-Projekt. ISST-Bericht 77/06. http://veia.isst.fraunhofer.de/.

[Kle08] *Kleinod*, Ekkart: Modeling Overlapping Functionality. ISST-Bericht 88/08. http://veia.isst.fraunhofer.de/.

[Man08a] *Mann*, Stefan: Anwendung von Kohäsions und Kohärenzmetriken auf VEIA-Architekturmodelle zur Bewertung von Produktlinien. ISST-Bericht 86/08. http://veia.isst.fraunhofer.de/.

[Man08b] *Mann*, Stefan: *Komposition, Konfiguration und Wiederverwendung von Architekturmodellen eingebetteter Systeme*, Dissertation, TU Berlin, erscheint 2008.