Fraunhofer Institut
Software- und
Systemtechnik

Modellbasierte Systementwicklung in der Automobilindustrie

Das MOSES-Projekt



Redaktion: Ekkart Kleinod
ekkart.kleinod@isst.fraunhofer.de

ISST-Bericht 77/06

April 2006

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

ISSN 0943-1624

Inhalt

1	**Einleitung**	9
I	**Übersicht**	11
2	**Warum MOSES?**	11
2.1	Die Ausgangslage	11
2.2	MOSES	12
2.3	Kein Ziel von MOSES	16
2.4	Anwender von MOSES	17
3	**MOSES-Modelle**	19
3.1	Anforderungen	21
3.2	Logische Architektur	24
3.3	Technische Architektur	27
3.4	Partitionierung	31
4	**Benutzung**	33
4.1	MOSES-Prozesse	33
4.2	Umsetzung in die Praxis	36
5	**Ausblick**	38
II	**Modelle (Artefakte) im Detail**	41
6	**Anforderungsmodell**	41
6.1	Anforderungen	42
6.2	Kunden- und Systemanforderungen	43
6.3	Strukturierung von Anforderungen	44
6.4	Konkretisierung	45
6.5	Verfolgbarkeit – *traceability*	46
7	**Logische Architektur**	47
7.1	Funktionen	47
7.2	Hierarchisierung von Funktionen	50
7.3	Verbindung von Funktionen	51
7.4	Verhalten	53

8	**Technische Architektur**	58
8.1	Die Softwarearchitektur	59
8.2	Die Hardwarearchitektur	59
9	**Partitionierung**	65
9.1	Partitionierung auf die technische Architektur	66
9.2	Allokation der Software auf die Hardware	68
9.3	Partitionierung und Allokation	68
10	**AUTOSAR**	69
III	**Erweiterungen: Redundanz, Varianz, Domäne**	71
11	**Redundanz**	71
12	**Varianz**	73
12.1	Überlagerung und Ausleitung	73
12.2	Anforderungsmodell	75
12.3	Varianzpunkte	79
12.4	Annotation im Kontext der Partitionierung	85
13	**Domäne**	87
13.1	Anforderungen	87
13.2	Platzhalter	88
IV	**Produktlinien und Domänenmodelle**	91
14	**Umgang mit Produktlinien und Domänenmodellen**	92
14.1	Das Produktmodell	93
14.2	Das Produktlinienmodell	93
14.3	Das Domänenmodell	95
V	**Benutzung von MOSES in der Praxis**	99
15	**Sichten**	99
16	**Versionierung**	102

17	**Prozess**	104
17.1	Der MOSES-Prozess	104
17.2	Berücksichtigung von Varianz	106
17.3	Aufarbeitung von Domänenwissen	107
18	**Umsetzung in die Praxis**	109
18.1	Rollen für MOSES	109
18.2	Einführung der MOSES-Modelle	111
18.3	Organisatorische Einführung	112
Literatur		115

Abbildungen

1	Komplexität von E/E-Systemen im Auto	12
2	Einordnung von MOSES.	13
3	Sichten auf ein System	14
4	Modellierte Kernsichten von MOSES.	15
5	Produktfamilien	16
6	Das MOSES-Gesamtmodell.	19
7	Zuordnung der Modelle zu Abstraktionsniveaus.	20
8	Anforderungen im Gesamtmodell.	21
9	Stakeholder eines Systems (Beispiele).	22
10	Kunden- und Systemanforderungen.	23
11	Logische Architektur im Gesamtmodell.	24
12	Logische Architektur/Funktionsnetzwerk (Beispiel).	25
13	Funktion (Beispiel).	26
14	Technische Architektur im Gesamtmodell.	27
15	Steuergerät (Beispiel)	28
16	Systemtopologie (Beispiel).	29
17	Hardwarekomponente (Beispiel).	30
18	Partitionierung im Gesamtmodell.	31
19	Skizze des MOSES-Prozesses.	34
20	Produktlinienprozess.	35
21	Domänenprozess.	36
22	MOSES Roadmap.	38
23	Anforderungen.	42
24	Anforderungsbaum (Beispiel).	44
25	Die logische Architektur.	47
26	Funktion (Beispiel).	48
27	Funktionen: atomar und hierarchisch.	51
28	Verbindung von Funktionen.	52
29	Inneres Verhaltensmodell einer Funktion	55
30	Die technische Architektur.	58
31	Hardwarekomponenten (Beispiele).	61
32	Innere Hardwareelemente (Beispiele).	62
33	Hardwareports (Beispiele).	62
34	Auflösung intelligenter Ports.	63
35	Verbindungen (Beispiele).	64
36	Partitionierung.	65

37	Partitionierungsmöglichkeiten.	67
38	MOSES – MOSAR – AUTOSAR.	69
39	Produktlinien	73
40	Überlagerung.	74
41	Ausleitung.	75
42	Varianzpunkte und Optionalität.	80
43	Varianz und abhängige Ports (Beispiel).	83
44	Übernahme der Annotation	86
45	Kontexte und Anforderungen.	88
46	Modelle und Aktivitäten.	92
47	Aktivitäten für Produktmodelle.	93
48	Aktivitäten für Produktlinienmodelle.	94
49	Aktivitäten für Domänenmodelle.	96
50	Verwendung und Referenzierung.	97
51	Versionen.	103
52	Versionen und Varianz.	103
53	Skizze des MOSES-Prozesses.	105
54	Aktivitäten der Produktlinienmodelle.	107
55	Aktivitäten der Domänenmodelle.	108
56	Das MOSES-Gesamtmodell.	111

Tabellen

1	Dokumentation des MOSES-Projekts.	9
2	Rollen der MOSES-Methodik (Ausschnitt)	17
3	Anforderungstabelle (Beispiel).	45
4	Konkretisierung (Beispiel).	46
5	Erweiterte UND/ODER-Matrix (Beispiel).	56
6	Notwendigkeitsattribute (Beispiele).	76
7	Dimensionen in MOSES.	77
8	Abkürzende Notation für Dimensionsattribute.	77
9	Komplette Annotation (Beispiele).	78
10	*needs*-Relation (Beispiel).	78
11	*excludes*-Relation (Beispiel).	79
12	Fahrzeugidentifikatoren (VIDs).	81
13	Varianzmöglichkeiten der Hardware.	84
14	Partitionierung varianter Elemente.	85
15	Übersicht der Modelle.	91
16	Rollen der MOSES-Methodik.	109

Zum Geleit

Von August 2001 bis April 2005 hat die Abteilung »Verlässliche Technische Systeme« (VTS) des Fraunhofer ISST für die BMW Group eine Methode zur modellbasierten Entwicklung von Elektrik-/Elektroniksystemen im Automobil entworfen. Am Projekt haben mitgewirkt: Ahmad Al-Labadi, Caroline Berthomieu, Alexander Borusan, Marek Feldo, Martin Große-Rhode, Markus Hardt, Augustin Kebemou, Ekkart Kleinod, Jürgen Likkei, Sven Lindow, Rainer Mackenthun, Stefan Mann, Mesut Özhan, Juliane Siegeris, Wolfram Webers. Das Projekt war eingebettet in das »Change-Programm Systemorientierung« der BMW Group, in dem die gesamte E/E-Prozesskette im Sinne einer konsequenten Ausrichtung auf das E/E-Gesamtsystem redefiniert wurde.

Die Aufgabe des Projekts MOSES (MOdellbaSiertE Systementwicklung) bestand darin, Methoden und Notationen für die Engineering-Prozesse Anforderungsmanagement, Systemdesign, Software- und Hardware-Entwicklung sowohl auf der System- als auch auf der Komponentenebene zu definieren. Die Ergebnisse sollten neben der Entwicklung individueller Systeme auch Engineering-Prozesse für die Domäne unterstützen und damit Wiederverwendungskonzepte systematisch in die Prozesse einführen. Dazu wurden Konzepte der (Software-)Produktlinienentwicklung adaptiert und erweitert. Diese Konzepte zielen insbesondere auf die Repräsentation von Varianten, die Konfiguration von Modellen und das Erstellen und Verwenden von Modell-Baukästen ab.

Das Projekt war untergliedert in vier Phasen, in denen die Lösungen für die Modellierung von (1) Anforderungen, (2) vernetzten Funktionen und (3) Architekturen erarbeitet wurden; sowie (4) die Validierung und Umsetzung der Methode untersucht wurden. Im Anschluss an die einzelnen Phasen wurden zusätzliche Migrationsprojekte durchgeführt, in denen die Ergebnisse in laufenden Entwicklungsprojekten bei der BMW Group evaluiert wurden. Die kontinuierliche Überwachung der Umsetzbarkeit diente auch der Steuerung des Gesamtprojekts.

Gemäß der Projektstruktur wurden die Ergebnisse sukzessive in überschaubaren Abschnitten erarbeitet. Die Validierungsphase wurde insbesondere genutzt, um Rückwirkungen von Entwurfsentscheidungen auf die bereits definierten Phasen umzusetzen. Dadurch wurde sichergestellt, dass einheitliche und durchgänge Konzepte für den gesamten Engineering-Prozess auf Komponenten-, System- und Domänenebene zur Verfügung stehen.

In der wissenschaftlichen Nachbereitung des Projekts ergeben sich naturgemäß andere Sichtweisen auf die in dem Anwendungsprojekt erarbeiteten Konzepte,

als sie in der auftragsgemäßen Durchführung eingenommen werden. Von diesem Standpunkt aus würden wir heute einige Dinge anders formulieren, als sie in diesem Bericht zusammengefasst sind. Das betrifft vor allem die erste Phase des Projekts, in dem die Anforderungsmodellierung definiert wird. Ziel des vorliegenden Berichts ist ausschließlich die Darstellung der Ergebnisse so, wie sie im Projekt erarbeitet wurden. Die wissenschaftliche Ausarbeitung weiterer, vertiefter Fragestellungen, die sich im Projekt ergeben haben, sowie geänderter Sichtweisen auf Probleme und Lösungen, wird an anderer Stelle publiziert.

Neben dem Autor waren weitere Personen an der Erstellung des Dokuments beteiligt. Allen voran half Martin Große-Rhode bei der Konzeption und im Entstehungsprozess mit Rat und Tat weiter. Stefan Mann, Sven Lindow und Marek Feldo übernahmen die Korrektur inhaltlicher und formaler Fehler. Bertram Lamm übernahm die Koordination mit der BMW Group und sorgte für die Freigabe des Dokuments durch BMW.

1 Einleitung

Das Akronym MOSES steht für »MOdellbaSiertE Systementwicklung« und bezeichnet eine Methodik für durchgängige modellbasierte Systementwicklung von Elektrik-/Elektroniksystemen (E/E-Systemen) in Fahrzeugen. MOSES besteht aus der Definition eines durchgehenden Modells für die Systementwicklung. Prozessaspekte sind bisher ausgeklammert worden.

Dieses Dokument fasst die bisher gewonnenen Erkenntnisse zusammen und gibt einen Überblick über die entstandenen Ergebnisse. Das Dokument ist für Anwender von MOSES geschrieben worden, es werden keine Kenntnisse von MOSES vorausgesetzt. Vorkenntnisse im Bereich der Systementwicklung bzw. -modellierung sind zum Verständnis der Thematik nötig. Insbesondere werden Begriffe, wie der System- oder der Architekturbegriff, nicht weiter erläutert.

Das Dokument beschreibt die Konzepte von MOSES, das gilt vor allem für die Metamodelle. Offene Punkte bzw. Fragen von MOSES werden benannt, aber nicht ausführlich diskutiert.

Ergänzend zu diesem Dokument gibt die Metamodellbeschreibung detaillierte Informationen über die Metamodellklassen der beschriebenen Konzepte. Die gesamte Dokumentation zum MOSES-Projekt liegt in sechs Einzeldokumenten vor (siehe auch das Literaturverzeichnis):

Tabelle 1 Dokumentation des MOSES-Projekts.

MOSES 1	Anforderungsmodellierung und Anforderungsmanagement im Domain Engineering	[Fra02]
MOSES 2	Modellierung von hierarchischen, vernetzten Funktionen	[Fra03a]
MOSES 3.1	Architekturmodellierung, Teil 1, Metamodell für technische Architekturmodelle und Partitionierung von logischen Funktionen	[Fra03b]
MOSES 3.2	Architekturmodellierung, Teil 2, Erweiterte Modellierung und Bewertung von Partitionierungen	[Fra04a]
MOSES 4.1	Validierung und Umsetzung der modellbasierten Entwicklungsmethodik MOSES, Teil 1: Konfiguration, Produktlinien- und Domänenmodelle für Funktionsnetzwerke	[Fra04b]
MOSES 4.2	Validierung und Umsetzung der modellbasierten Entwicklungsmethodik MOSES, Teil 2: Produktlinien- und Domänenmodelle technischer Architekturen	[Fra05b]
Gesamtmodell	Das MOSES-Gesamtmetamodell	[Fra05a]

Teil I

Übersicht

Dieser Teil stellt die MOSES-Methode knapp vor und gibt einen Überblick, wozu MOSES dient, was es leisten kann und was nicht. Die Bestandteile werden aufgeführt und der praktische Einsatz von MOSES wird kurz thematisiert. Teil I schließt mit einem Ausblick auf offene Fragen und Weiterentwicklungen im Kontext von MOSES.

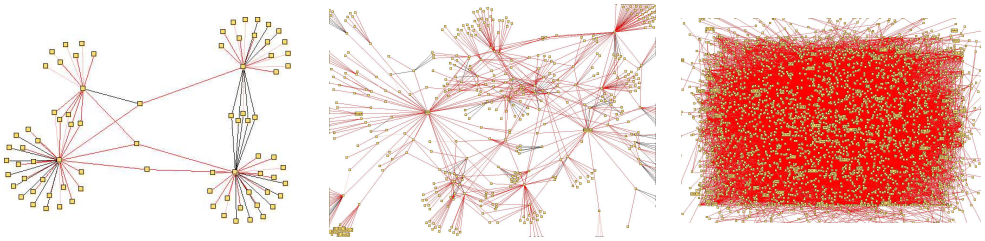
2 Warum MOSES?

Dieses Kapitel erläutert, warum es nötig war, MOSES zu entwickeln. Dabei wird zunächst die aktuelle Situation im Bereich der Systementwicklung eingebetteter Systeme, speziell im Automobilbau, vorgestellt. Dann werden die Ideen von MOSES erläutert und geklärt, wozu MOSES dient und was es nicht leisten kann. Das Kapitel schließt mit der Vorstellung der Stakeholder von MOSES, bevor das folgende Kapitel detaillierter auf die Bestandteile von MOSES eingeht.

2.1 Die Ausgangslage

Die Automobilindustrie steht vor großen Herausforderungen. Die Bedeutung der Elektrik-/Elektronik-Systeme in Fahrzeugen nimmt rapide zu, kein Hersteller kann sich dem Einsatz moderner E/E-Systeme verschließen, da der Einsatz von Funktionen, die auf Elektrik- bzw. Elektronik basieren, zunehmend zum entscheidenden Sicherheits-, Funktions- und Marketingargument wird. Insbesondere die Modelldifferenzierung wird von Elektrik-/Elektronik-Systemen getrieben. Gleichzeitig steigt die Komplexität der eingebauten Systeme stark an. Abbildung 1 zeigt die Komplexität von einer Komponente bis hin zum gesamten Bordnetz eines modernen Autos. Die Beherrschung dieser Komplexität ist leicht zu unterschätzen und muss durch geeignete Prozesse unterstützt werden.

Abbildung 1 Komplexität von E/E-Systemen im Auto (Quelle: BMW [bmw]).



Elemente und Abhängigkeiten einer Komponente

Elemente und Abhängigkeiten zweier Komponenten

Elemente und Abhängigkeiten des gesamten Bordnetzes

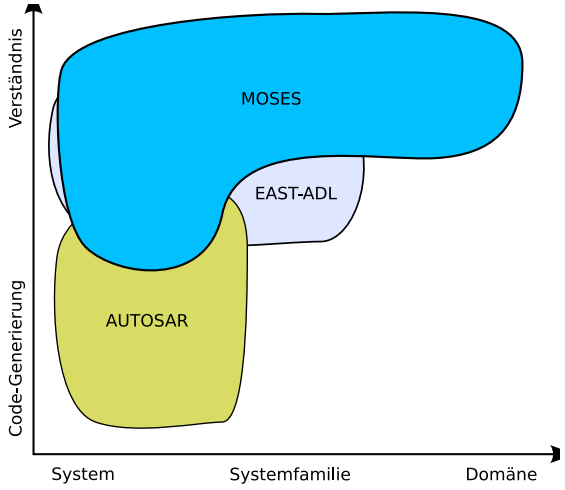
Die bisher eingesetzten bzw. entwickelten Software-Entwicklungsprozesse greifen in diesem Kontext zu kurz oder sind zu weit gefasst: Zum einen sind die zu entwickelnden Systeme keine reinen Softwaresysteme, sondern enthalten Software- und Hardwareanteile. Weiterhin handelt es sich um eingebettete Systeme mit Echtzeitanforderungen, die von den meisten Prozessen per Definition nicht unterstützt werden. Drittens sind viele Prozessbeschreibungen sehr generisch und müssen vom Anwender per »Tailoring« angepasst werden. Damit können diese Prozesse nicht auf die spezifischen Bedürfnisse von Automobilherstellern eingehen.

Zusätzlich zu den ungenügend unterstützten Prozessen sind auch die Modellierungsmittel für eingebettete Systeme unzureichend. Beschreibungssprachen wie UML setzen erst in neueren Versionen (UML 2.0) den Komponentengedanken konsequent um. Dabei wird die Sprache jedoch so weit aufgefächert, dass das eigentliche Ziel – schnelle, einfache und adäquate Modellierung – nicht mehr erreicht wird. Spezielle Modellierungssprachen und -konzepte wie EAST-ADL oder AUTOSAR entstanden parallel zu MOSES und verdeutlichen, dass Lösungen gebraucht werden.

2.2 MOSES

In dieser Situation hat das Fraunhofer-Institut für Software- und Systemtechnik in Zusammenarbeit mit der BMW Group ein langfristiges Projekt mit dem Ziel gestartet, eine Methodik zur Systementwicklung eingebetter E/E-Systeme zu schaffen. Diese Methodik sollte auf den Automobilbau zugeschnitten sein und Modelle sowie Prozessvorgaben umfassen. Es wurde zunächst eine Beschreibungssprache

Abbildung 2 Einordnung von MOSES.



geschaffen, in der die E/E-Systeme beschrieben werden können. In weiteren Projekten wird der Prozess definiert und langfristig in die Praxis umgesetzt.

Unter anderem sind folgende Punkte bei der Konzeption von MOSES von entscheidender Bedeutung gewesen:

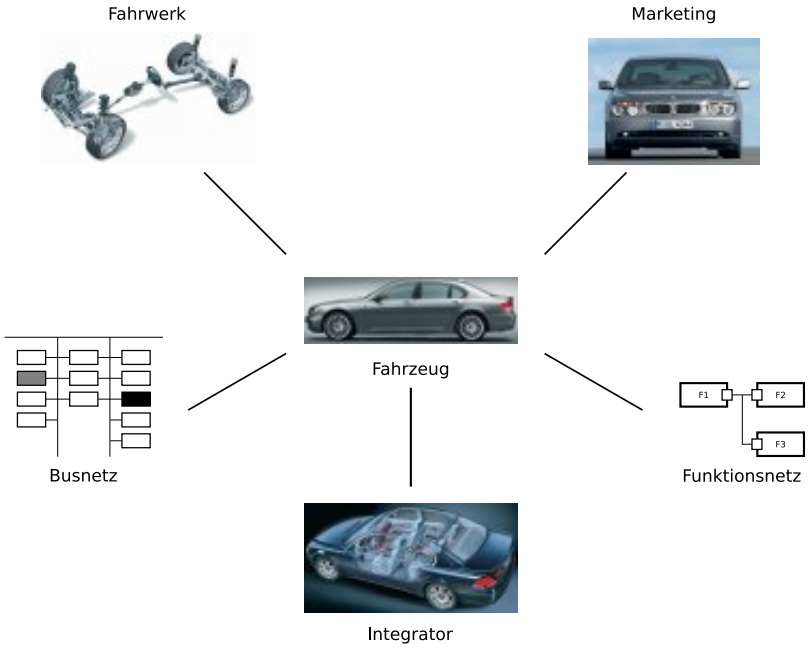
- die entstehenden Modelle müssen praxisnah sein
- die Arbeit verschiedener Anwender (Stakeholder) vom Entwickler bis zum Integrator soll unterstützt werden
- Informationen über das System müssen redundanzfrei am richtigen Ort modelliert werden
- die Modelle sollen abstrakte Sichten auf das System ermöglichen
- Entwurfsentscheidungen bzw. -änderungen müssen nachvollzogen werden können (*traceability*)

Abbildung 2 zeigt die Einordnung von MOSES in die bestehende Modellwelt: MOSES beschreibt Systeme, Systemfamilien sowie Domänen auf Verständnisebene. EAST-ADL zielt ebenfalls auf Verständnismodelle ab, beschränkt sich allerdings auf Systeme und Systemfamilien. AUTOSAR schließlich beschreibt Generierungsmodelle für Systeme, aus AUTOSAR können Software- und Hardware-Systembeschreibungen generiert werden.

Ein System kann je nach Betrachter unterschiedlich aussehen, Abbildung 3 zeigt (nicht vollständig), welche Vielfalt der Beschreibung bei einem Auto möglich ist.

Das Marketing will das Auto vermarkten und muss dafür die wichtigsten Merkmale kennen. Der Systemverantwortliche interessiert sich für die Integration der einzelnen Subsysteme, der Busnetzverantwortliche für die Systemtopologie, während der Funktionsmodellierer Funktionen im Blick hat. Das Fahrwerk ist sowohl aus E/E-Sicht als auch als Bauteil wichtig.

Abbildung 3 Sichten auf ein System (Bilder: BMW [bmw]).

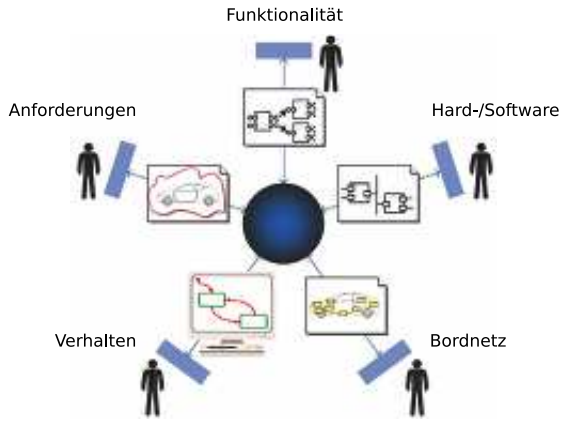


MOSES beschränkt sich auf vier Kernsichten, die ein E/E-System detailliert beschreiben:

- Anforderungen (verbale Beschreibung des Gesamtsystems)
- Logische Architektur (Beschreibung der umzusetzenden Funktionalität)
- Technische Architektur (Beschreibung der technischen Umsetzung in Hard- und Software)
- Partitionierung (Beschreibung der Verteilung der logischen auf die technische Architektur)

Die Kernsichten sind in Abbildung 4 visualisiert: Anforderungen (Anforderungsmodell), Funktionalität (logische Architektur), Hard- und Software sowie das Bordnetz (alles technische Architektur) und Verhalten (Anforderungen und logische Architektur). Die Partitionierung als Bindeglied zwischen logischer und technischer Architektur ist in der Abbildung im Kreis in der Mitte anzusiedeln.

Abbildung 4 Modellierte Kernsichten von MOSES.



Der Fokus bei der Auswahl der Sichten lag darauf, dass das System abstrakt modelliert und übersichtlich in den Kernaspekten dargestellt wird, die letztendlich zur Systemintegration notwendig sind. MOSES-Modelle sind Verständnismodelle, keine Generierungsmodelle. MOSES ist ein architekturzentrierter Ansatz, der gleichzeitige Sichten auf ein- und dasselbe System ermöglicht.

Die Anforderungen beschreiben das gesamte System verbal. Die implementierungsunabhängige Funktionalität wird durch die logische Architektur modelliert. Die Soft- und Hardware wird in der technischen Architektur beschrieben, der Zusammenhang zwischen Funktion und Implementierung durch die Partitionierung. Es werden also Anforderungen, Funktion und Technik voneinander getrennt. Die einzelnen Modelle werden in Kapitel 3 kurz vorgestellt und in Teil II detailliert erläutert.

Ein weiterer Aspekt der Systementwicklung ist die Entwicklung von Systemfamilien. Im Automobilbereich ist das System beispielsweise ein Auto und die Systemfamilie eine Produktfamilie wie eine Baureihe oder Plattformmodelle. Zwei Beispiele für Produktfamilien sind in Abbildung 5 abgebildet. Die gezeigten Produktfamilien sind fiktive Beispiele, keine realen Produktfamilien bei BMW. Welche Systeme zu einer Systemfamilie zusammengefasst werden, ist eine Entscheidung eines Herstellers nach seinen Kriterien.

Systemfamilien werden in MOSES explizit durch Varianzkonzepte unterstützt, die für jedes Architekturmodell speziell auf dessen Möglichkeiten und Bedürfnisse abgestimmt sind. Das Varianzkonzept wird zusammen mit dem Redundanzkonzept in Teil III vorgestellt.

Abbildung 5 Produktfamilien (fiktiv, Bilder: BMW [bmw]).

3er-Familie 2005



Produktpalette 2005



Die Anbindung an bestehende bzw. entstehende Konzepte wurde bei der Entwicklung von MOSES berücksichtigt. Insbesondere die Nutzung von AUTOSAR, einer Beschreibung auf implementierungsnahem Niveau, wurde explizit bei der Modellierung berücksichtigt. MOSES kann als Bindeglied zwischen Anforderungserfassung und Modellierung in AUTOSAR dienen und damit die Abstraktionsschritte und Designentscheidungen dokumentieren und unterstützen. Die Anbindung und Nutzung von AUTOSAR wird in Kapitel 10 beschrieben.

2.3 Kein Ziel von MOSES

Für die Einordnung bzw. das Verständnis von MOSES ist es wichtig zu wissen, was MOSES nicht leisten kann.

Bisher wird durch MOSES kein Prozess definiert. Das heißt nicht, dass die Schritte zu MOSES-Modellen unklar oder die Beziehungen zwischen den Modellen nicht definiert sind. Diese Schritte wurden jedoch bisher als Handlungsanweisungen oder *best-practice*-Ansätze formuliert, sie wurden nicht formalisiert. Insbesondere wurde kein durchgehender Prozess von den Anforderungen bis zu den partitionierten Architekturmodellen beschrieben.

Des Weiteren bietet MOSES keine Unterstützung für begleitende Prozesse. Soweit diese Prozesse im Projektverlauf aufgetreten sind, wurden Handlungsanweisungen und Vorschläge unterbreitet, wie die Prozesse in MOSES integriert werden

können. So wurde beispielsweise die Modellverwaltung bezüglich der Versionierung untersucht (mehr in Kapitel 16).

Die Verhaltensmodellierung von MOSES ist rudimentär, da aus projekthistorischen Gründen die statischen Anteile der Modellierung in den Vordergrund gerückt sind. Verhalten wird bei der logischen Architektur angerissen, aber nicht ausformuliert. In der technischen Architektur werden dynamische Anteile nicht berücksichtigt, diese Modellierung wird zur Zeit in einem Nachfolgeprojekt vorangetrieben.

2.4 Anwender von MOSES

MOSES wird von Anwendern genutzt, die während des Systementstehungsprozesses mit der Systementwicklung betraut sind. Da der Prozess von der Anforderungserfassung bis zum Systemdesign unterstützt wird, sind die Anwender dementsprechend vom Anforderungsverantwortlichen bis zum Steuergeräteverantwortlichen vertreten.

Der Übersichtlichkeit halber wird in Systementwicklungsprozessen statt von Anwendern von »Rollen« gesprochen, die für typische Anwender im Prozess stehen. So umfasst z. B. die Rolle der »Softwareentwickler« alle Anwender, die mit der Erstellung und Implementierung von Software befasst sind. Tabelle 2 zeigt eine Auswahl der Rollen im Rahmen von MOSES, eine vollständige Auflistung ist Tabelle 16 auf Seite 109 zu entnehmen:

Tabelle 2 Rollen der MOSES-Methodik (Ausschnitt aus Tabelle 16).

Rolle	Beschreibung
Anforderungsverantwortlicher	Der Anforderungsverantwortliche ist dafür zuständig, die Anforderungsmodellierung zu planen und zu überwachen.
Systemintegrator	Der Systemintegrator ist für die Planung und Überwachung des Systementwurfs zuständig. Dazu gehört neben den Entwurfsprozessen auch die Integration der fertigen Subsysteme in das Gesamtsystem. Zunächst ist der Systemintegrator für die Erstellung der logischen und technischen Architektur zuständig. Dazu erarbeitet er mit dem Funktionsnetzverantwortlichen, dem Softwareverantwortlichen und dem Steuergeräteverantwortlichen jeweils deren Architekturen. Dann übergibt er die Architekturen an die Verantwortlichen und integriert deren Ergebnisse in das Gesamtsystem.
Softwareentwickler	Der Softwareentwickler implementiert eine Softwarekomponente oder Teile davon.

Rolle	Beschreibung
Steuergeräteverantwortlicher	Der Steuergeräteverantwortliche überwacht und leitet die Erstellung der Hardwarearchitektur. Dabei verfeinert er eine gegebene oder selbst erstellte Hardwaretopologie und teilt diese in Steuergeräte auf, die dem Steuergeräteentwickler in Auftrag gegeben werden. Die fertigen Geräte werden in die Hardwarearchitektur integriert.
Domänenverantwortlicher	Der Domänenverantwortliche ist für die Pflege des Domänenmodells verantwortlich. Wie der Produktlinienverantwortliche hat er Prozesse zu definieren und deren Einhaltung zu überwachen, so dass das Domänenmodell konsistent bleibt.

Die Rollen bezeichnen zum einen Verantwortlichkeiten: »Anforderungsverantwortlicher«. Zum anderen geben sie vor, mit welchen Modellen die Rolle zu tun haben wird: »Steuergeräteverantwortlicher«. Drittens bezeichnen sie auch die Art der Modelle, die bearbeitet werden: »Domänenverantwortlicher«.

Nach dieser sehr kurzen Einführung in die MOSES-Welt stellt der folgende Abschnitt die Architekturmodelle von MOSES genauer vor. Dabei wird immer noch ein Überblick angestrebt, die Detailbesprechung der Architekturmodelle erfolgt in Teil II.

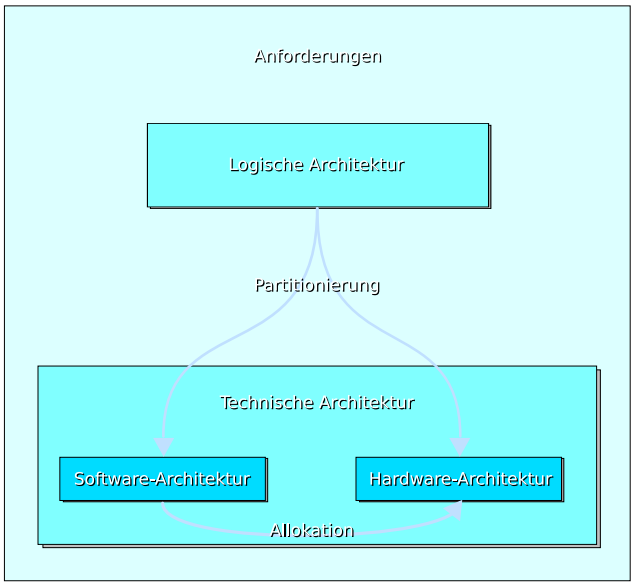
3 MOSES-Modelle

Wie bereits erwähnt, beschreibt MOSES ein System mit folgenden Modellen:

- Anforderungen
- Logische Architektur
- Technische Architektur
- Partitionierung

Der Zusammenhang zwischen den Modellen ist in Abbildung 6 dargestellt. Die Anforderungen bilden die Basis der Systembeschreibung und damit auch die Basis der anderen Modelle. Die technische Architektur besteht aus den zwei Teilarchitekturen für Software und Hardware. Die Partitionierung verbindet logische und technische Architektur sowie Software und Hardware.

Abbildung 6 Das MOSES-Gesamtmodell.



Die einzelnen Sichten werden in MOSES jeweils als Metamodell definiert, aus denen Instanzen gebildet werden – die Systemmodelle – pro Sicht ein Modell. Wie in Abbildung 6 zu sehen ist, stehen die Modelle miteinander in Verbindung. Die Anforderungen sind die Basis der anderen Modelle, die Partitionierung verbindet

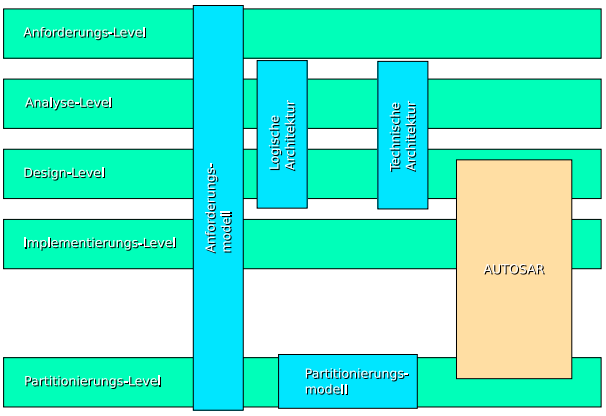
Logik und Implementierung. Damit wird die Verfolgbarkeit (*traceability*) der Modellierungsentscheidungen umgesetzt. Das heißt, Änderungen in einem Teilmodell können mit den anderen Teilmodellen abgeglichen und konsistent gehalten werden.

Die Aufteilung in diese vier Teilsichten resultiert aus dem Prinzip des *Separation of Concerns*. Dieses Prinzip besagt, dass verschiedene Aspekte eines Systems getrennt beschrieben werden sollten. Dabei wird nicht vorgegeben, welche oder wieviele Sichten zu berücksichtigen sind, die modellierten Sichten decken die Aspekte der Systemmodellierung gut ab.

Damit werden Informationen an dem Ort gehalten, an dem sie entstehen und logisch hingehören. Die vier Architekturmodelle wurden von BMW als Kernelemente bestätigt. Ein Aspekt wurde nicht berücksichtigt: die Geometrie. Dieser Aspekt wurde bewusst nicht modelliert, da von dessen Einflüssen auf z. B. Kabelbaumgestaltung, Platinenbau etc. bei MOSES abstrahiert wurde.

Die Sichten beschreiben das System auf jeweils unterschiedlichem Abstraktionsniveau. Die Zuordnung der Modelle zu den unterschiedlichen Abstraktionsniveaus gibt Abbildung 7.

Abbildung 7 Zuordnung der Modelle zu Abstraktionsniveaus.



Die Anforderungen beschreiben das System allgemein, sie sind dem Anforderungs-Level sowie dem Analyse-Level zuzuordnen. Auf dem Analyse-Level beginnt der Übergang von den Anforderungen zum Design, dieser Übergang wird von der logischen und der technischen Architektur abgedeckt. Der Implementierungs-Level wird in MOSES nicht modelliert, er wird von AUTOSAR vollständig abgedeckt. Auf dem Partitionierungs-Level findet die Partitionierung statt, die Aufteilung der Funktionen auf Soft- bzw. Hardware, sowie die Verteilung der

Software auf die Hardware.

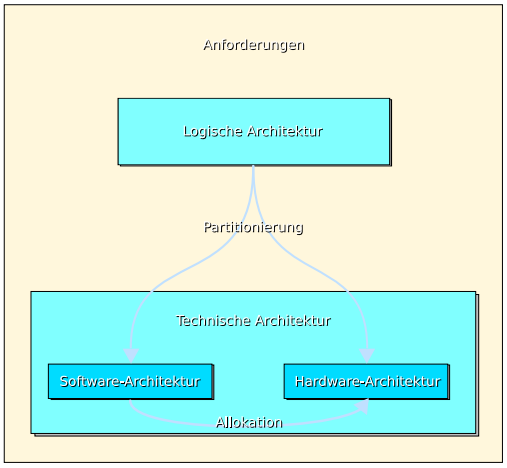
Diese Aufteilung des Systems in Anforderungen, logische und technische Architektur sowie Partitionierung ist nicht automobilspezifisch. Jedes technische System kann in diesen Sichten betrachtet werden. Die konkrete Ausgestaltung der Metamodelle innerhalb von MOSES ist jedoch spezifisch für den Automobilbau erfolgt. Damit sind die Metamodelle für Automobile einfach anzuwenden und passen sich in die gewohnten Abläufe und Begriffe bei Automobilherstellern ein. Die Anpassung für andere Anwendungsbereiche ist nicht trivial und erfordert teilweise Neumodellierung.

Die Metamodelle werden durch Klassendiagramme und Object-Z-Constraints beschrieben. Sie lehnen sich konzeptuell an UML und UML-RT an, sind aber nicht so nah verwandt, dass eine Transformation möglich wäre. Die formale Definition ist sehr umfangreich und ist in den Dokumenten zu den MOSES-Phasen sowie in einem Metamodelldokument hinterlegt. An dieser Stelle werden nur die Grundkonzepte erläutert, Details sind dem Metamodell-Dokument zu entnehmen.

Im Folgenden werden die vier Einzelmodelle näher vorgestellt.

3.1 Anforderungen

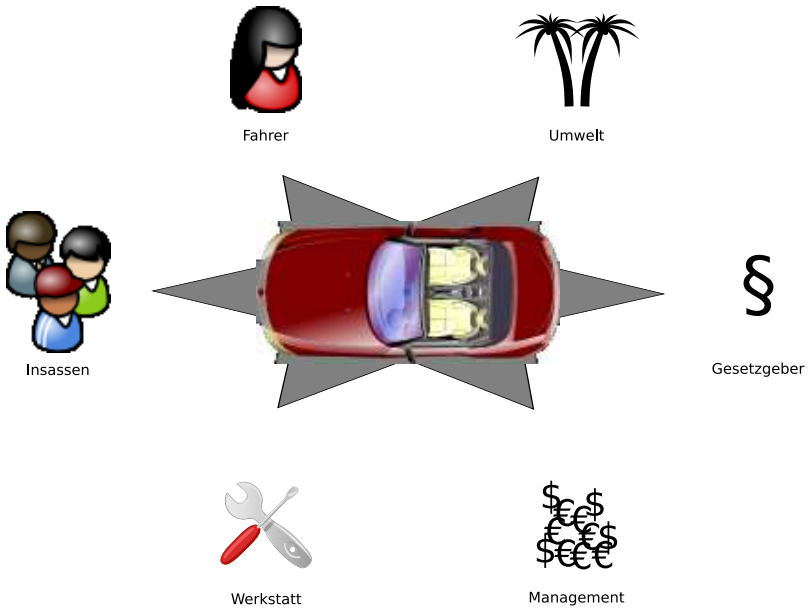
Abbildung 8 Anforderungen im Gesamtmodell.



Zweck

Das Anforderungsmodell (gelb in Abbildung 8) sammelt die Anforderungen der Stakeholder an ein System. Der Begriff »Stakeholder« bezeichnet dabei alle Personen bzw. Institutionen, die Anforderungen an das System stellen. Abbildung 9 zeigt einige Beispiele für Stakeholder. Es ist zu sehen, dass nicht nur Fahrer oder Werkstätten Stakeholder sind, sondern auch Gesetzgeber, Management, Insassen etc. Damit werden sehr viele Anforderungen erhoben, daraus ergibt sich die Notwendigkeit, die unterschiedlichen Anforderungen zu strukturieren und einzuordnen.

Abbildung 9 Stakeholder eines Systems (Beispiele).

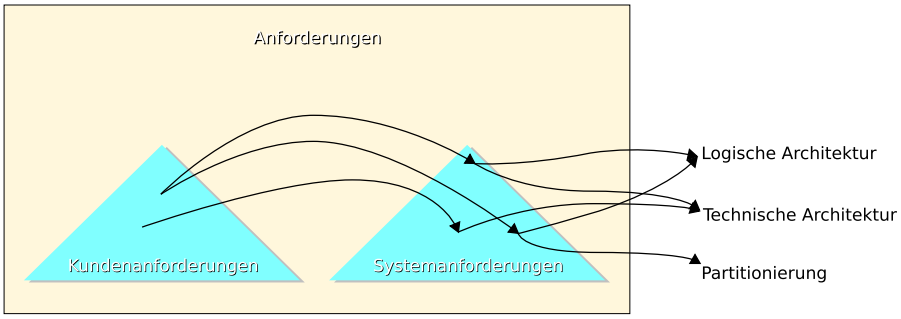


Umsetzung

Üblicherweise werden Features und Systemanforderungen unterschieden. MOSES trifft die Unterscheidung etwas feiner und definiert Kunden- und Systemanforderungen. Kundenanforderungen sind implementierungsunabhängig und beschreiben allgemeine Anforderungen an ein Fahrzeug ohne technische Details. Die Features der featureorientierten Methoden sind in den Kundenanforderungen enthalten, Kundenanforderungen sind aber umfangreicher und oft detaillierter.

Systemanforderungen realisieren Kundenanforderungen. Sie sind die implementierungsabhängige Umsetzung der Kundenanforderungen. Sie beschreiben, wie

Abbildung 10 Kunden- und Systemanforderungen.



das System realisiert wird, dabei wird von den Kundenanforderungen ausgegangen und beschrieben, wie diese umgesetzt werden sollen.

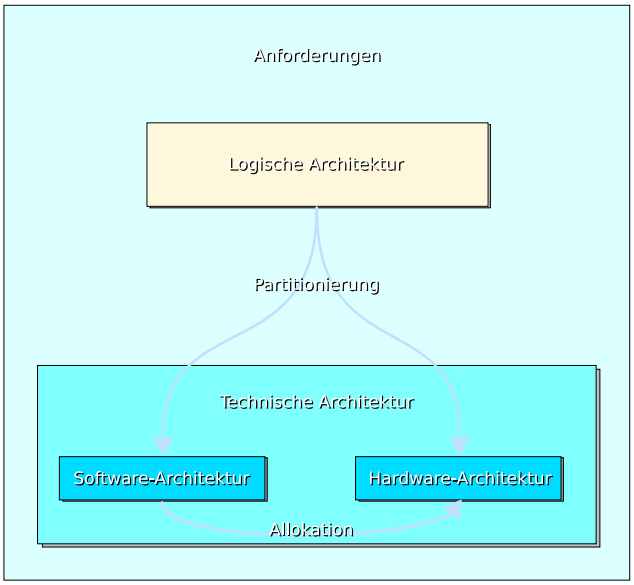
Diese Modellierung bietet den Vorteil, dass implementierungsunabhängige von implementierungsabhängigen Anteilen getrennt werden. Trotzdem können die Anforderungen über die Realisierungsbeziehung nachvollzogen werden.

Die Anforderungen betreffen das gesamte Fahrzeug, das heißt, sie stellen Anforderungen an alle anderen Architekturmodelle. Diese Anforderungen müssen von den anderen Teilmodellen umgesetzt werden. Damit wird eine Kette von den Kunden- über die Systemanforderungen zu z. B. Funktionen geschaffen und die Funktionen so fundiert. Diese Kette wird in Abbildung 10 gezeigt.

Produktlinien

Produktlinien werden durch Varianzkonzepte realisiert. Innerhalb des Anforderungsmodells sind dafür *needs*- und *excludes*-Beziehungen vorgesehen, die Beziehungen zwischen den Anforderungen darstellen. Außerdem können Anforderungen mit Konfigurationsinformationen versehen werden, die eine automatische Ausleitung ermöglichen.

Abbildung 11 Logische Architektur im Gesamtmodell.



3.2 Logische Architektur

Zweck

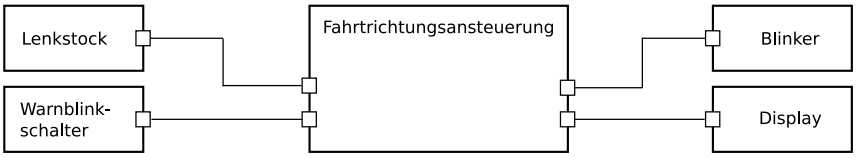
Die logische Architektur (gelb in Abbildung 11) dient zur Modellierung der Funktionalität eines Systems. Diese Funktionalität bzw. eine Teilfunktionalität davon wird durch Funktionen realisiert. Durch geschicktes Kombinieren der Funktionen werden größere Funktionalitäten gebildet.

Diese Kombination umfasst zum einen die hierarchische Strukturierung der Funktionen, zum anderen die Verbindung der Funktionen untereinander. Damit werden zwei Effekte erzielt: die Hierarchisierung dient dazu, Funktionen zu kapseln, also die Details einer Funktion nach außen hin zu verstecken. Die äußere Sicht dieser Funktion wird als Schnittstelle der Funktion bezeichnet. Die Kenntnis dieser Schnittstelle ermöglicht es anderen Funktionen, Daten mit der Funktion auszutauschen oder Operationen der Funktion aufzurufen. Diese Kommunikation der Funktionen wird durch die Verbindung der Funktionen ausgedrückt.

Mit diesen zwei grundlegenden Prinzipien kann ein Netzwerk von Funktionen aufgebaut werden, das die gewünschte Gesamtfunktionalität realisiert. Dieses Netzwerk wird als Funktionsnetzwerk bezeichnet. Das Resultat der logischen Ar-

chitekturmodellierung ist genau dieses Funktionsnetzwerk. Die Begriffe logische Architektur und Funktionsnetzwerk sind also gleichwertig. Abbildung 12 zeigt ein Funktionsnetzwerk für einen Fahrtrichtungsanzeiger.

Abbildung 12 Logische Architektur/Funktionsnetzwerk (Beispiel).



Wichtig ist, dass die Funktionalität eines Fahrzeugs unabhängig von der konkreten Umsetzung beschrieben ist. Ob z. B. Sensoren im Steuergerät liegen oder extern verbaut werden, ist für eine Funktion unerheblich, für sie ist die Qualität der Daten entscheidend. In diesem Sinne ist auch die logische Architektur implementierungsunabhängig. Sie enthält die Funktionalität des Fahrzeugs, nicht deren Umsetzung. Die Umsetzung der Funktionalität wird durch die technische Architektur geleistet.

Umsetzung

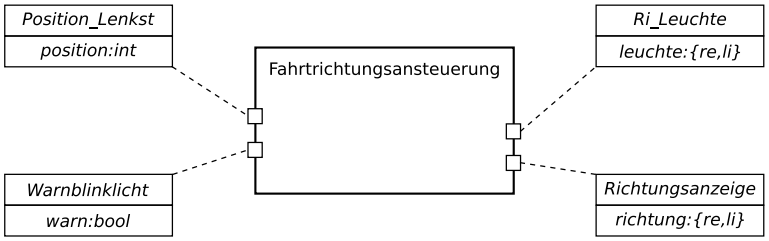
Die logische Architektur wird also durch hierarchische Funktionen beschrieben, die miteinander kommunizieren können und Schnittstellen besitzen. Diese Funktionen werden als Komponenten im Sinne von Software-Architektur-Beschreibungssprachen realisiert. Damit besitzen sie Eingänge und Ausgänge sowie einen *Body*, der die Ein- und Ausgänge verknüpft. Jegliches Verhalten der Komponente wird im *Body* beschrieben.

Die Schnittstelle einer Funktion wird über Ports und deren *Interfaces* beschrieben.[1] Ports sind die Verbindungspunkte einer Funktion, nur über Ports können Funktionen miteinander kommunizieren. Die *Interfaces* enthalten die Signale, die über die Ports ausgetauscht werden können bzw. die Operationen, die an den Ports aufgerufen werden können. Neben der Struktur wird das Verhalten der *Interfaces* über Protokollstatecharts spezifiziert. Damit wird eine Funktion von außen strukturell und im Verhalten hinreichend genau beschrieben, um Funktionen miteinander verbinden zu können.

[1] Im Folgenden wird der Begriff »Schnittstelle« für die Außendarstellung einer Funktion benutzt, während »*Interface*« die Gliederung der Signale ist, die dem Port zugeordnet werden.

Die Funktion *Fahrtrichtungsanzeiger* von Abbildung 13 besitzt vier Ports mit je einer Schnittstelle. Jede Schnittstelle enthält im Beispiel ein Signal.

Abbildung 13 Funktion (Beispiel).



Im Inneren der Funktion können weitere Funktionen oder ein Verhaltensmodell liegen. Die Idee dahinter ist, dass sich das Gesamtverhalten einer Funktion aus dem Verhalten ihrer Unterfunktionen zusammensetzt. Hat die Funktion keine inneren Funktionen, ist sie also ein Blatt im Funktionsbaum, so muss ihr Verhalten definiert werden. Das Verhalten kann über übliche UML-Mittel, z. B. als Statechart notiert werden.

Damit sind die Schnittstellen und das Verhalten einer Funktion beschrieben, das Gesamtverhalten ist ebenfalls determiniert.

Produktlinien

Für Produktlinien werden ebenfalls logische Architekturen modelliert. Dafür wird, wie auch in der technischen Architektur und der Partitionierung, das Konzept der Varianzpunkte benutzt. Dieses Konzept besagt, dass an Stellen, an denen Varianten in einer Produktlinie vorkommen, ein Varianzpunkt modelliert wird. An diesem Varianzpunkt muss sich der Entwickler später entscheiden, welche Variante in einem konkreten Fahrzeug realisiert wird.

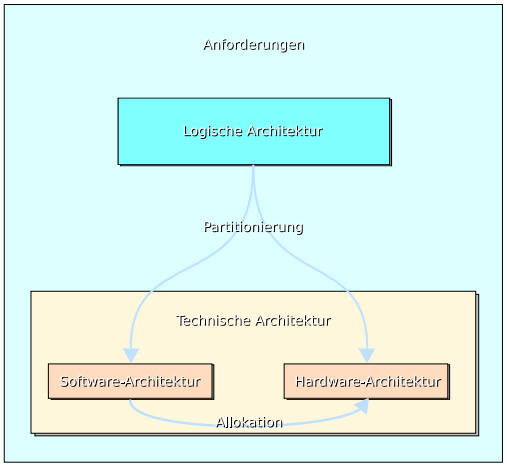
Varianzpunkte können zwei Ausprägungen haben: Varianzpunkte für Optionalität und Varianzpunkte für Auswahl von Varianten. Bei Varianzpunkten, die Optionalität repräsentieren, muss die Entscheidung getroffen werden, ob eine Funktion zum Funktionsnetzwerk gehören soll oder nicht. An Varianzpunkten für die Auswahl von Varianten muss die Entscheidung getroffen werden, welche der vorgeschlagenen Varianten zum Funktionsnetzwerk gehören soll. Diese beiden Prinzipien bieten geeignete Möglichkeiten, Variantenvielfalt in Modellen zu berücksichtigen.

In der logischen Architektur erstrecken sich die Varianzmöglichkeiten auf Funktionen, Ports, Signale und Verbindungen. Die hauptsächlich verwendete Varianz tritt

bei Funktionen auf; Port-, Signal- und Verbindungsvarianz resultieren meist aus der Funktionsvarianz. Die Ausleitung der Produktlinien ist über die Annotation möglich, die logische Bedingungen mit Entscheidungen verknüpft.

3.3 Technische Architektur

Abbildung 14 Technische Architektur im Gesamtmodell.



Zweck

Mit der technischen Architektur (gelb in Abbildung 14) werden die technischen Details des Systems modelliert. Dabei werden sowohl Software als auch Hardware modelliert. Die beiden Aspekte werden auch als Software- und Hardwarearchitektur bezeichnet.

Software besteht aus Routinen, die zu Softwarekomponenten zusammengefasst werden können. Eine solche Softwarekomponente bildet eine auslieferbare Einheit. Die Routinen selbst werden auf Prozessoren ausgeführt und müssen dafür zu Tasks zusammengesetzt werden können. Die Verteilung der Softwarekomponenten auf unterschiedliche Hardware soll möglichst frei sein, es sind jedoch hardwareabhängige Elemente (z. B. Treiber) zu berücksichtigen.

Auf Hardwareseite sind Steuergeräte (Abbildung 15) zu modellieren. Diese Steuergeräte kommunizieren miteinander z. B. über verschiedene Busse; Steuergeräte und Busse bilden die Hardwaretopologie des Systems. Innerhalb der Steuergerä-

Abbildung 15 Steuergerät (Beispiel, Quelle: BMW [bmw]).



te sollen deren Bestandteile modelliert werden. Diese können von generischen Komponenten zu konkreten Mikrocontrollern reichen. Feinere Strukturen, z. B. das Innenleben von Mikrocontrollern, sind nur für Analysezwecke notwendig, das heißt, Performanceuntersuchungen sollen möglich sein, eigenes Design ist nicht erforderlich.

Damit ist die technische Architektur eine implementierungsnahe Modellierung. Trotzdem ist das Abstraktionsniveau der technischen Architektur hoch, das heißt, Software- oder Hardware-Systembeschreibungen können aus diesen Modellen nicht generiert werden. Das ist erst z. B. durch die Anbindung von AUTOSAR möglich.

Wichtig ist auch hier, dass die technische Architektur zunächst einmal unabhängig von der logischen Architektur entwickelt wird. Das ist natürlich kein Dogma, da die technische Architektur im Endeffekt die Funktionen realisieren soll. Die technische Architektur soll jedoch nicht einfach die Struktur der logischen Architektur übernehmen, sondern unabhängig nach technischen Gesichtspunkten modelliert werden, ebenso wie die logische Architektur nicht direkt aus der technischen Architektur entwickelt werden sollte. Die Zuordnung der Funktionen zur technischen Architektur erfolgt später bei der Partitionierung.

Umsetzung

Die technische Architektur besteht aus zwei Teilmodellen: der Software- und der Hardwarearchitektur. Die Softwarearchitektur ist sehr knapp gehalten, da hier nur eine Anbindung an AUTOSAR geschaffen wurde. Die Modellierung der Softwareanteile in AUTOSAR ist wesentlich detaillierter, für MOSES reicht die erreichte Abstraktionsebene aus. Das zentrale Element der Softwarearchitektur ist das Soft-

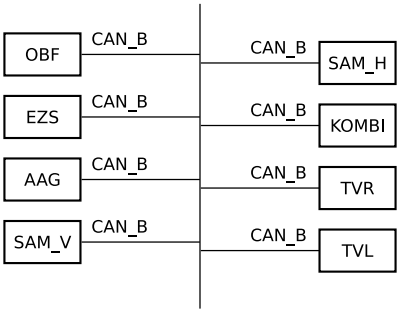
wareelement. Es kann in sich geschachtelt werden und enthält auf der untersten Ebene Routinen. Routinen sind die Softwareteile, die eine Funktion kapseln. Routinen können zu Tasks zusammengesetzt werden, die lauffähig sind.

Die Hardwarearchitektur wird aufwendiger modelliert. Die verschiedenen Aspekte der Modellierung: Topologie, Verbindungsstruktur, Steuergeräte sowie der Innenaufbau von Steuergeräten fordern ihren Tribut bei der Komplexität der Modellierung. Um die Modelle einfach zu halten, wurden generische Prinzipien eingesetzt, auf die für viele Modelle zurückgegriffen werden kann.

Eine Hardwarearchitektur wird hauptsächlich aus Hardwarekomponenten, inneren Elementen und Ports gebildet. Diese drei Elemente werden unter dem Begriff des Hardwareelements zusammengefasst. Die Hardware selbst wird aus Hardwarekomponenten und ihren inneren Elementen gebildet. Als Schnittstellen enthalten Hardwarekomponenten Ports, die miteinander verbunden werden können.

Die Hardwarearchitektur besteht also aus miteinander verbundenen Hardwarekomponenten. Hardwarekomponenten sind z. B. Steuergeräte, Aktoren, Sensoren oder Mikrocontroller. Die Verbindungen werden in Busse und andere Verbindungen unterschieden und können bis auf Pinebene spezifiziert werden. Solange man sich auf der Ebene der Hardwarekomponenten befindet und nur die Komponenten und ihre Verbindungen betrachtet werden, sieht man die Systemtopologie (Abbildung 16).

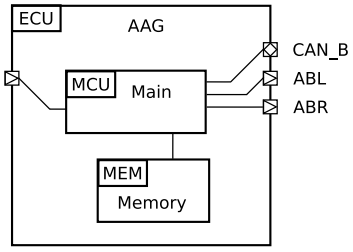
Abbildung 16 Systemtopologie (Beispiel).



Wird die innere Struktur einer Hardwarekomponente modelliert, kommen andere Aspekte zum Tragen. So können Hardwarekomponenten hierarchisch strukturiert sein, um komplexe Komponenten zu bilden. Außerdem werden auf dieser Ebene innere Elemente sichtbar, die nur innerhalb von Hardwarekomponenten auftreten können. Die innere Struktur dient der feineren Modellierung der Hardwarekomponente und vor allem der Bereitstellung von Attributen für Performancetests,

Ressourcenabschätzungen etc. Abbildung 17 zeigt eine modellierte Hardwarekomponente.

Abbildung 17 Hardwarekomponente (Beispiel).



Produktlinien

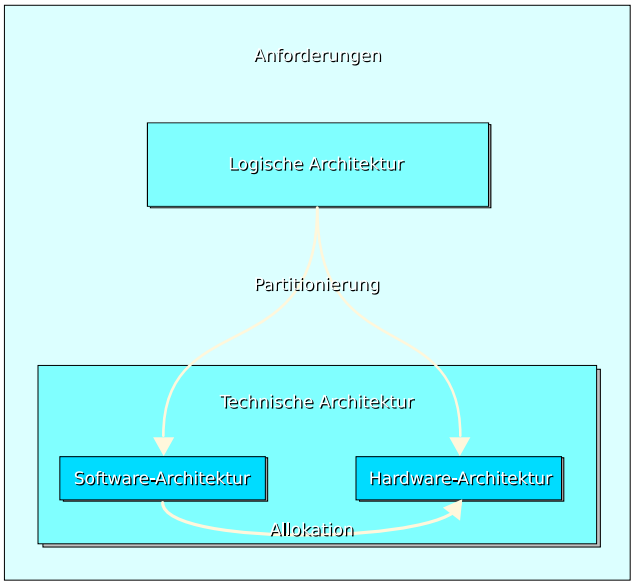
Produktlinien werden, wie bei der logischen Architektur, über das Konzept der Varianzpunkte eingebracht. Im Unterschied zur logischen Architektur sind nicht alle Elemente der technischen Architektur gleichwertig variabel. Das resultiert aus der zu erwartenden Komplexität bei freier Variabilität, die nicht mehr vernünftig handhabbar wäre.

Die Varianz wurde anhand des Gleich- und Ähnlichteilprinzips bestimmt. Dabei wird davon ausgegangen, dass den Hersteller bezüglich der technischen Umsetzung interessiert, welche Gleichteile zu entwerfen sind, welche Ähnlichteile vorkommen und welche Teile vollkommen unterschiedlich sind. Daraus wurde ein Varianzkonzept entwickelt, das genau diese Aspekte betont.

Hardwareelemente sind als Varianzpunkte zugelassen. Dabei sind Hardwarekomponenten voll variabel, Hardwareports nur im Rahmen der sie enthaltenden Hardwarekomponente. Die Pins und die zugehörigen Verbindungen können ebenfalls variiert werden, um unterschiedliche Anschlussmöglichkeiten einer Hardwarekomponente zuzulassen. Wichtig ist in diesem Zusammenhang, dass nicht nur die Elemente selbst variieren können, sondern auch ihre Attribute.

Die Ausleitung erfolgt wieder über logische Attribute, die den Varianten zugeordnet werden.

Abbildung 18 Partitionierung im Gesamtmodell.



3.4 Partitionierung

Zweck

Die Partitionierung (gelb in Abbildung 18) verbindet die logische mit der technischen Architektur und innerhalb der technischen die Software- mit der Hardwarearchitektur. Sie dient dazu, die verbundenen Modelle unabhängig voneinander modellieren zu können.

Die Partitionierung ist eine Abbildung. Im ersten Fall wird die logische Architektur auf die technische Architektur (Software oder Hardware) abgebildet. Wird eine Funktion z. B. auf eine Routine partitioniert, so bedeutet das, dass die Funktion durch diese Routine realisiert wird. Dabei können Funktionen auf mehrere technische Elemente abgebildet werden und mehrere Funktionen auf dasselbe Element. Das heißt, die Funktionen sind unabhängig von ihrer Realisierung, diese Beziehung wird durch die Partitionierung vorgegeben. Damit können die Funktionen unabhängig von ihrer Implementierung modelliert werden.

Der zweite Fall, die Abbildung der Software auf die Hardware, bedeutet, dass die partitionierte Software auf der Hardware ausgeführt wird. Auch hier können wieder eine oder mehrere Routinen auf eine oder mehrere Hardwarekomponenten

abgebildet werden. Damit sind auch die Softwareelemente unabhängig von der sie ausführenden Hardware modellierbar.

Umsetzung

Die Partitionierung wird durch Relationen zwischen den betroffenen Modellelementen realisiert. Dabei wird festgelegt, welche Elemente auf welche Ziele partitioniert werden können.

Im ersten Fall wird die logische auf die technische Architektur partitioniert. Für die Hardware bedeutet das: Funktionen können auf Hardwareelemente partitioniert werden. Die zu den Funktionen gehörenden Ports werden mit ihren Signalen auf ihre Gegenstücke, die Verbindungsenden abgebildet. Zu guter Letzt können auch die Verbindungen partitioniert werden, das ist notwendig, wenn die berechenbare Verbindung in der technischen Architektur nicht eindeutig ist. Die Software betreffend können Funktionen auf Routinen partitioniert werden.

Der zweite Fall, die Partitionierung der Software auf die Hardware, wird durch die Softwarepartitionierung realisiert. Dabei werden Softwareelemente auf Hardwarekomponenten abgebildet. Das ist natürlich nur für unabhängige Softwareelemente zulässig, hardwareabhängige Softwareelemente, wie Treiber, sind nicht frei partitionierbar. Es können auch Tasks auf Prozessoren abgebildet werden. Dabei gilt, dass Routinen und deren Tasks nur auf gleiche Hardwarekomponenten und Prozessoren partitioniert werden können.

Produktlinien

Für Produktlinien ist lediglich der Fall interessant, dass Funktionen auf unterschiedliche Hardwareelemente partitioniert werden. Damit können Funktionen je nach Fahrzeug auf unterschiedlicher Hardware ausgeführt werden. Dem folgend müssen die zugehörigen Routinen ebenfalls auf unterschiedlicher Hardware laufen, diese Varianz folgt aber aus der Funktionspartitionierung und wird nicht eigenständig modelliert.

4 Benutzung

Nachdem die Architekturmodelle vorgestellt wurden, bleibt die Frage zu klären, wie die Architekturmodelle erstellt werden. Solche Aspekte werden meist unter dem Punkt »Methode« oder »Prozess« subsumiert. In MOSES wurde jedoch kein formaler Prozess definiert, es sind lediglich Handlungsanweisungen und *best practices* bekannt, die zu MOSES-Modellen führen.

Das liegt zum einen daran, dass der Fokus des MOSES-Projekts zunächst auf den Modellen lag. Prozessaspekte wurden bewusst ausgeklammert oder nur angerissen. Zum anderen weichen vorgegebene Prozesse oft dann von der Wirklichkeit ab, wenn z. B. Vorwissen berücksichtigt werden muss. Das verkompliziert die Definition eines Vorgehens derart, dass ein MOSES-Prozess in einem Folgeprojekt definiert wird.

Trotzdem gibt es Vorstellungen, wie ein MOSES-Prozess aussehen sollte, ohne eine formale Definition dieses Prozesses anzubieten. Gleiches gilt für die Umsetzung des MOSES-Prozesses in die Praxis, auch hier gibt es Vorstellungen, die noch nicht formalisiert wurden.

4.1 MOSES-Prozesse

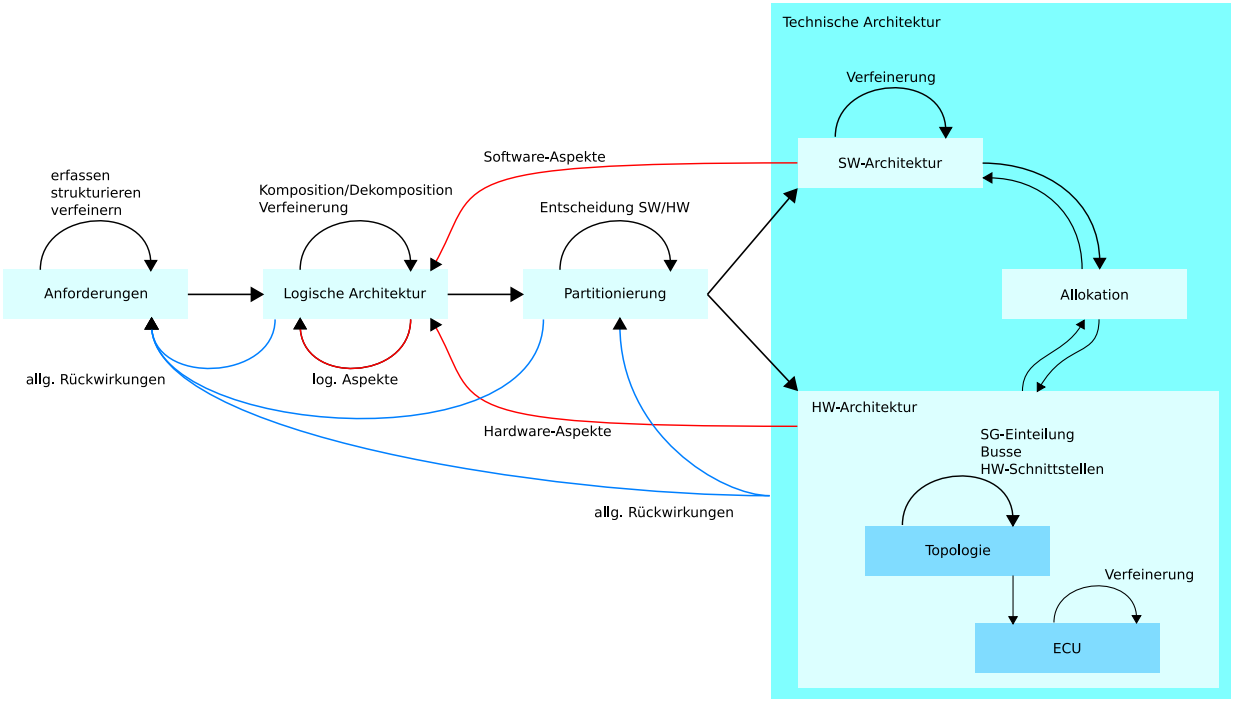
In diesem Abschnitt werden drei typische Vorgehensweisen kurz vorgestellt, die im Umgang mit MOSES-Modellen auftreten, ausführlich werden sie in Teil V beschrieben:

1. der MOSES-Prozess
2. Berücksichtigung von Varianz
3. Aufarbeitung von Domänenwissen

Der MOSES-Prozess

Der MOSES-Prozess, der in Abbildung 19 skizziert wird, beginnt mit der Anforderungsmodellierung. Dabei werden alle relevanten Stakeholder mit den üblichen Methoden der Anforderungserfassung befragt und die Ergebnisse strukturiert im Anforderungsmodell festgehalten.

Abbildung 19 Skizze des MOSES-Prozesses.



Danach wird aus den Anforderungen das logische Modell entwickelt. Dabei werden zunächst die Kundenanforderungen als Grundlage der Hauptfunktionalitäten genommen und die Funktionen mit Hilfe der Systemanforderungen soweit verfeinert, dass die technische Architektur entworfen werden kann. In dieser Phase werden logische Aspekte wie Sicherheit berücksichtigt und das Modell daraufhin verfeinert.

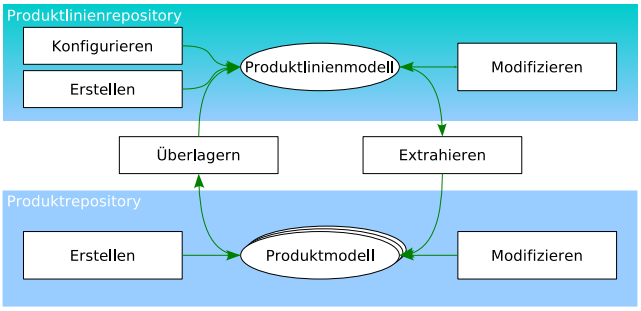
Im nächsten Schritt muss für den Entwurf der technischen Architektur zunächst geklärt werden, welche Funktionen in Software und welche in Hardware umgesetzt werden sollen. Es findet eine erste Grobpartitionierung statt. Danach wird die Software entworfen und die zugehörige Hardware ebenfalls. Damit ist der erste Entwurf des Systems vom Anforderungsmodell zur Partitionierung fertig.

Dieser erste Entwurf wird durch implementierungsspezifische Aspekte wie Redundanz, Signalwandlung etc. verfeinert. Diese Aspekte verändern sowohl die logische als auch die technische Architektur und, daraus folgend, die Partitionierung. Damit ist das MOSES-Modell vollständig beschrieben und kann implementiert oder in ein AUTOSAR-Modell überführt werden.

Berücksichtigung von Varianz

Die Berücksichtigung von Varianz kann in allen Modellen in jedem Schritt stattfinden. Sie ändert nicht den Prozess an sich, sondern fügt zusätzliche Schritte ein, die für die Varianz berücksichtigt werden müssen.

Abbildung 20 Produktlinienprozess.



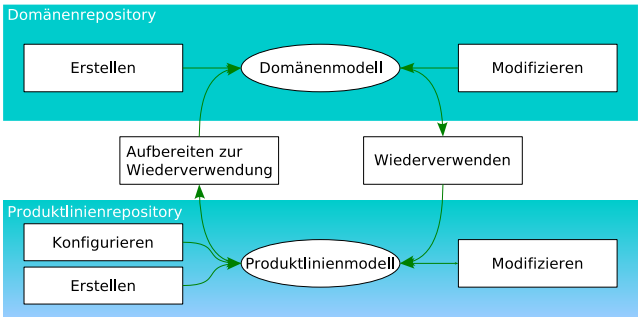
Diese Schritte führen, wie in Abbildung 20 zu sehen ist, von Produktmodellen zu Produktlinienmodellen. Das kann man sich so vorstellen, dass mehrere Produktmodelle überlagert werden und die entstehende Varianz notiert wird. Dazu muss festgelegt werden, an welcher Stelle ein Varianzpunkt eingeführt wird. Dann wird entschieden, welche Art von Varianz vorliegt, z. B. Optionalität, Auswahlvarianz oder Attributvarianz. Jetzt müssen die Ausprägungen der Varianz beschrieben werden. Diese Beschreibung kann auch später erfolgen, dann sollte die entsprechende Stelle gekennzeichnet werden. Danach muss festgelegt werden, wann die Varianz aufgelöst werden soll: bei der Ausleitung, am Bandende etc. Diese Schritte werden als Konfiguration bezeichnet.

Im letzten Schritt der Konfiguration muss der Varianzpunkt annotiert werden. Das heißt, für jede Variante wird festgelegt, unter welchen Bedingungen sie im Fahrzeug auftritt. Diese Festlegung wird später bei der Ausleitung (beim Extrahieren) dafür benutzt, das konkrete Fahrzeug aus dem Produktlinienmodell zu generieren (auszuleiten).

Aufarbeitung von Domänenwissen

Die Aufarbeitung und Nutzung von Domänenwissen ist ebenfalls ein ergänzender Prozess zum Entwurfsprozess. Dementsprechend müssen eigene Rollen und Verantwortlichkeiten für diesen Prozess definiert werden. Der Prozess dient dazu, Produktwissen in sogenanntes Domänenwissen zu überführen, das langlebig ist und Entwurfsentscheidungen, Bausteine oder Musterlösungen enthält.

Abbildung 21 Domänenprozess.



Wir arbeiten uns an Abbildung 21 von unten nach oben und zurück. Im Produktlinienmodell wird Domänenwissen identifiziert und in ein Zwischenmodell überführt. Dort wird das Wissen aufbereitet, so dass es den Ansprüchen des Domänenrepositories genügt. Dann wird das Wissen in die Domäne übernommen.

Soll das Domänenwissen in einem Produktlinienmodell verwendet werden, wird es aus der Domäne wieder in ein Zwischenmodell geholt. Dort werden die produktlinienspezifischen Anpassungen vorgenommen, die im Domänenmodell vorgesehen waren. Danach wird das Zwischenmodell in das Produktlinienmodell überführt und kann dort normal weiterbearbeitet werden.

Dieser Prozess ist aufwendig und wegen dieses Aufwands erst bei großen Produktfamilien sinnvoll. Dann kann aber über geschickte Verwendungs- und Bausteinkonzepte ein hoher Grad an Wiederverwendung und Qualität erreicht werden.

4.2 Umsetzung in die Praxis

Die Umsetzung von MOSES in die Praxis muss verschiedene Faktoren berücksichtigen. Zunächst werden in MOSES Modelle beschrieben. Damit diese Modelle produktiv genutzt werden können, müssen entsprechende Werkzeuge eingesetzt werden. Die Personen, die mit den Werkzeugen und Modellen arbeiten sollen, müssen geschult und unterstützt werden.

MOSES baut stark auf dem Rollenprinzip für die Benutzer der Modelle auf. So sind z. B. Entwickler mit der Entwicklung von logischen oder technischen Architekturmodellen beschäftigt, während die Domänenpflege vom Domänenverwalter

übernommen wird. Diese Rollen sind spezifisch für den Anwender zu definieren und im laufenden Prozess umzusetzen.

Die nahtlose Integration von MOSES in bestehende Prozesse durch die vollständige, gleichzeitige Umsetzung des neuen Prozesses, neuer Rollen und neuer Modelle ist in der Praxis illusorisch. Meist wird nur ein Teil der Möglichkeiten umgesetzt und etabliert, bevor nachfolgende Prozesse und Modelle aufgesetzt werden.

Diese Vielfalt der Schwierigkeiten lässt erkennen, dass es für die Umsetzung der MOSES-Methode in die Praxis kein Patentrezept gibt. Die Einführung ist für jede Firma anders, sie hängt von dem bestehenden Prozess ab oder vom Umfang der Änderungen, die vorgenommen werden sollen.

Trotzdem können grobe Erfahrungswerte gegeben werden: Da die Anforderungen Basis für alle folgenden Modelle sind, ist es ratsam, die Anforderungsmodellierung als erstes in die Praxis umzusetzen. Sofern bereits ein Funktionsbegriff etabliert ist, ist die logische Architekturmodellierung als nächstes einzusetzen. Gleichzeitig kann die Hardwarearchitektur eingeführt werden. Die Umsetzung von Softwarearchitektur und Partitionierung ist aufgrund der neuen Sichtweise das komplizierteste Thema, das erst nach Etablierung der logischen Architekturmodellierung angegangen werden sollte.

Bezüglich der Produktlinien/Domänenproblematik kann gesagt werden, dass Produktlinien so früh wie möglich eingeführt werden müssen, um den Erfolg der Methode zu sichern. Gerade der Zusatznutzen, der aus Produktlinien erwächst, rechtfertigt die Investitionen von Zeit und Geld in die MOSES-Methode. Mit der Einführung von Domänen sollte solange gewartet werden, bis sich die Methode gefestigt und in den Produktentstehungsprozess eingebunden hat, da Domänen eigene Verwalter brauchen und die Erfahrung über das Domänenwissen erst mit fortschreitender Nutzung der Methode gebildet wird.

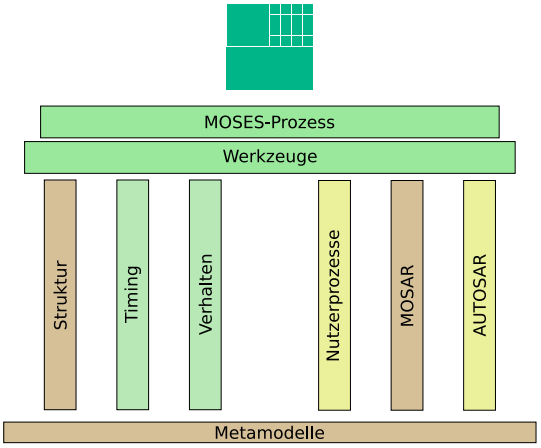
Zusätzlich zu den Entwicklungsprozessen sind auch die entwicklungsbegleitenden Prozesse Projektmanagement, Modellmanagement, Versionierung etc. zu betrachten. Sie müssen für die Einführung einer neuen Methodik sauber definiert und praktisch gelebt werden. Die Erfahrung zeigt, dass unter diesen Voraussetzungen der Umstieg auf MOSES im laufenden Betrieb möglich ist und positiv bewertet wurde.

5 Ausblick

Nach der kurzen Einführung in MOSES sollen die wichtigsten Punkte noch einmal zusammengefasst und ein Ausblick gegeben werden, welche Entwicklungen von MOSES noch zu erwarten sind. Abbildung 22 zeigt dabei die im Folgenden besprochenen Modelle.

Wie erwähnt, sind in MOSES bisher hauptsächlich Metamodelle für die einzelnen Architekturmodelle definiert worden. Die dabei getroffene Auswahl der Modelle orientierte sich an praktischen Gegebenheiten und deckt die E/E-Systementwicklung im Automobilbau recht gut ab. Aspekte wie Deployment oder Geometrie wurden nicht berücksichtigt.

Abbildung 22 MOSES Roadmap.



Dabei wurde Wert auf abstrakte Modellierung gelegt, MOSES-Modelle sind Verständnismodelle, keine Generierungsmodelle. Die Anbindung an Generierungsmodelle wurde beispielhaft für die Software durchgeführt. Hier wird AUTOSAR als Modellierungsmittel für die Generierung gesehen und die Softwarebeschreibung in MOSES so gewählt, dass die Überführung in AUTOSAR-Softwarekomponenten möglichst einfach ist. Damit einhergehend wurden die Modelle der Hardwarearchitektur und der Partitionierung ebenfalls in Richtung AUTOSAR angepasst. Der Übergang zwischen MOSES und AUTOSAR wird im MOSAR-Projekt vorgenommen, das zur Zeit durchgeführt wird.

Die fehlenden Aspekte Deployment, Einbau und andere Sichtweisen werden dar-

aufhin überprüft, ob eine Erweiterung von MOSES nötig ist oder die Definition des Übergangs zwischen MOSES und anderen Modellierungstechniken für eine adäquate Modellierung ausreicht. Je nach Resultat dieser Überprüfung wird MOSES erweitert oder ein Übergang definiert werden.

Ein weiterer Teilaspekt sind die Zeiteigenschaften der modellierten Systeme, die im Timingprojekt untersucht werden. Hier werden die Echtzeiteigenschaften des Systems in die Modellierung einbezogen, um Echtzeitfragestellungen des Systems beantworten zu können.

Damit einhergehend werden Fragen des Systemverhaltens in MOSES bisher nur angerissen. Die Erweiterung der Verhaltensmodellierung ist im Zusammenhang mit dem Timingprojekt geplant. Dabei wird auf bestehende Verhaltensmodelle zurückgegriffen werden, die in die MOSES-Modelle integriert werden.

Der MOSES-Prozess ist ein Punkt, der ebenfalls in naher Zukunft definiert wird. Die Grundkonzepte und Vorgehensweisen wurden bereits skizziert, es muss nun ein konsistentes Prozessmodell entwickelt werden. Da sich Nutzer des *Rational Unified Process* (RUP), des Wasserfallmodells und anderer Prozessmodelle möglichst wiederfinden sollen, ist diese Prozessdefinition noch nicht abgeschlossen.

Das Thema Werkzeuge wurde in MOSES prototypisch behandelt. Die Anforderungsmodellierung kann mit Standardwerkzeugen (z. B. DOORS) vorgenommen werden. Diese Werkzeuge müssen teilweise erweitert werden, so können die Konfigurationsinformationen nicht ohne weiteres in DOORS nachgebildet werden. Das Fraunhofer ISST hat ein Plugin dafür geschrieben, mit dem die MOSES-Anforderungsmodellierung ohne Abstriche umsetzbar ist. Ein weiterer Prototyp, der speziell für Anforderungsmodellierung mit MOSES geeignet ist, wird programmiert. Für die Modelle der logischen und technischen Architektur sowie der Partitionierung wurde ein Werkzeug gefertigt, an dem die Umsetzbarkeit der Methode gezeigt werden konnte.

MOSES ist für den Automobilbau entwickelt worden. In diesem Einsatzbereich liegen die Stärken des Modells, da die speziellen Anforderungen des Automobilbaus an eingebettete Echtzeitsysteme mit verteilter Funktionalität explizit berücksichtigt wurden. MOSES ist im Kern jedoch flexibel. Die Anpassung an andere Echtzeitbereiche außerhalb des Automobilbaus ist möglich. Ein Schritt in diese Richtung ist bereits mit der Definition eines MOSES-Kernels vorgenommen worden.

Zusammenfassung

Zusammenfassend ist MOSES im Automobilbereich gut geeignet, um von Anforderungen zu einer abstrakten Systemsicht zu kommen. Das Ziel ist Systemverständnis sowie die Schaffung eines Anknüpfungspunkts für Generierungsmodelle. Diese Anbindung an Generierungsmodelle ist bereits umgesetzt oder geplant. Die explizite Modellierung von Produktlinien und Domänenwissen berücksichtigt die reale Situation in laufenden Produktionsprozessen. MOSES kann auch außerhalb des Automobilbereichs eingesetzt werden, dafür sind jedoch Anpassungen nötig. Die Arbeit an MOSES ist noch nicht abgeschlossen, die bereits fertigen Teile sind praxiserprobt.

Teil II

Modelle (Artefakte) im Detail

Nach den allgemeinen Bemerkungen von Teil I werden in diesem Teil die MOSES-Modelle im Detail vorgestellt. Dabei werden die Modelle anhand von Beispielen vorgestellt, die Metamodelle selbst werden nicht explizit aufgeführt.

Dieser Teil stellt die Modelle für Applikationen, also einzelne Fahrzeuge vor. Aspekte wie Varianz und Redundanz werden zunächst ausgeklammert, sie werden im Teil III gesondert behandelt.

Die vorgestellten Modelle für die jeweiligen Sichten sind aus Teil I bekannt: das Anforderungsmodell, die logische Architektur, die technische Architektur und die Partitionierung.

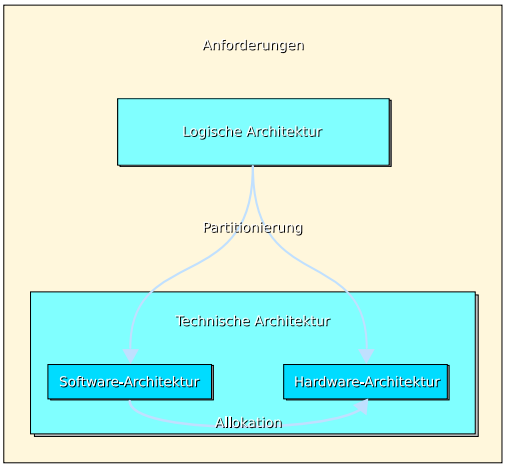
Ein kurzer Ausflug in die MOSES-Metamodellierung sei erlaubt: MOSES definiert ein Metamodell, in dem die MOSES-Modellelemente formal beschrieben werden. Die vier erwähnten Sichten sind Teile dieses Gesamtmetamodells. Zusätzlich gibt es einen fünften Teil, den MOSES-Kern. Er enthält die Kernkonzepte wie Attributierung von Elementen oder die Zerlegung von Teilmodellen in Modelle. Diese Kernkonzepte werden in jedem Architekturmodell aufgegriffen und sind deshalb zentral definiert worden. Das Gesamtmodell ist in [Fra05a] nachzulesen.

6 Anforderungsmodell

Das Anforderungsmodell (gelb in Abbildung 23) beschreibt, wie Anforderungen erfasst werden, in welche Struktur sie eingebettet werden und die Art, wie Konfigurationsinformationen an Anforderungen annotiert werden. Das Anforderungsmodell ist im Rahmen von MOSES 1 erstellt worden und ist in [Fra02] beschrieben.

Im Folgenden wird der Anforderungsbegriff kurz erläutert, danach wird beschrieben, auf welche Weise Anforderungen im MOSES-Rahmen repräsentiert und genutzt werden.

Abbildung 23 Anforderungen.



6.1 Anforderungen

Durch Anforderungen wird beschrieben, was von einem System oder von einer Familie von Systemen erwartet wird. Gegenstand einer Anforderung, also das, was angefordert wird, kann jegliche Funktion, Eigenschaft oder Einrichtung sein, die als Merkmal für den Nutzer des Systems von Belang ist. Zum Beispiel:

- Funktion: Aktivieren der Fahrtrichtungsanzeiger
- Eigenschaft: Frequenz des Blinktakts
- Einrichtung: Lenkstockhebel

Zur Verbesserung der Verständlichkeit und Eindeutigkeit von Anforderungen werden diese stets durch ganze Sätze beschrieben, wie z. B.:

- Die Fahrtrichtungsanzeiger werden durch Betätigen des Lenkstockhebels aktiviert.
- Die Blinkfrequenz beträgt 1,5 Hz.
- Ein Lenkstockhebel ist vorhanden.

Dies ist insbesondere dann wichtig, wenn komplexe und detailreiche Anforderungsdokumente auch von Personen genutzt werden sollen, die nicht an ihrer Formulierung beteiligt waren und den genauen Entstehungszusammenhang nicht kennen.

Zusätzlich zum Gegenstand der Anforderung (dem »Was?«) werden noch weitere Kontextinformationen festgehalten:

- Wie?: Auf welche Weise wird es angefordert? Muss, soll oder kann es vorhanden sein?
- Wer?: Wer fordert das Merkmal?
- Wovon oder von wem?: Von welchem System oder welcher Familie von Systemen wird das Merkmal angefordert, bzw. von wem wird verlangt, das Merkmal zu realisieren (wer ist für die Realisierung verantwortlich)?

6.2 Kunden- und Systemanforderungen

Für die Anforderungsmodellierung werden Kunden- und Systemanforderungen unterschieden, die den Rollen »Kunde« bzw. »Entwickler« zugeordnet sind. Kundenanforderungen werden vom Kunden gestellt, Systemanforderungen werden an den Entwickler gestellt. Die Rolle des Kunden nehmen z. B. die Käufer von Fahrzeugen ein. Aber auch andere Abnehmer oder Interessenten (*stakeholder*) wie z. B. die Marketing-Abteilung, die das System verwenden aber nicht selbst entwickeln, sind in diesem Sinne Kunden.

Die Zuordnung von Anforderungen zu Rollen ermöglicht neben der inhaltlichen Trennung auch eine Zuweisung von Verantwortlichkeiten für die Erstellung und Verwaltung von Anforderungen.

Kundenanforderungen

Die Kundenanforderungen beschreiben die erlebbare Funktionalität, d. h. diejenigen Merkmale, die ein Kunde von dem Fahrzeug erwartet:

- Die Fahrtrichtungsanzeiger müssen einfach und komfortabel aktiviert werden.
- Eine Komfortblinkfunktion für Kurzblinken ist vorhanden.
- Ein Leuchtmittelausfall wird signalisiert.

Systemanforderungen

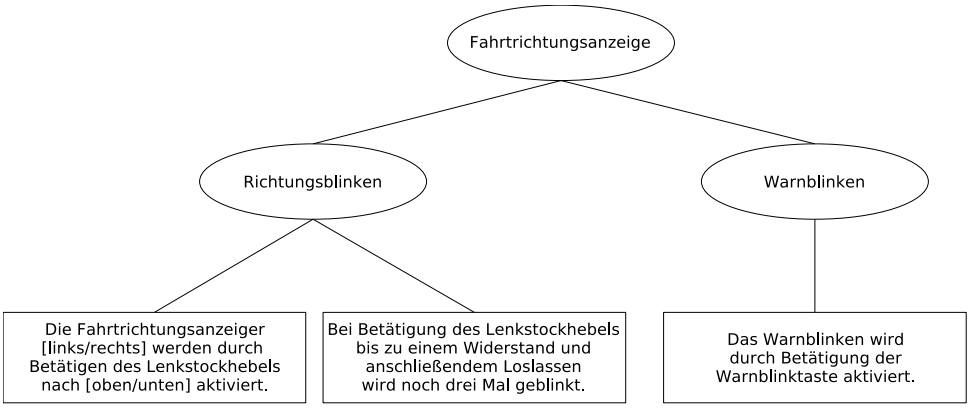
Die Systemanforderungen beschreiben, was das Fahrzeug bzw. System können soll, das heißt, was von den Entwicklern verlangt wird. Im Unterschied zu Kundenanforderungen können hier technische Gegebenheiten wichtig sein:

- Die Fahrtrichtungsanzeiger [links/rechts] werden durch Betätigen des Lenkstockhebels nach [oben/unten] aktiviert.
- Die Funktion der Fahrtrichtungsanzeiger wird dem Fahrer optisch und akustisch angezeigt.
- Bei Betätigung des Lenkstockhebels bis zu einem Widerstand und anschließendem Loslassen wird noch drei Mal geblinkt.
- Bei Ausfall einer Leuchte wird der Blinktakt erhöht und eine Warnung im Informationsdisplay ausgegeben.
- Bei Ausfall einer hinteren Leuchte wird die Funktion des Blinkens von der entsprechenden Bremsleuchte übernommen.

Die Trennung zwischen Kunden- und Systemanforderungen ist nicht immer eindeutig, so können z. B. auch Kundenanforderungen technische Details enthalten, z. B. weiße Blinker statt der gelben Originalblinker.

6.3 Strukturierung von Anforderungen

Abbildung 24 Anforderungsbaum (Beispiel).



Um eine übersichtliche Darstellung der Anforderungen zu erhalten und um Anforderungen schneller lokalisieren und wiederfinden zu können, werden sowohl Kunden- als auch Systemanforderungen hierarchisch strukturiert. Dazu werden Oberbegriffe eingeführt, anhand derer die Anforderungen klassifiziert werden können. Das ergibt eine Baumstruktur, an deren Blattknoten die Anforderungen stehen. Alle anderen Knoten sind Strukturierungsknoten, die selbst keine Anfor-

derungen enthalten, sondern nur die Oberbegriffe.[2] Abbildung 24 zeigt ein Beispiel für einen Anforderungsbaum.

In der kompakteren Tabellendarstellung eines Anforderungsbaums wird die Strukturierung durch die Nummerierung repräsentiert, wie z. B. in Tabelle 3. Die grafische Baumdarstellung dient nur der Erläuterung der Konzepte und der Visualisierung der Modellstrukturen anhand kleiner Beispiele, für reale Anwendungen mit entsprechenden Werkzeugen ist die Tabellendarstellung geeigneter.

Tabelle 3 Anforderungstabelle (Beispiel).

1	Fahrtrichtungsanzeige
1.1	Richtungsblinken
1.1.1	Die Fahrtrichtungsanzeiger [links/rechts] werden durch Betätigen des Lenkstockhebels nach [oben/unten] aktiviert.
1.1.2	Bei Betätigung des Lenkstockhebels bis zu einem Widerstand und anschließendem Loslassen wird noch drei Mal geblinkt.
1.2	Warnblinken
1.2.1	Das Warnblinken wird durch Betätigung der Warnblinktaste aktiviert.

6.4 Verbindung zwischen Kunden- und Systemanforderungen – Konkretisierung

Die Anforderungen für die Systementwickler sind in den Systemanforderungen festgelegt, während die für den Kunden bzw. den Verkauf des Systems relevanten Merkmale in den Kundenanforderungen beschrieben sind. Durch den Entwicklungsprozess muss also sichergestellt werden, dass die Kundenanforderungen durch die Systemanforderungen auch vollständig abgedeckt werden. Dazu wird im Modell eine Konkretisierungsrelation zwischen Kunden- und Systemanforderungen angegeben.

Die Semantik der Konkretisierungsrelation ist zunächst noch sehr weit gefasst: Eine Systemanforderung konkretisiert eine Kundenanforderung, wenn das entsprechende Systemmerkmal einen – wie immer gearteten – Beitrag zur Realisierung der Kundenanforderung liefert.

[2] Diese Anforderungsbäume entsprechen den in der *Feature Oriented Domain Analysis* benutzten *feature trees*.

Eine Systemanforderung kann zur Konkretisierung von mehreren Kundenanforderungen beitragen und umgekehrt können mehrere Systemanforderungen zur Konkretisierung einer Kundenanforderung nötig sein.

Tabelle 4 Konkretisierung (Beispiel).

Kundenanforderung	Konkretisierende Systemanforderung(en)
Die Fahrtrichtungsanzeiger müssen einfach und komfortabel aktiviert werden.	Die Fahrtrichtungsanzeiger [links/rechts] werden durch Betätigen des Lenkstockhebels nach [oben/unten] aktiviert.
Ein Leuchtmittelausfall wird signalisiert.	Bei Ausfall einer Leuchte wird der Blinktakt erhöht und eine Warnung im Informationsdisplay ausgegeben. Bei Ausfall einer hinteren Leuchte wird die Funktion des Blinkens von der entsprechenden Bremsleuchte übernommen.

6.5 Verfolgbarkeit – *traceability*

Die Konkretisierungsrelation dient zusätzlich dem Zweck der Verfolgbarkeit. Verfolgbarkeit (*traceability*) bedeutet, dass alle Modellelemente so untereinander verknüpft sind, dass klar ist, welches Modellelement mit welchem Element eines anderen Modells zusammenhängt.

Für das Anforderungsmodell muss nachvollzogen werden können, welche Kundenanforderung durch welche Systemanforderung realisiert wird. Das wird durch die Konkretisierungsrelation geleistet. Damit ist es möglich, die Auswirkungen (*impact*) einer Änderung nach jeder Seite zu verfolgen.

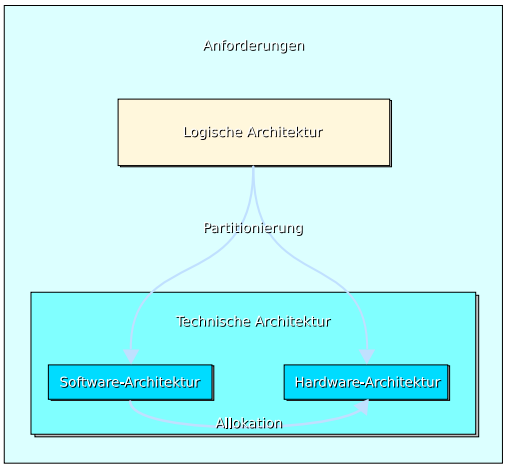
Wie bereits erwähnt, ist die Verfolgbarkeit auch zwischen den Modellen wichtig. So muss z. B. klar sein, welche Funktion auf Grund welcher Anforderung modelliert wurde.

7 Logische Architektur

Die logische Architektur (gelb in Abbildung 25) beschreibt die Funktionalität des Systems, ohne auf technische Details einzugehen. Die logische Architektur besteht aus hierarchischen Funktionen, die in ein Funktionsnetzwerk eingebunden werden und so miteinander kommunizieren können. Die logische Architektur wird in [Fra03a] beschrieben.

Im Folgenden werden Funktionen als Elemente der logischen Architektur vorgestellt. Dann wird auf die Hierarchisierung und Verbindung von Funktionen eingegangen. Die Beschreibung von Verhalten und das Fehlermodell bilden den Abschluss des Kapitels.

Abbildung 25 Die logische Architektur.



7.1 Funktionen

Sowohl im Entwicklungs- als auch Integrationsprozess eines Systems ist es von entscheidender Bedeutung, die Modellierung der sich aus den Anforderungen ergebenden Funktionalität bzw. des gewünschten Verhaltens (des *Was?*) von der Realisierung dieser Funktionalität (des *Wie?*) zu trennen. Erst dadurch können logisch notwendige von implementierungsbedingten Entwurfsentscheidungen unterschieden und getrennt modelliert werden.

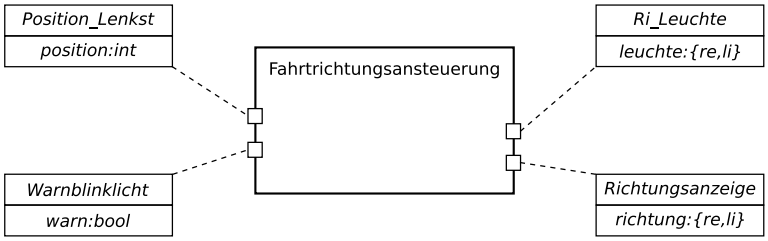
Ist die logische Funktionalität explizit festgehalten, können unterschiedliche Realisierungen entwickelt und eingesetzt werden. Ein gut strukturiertes Modell unterstützt dabei auch den Austausch von Teilfunktionalitäten des gesamten Systems, was insbesondere für die Integration von entscheidender Bedeutung ist.

Funktionen

Eine Funktion besteht aus einem Funktionskörper und einer Schnittstelle nach außen. Diese Schnittstelle wird durch *Ports* und deren *Interfaces* gebildet. Da jegliche Kommunikation mit der Funktion nur über ihre Schnittstelle erfolgen darf, wird das Innere der Funktion nach außen hin verborgen, die Funktion wird gekapselt. Ein Beispiel für eine Funktion zeigt Abbildung 26.

Diese Kapselung dient mehreren Zwecken: Erstens ist es möglich, Funktionen auszutauschen, sofern sie gleiche Schnittstellen anbieten. Zweitens kann die innere Struktur einer Funktion geändert werden, ohne dass die Änderungen über die Funktionsgrenzen hinaus sichtbar werden (vorausgesetzt, die Schnittstelle der Funktion ändert sich nicht). Drittens wird definiert, welche Umgebung die Funktion benötigt, um zu funktionieren, ein Vorteil, den z. B. objektorientierte Modellierungsmethoden im Allgemeinen nicht bieten.

Abbildung 26 Funktion (Beispiel).



Damit entspricht eine Funktion einer Komponente als abgeschlossene Einheit, die über Schnittstellen mit ihrer Umwelt kommuniziert. Dabei bietet sie Schnittstellen an den Ausgängen an (*provided*) oder fordert andere Schnittstellen an den Eingängen (*required*). Dieses Komponentenmodell entspricht in der Idee einem allgemeinen Komponentenbegriff, der u. a. in der UML 2.0 Einzug gefunden hat.

Ports

Ports sind Interaktionspunkte, über die eine Funktion mit ihrer Umgebung kommunizieren kann. Sie werden als kleine Quadrate gezeichnet, die sich auf dem Rand einer Funktion befinden. Sie können Signale oder Operationen anbieten

bzw. benötigen. Die folgenden Ausführungen über Signale gelten analog für Operationen.

Mit Ports lassen sich die Kommunikationsbeziehungen einer Funktion von ihrem aktuellen Kontext lösen. Anstatt anzugeben, von welcher Funktion ein Signal empfangen wird oder an welche Funktion ein Signal geschickt wird, werden nur die entsprechenden Ports angegeben, die lokal zu der Funktion gehören. Die Verbindung von Funktionen wird ausschließlich über die Verbindung ihrer Ports realisiert. Dadurch kann eine Funktion z. B. später aus ihrem Kontext herausgeschnitten und an anderen Stellen wieder verwendet werden; sie kann auch durch eine andere Funktion ersetzt werden.

Jeder Port ist entweder ein Eingangs- oder ein Ausgangsport. Über einen Eingangsport können Signale an die Funktion geschickt und damit ihre Dienste angefordert werden, über einen Ausgangsport kann die Funktion Signale verschicken und Dienste anderer Funktionen anfordern. Ein Port, der Signale verschickt, diese also anbietet, wird als *provided port* bezeichnet, ein Port, der Signale empfängt, die für die Funktion notwendig sind, als *required port*. Welche Signale an einen Port geschickt bzw. von ihm verschickt werden können, wird durch die mit einem Port assoziierten *Interfaces* spezifiziert.

Bei der Strukturierung der Ports ist darauf zu achten, dass bei einer Verbindung zweier Funktionen alle Signale, die an einem Eingangsport erwartet werden, auch gesendet werden können. Dabei spielt es keine Rolle, von wievielen Ausgangsports die Signale gesendet werden, wichtig ist nur, dass die Beziehung von Sender zu Signal eindeutig ist.

Interfaces

Interfaces werden Ports zugewiesen und enthalten eine Menge von Signalen. Damit kann festgelegt werden, welche Signale ein Port sendet bzw. empfängt. In einem *Interface* sind die Namen der Signale sowie deren formale Parameter mit ihrem Typ aufgelistet.

Ein Port kann mehrere *Interfaces* erhalten, dann verarbeitet er alle Signale, die in diesen *Interfaces* enthalten sind. Die Zuordnung von Signalen zu *Interfaces* dient der Strukturierung der Signale. Signale sollten dann in einem *Interface* zusammengefasst werden, wenn sie inhaltlich zusammengehören und davon ausgegangen werden kann, dass entweder alle Signale in diesem *Interface* für eine Kommunikation relevant sind oder keins. *Interfaces* können bei Bedarf durch Spezialisierungshierarchien strukturiert werden.

Signale

Signale werden in permanente und transiente Signale unterschieden. Permanente Signale besitzen ihren Wert permanent, das heißt, der Wert des Signals liegt jederzeit vor. Permanente Signale modellieren hauptsächlich den Datenfluss eines Systems, d. h. die Bereitstellung von Werten für die empfangende Funktion. Sie werden daher auch als »Datensignale« bezeichnet. Dabei wird während der Modellierung kein konkretes Zeitmodell unterstellt.

Transiente Signale sind flüchtig, ihr Auftreten löst in der empfangenden Funktion bestimmte Reaktionen aus. Daher werden transiente Signale vorrangig zur Modellierung des Kontrollflusses benutzt, sie werden auch als »Steuerungssignale« bezeichnet.

Innerhalb der Modelle erfolgt die Kennzeichnung der Signale durch eine entsprechende Stereotypisierung: «*permanent*» für permanente Signale, «*transient*» für transiente.

Operationen

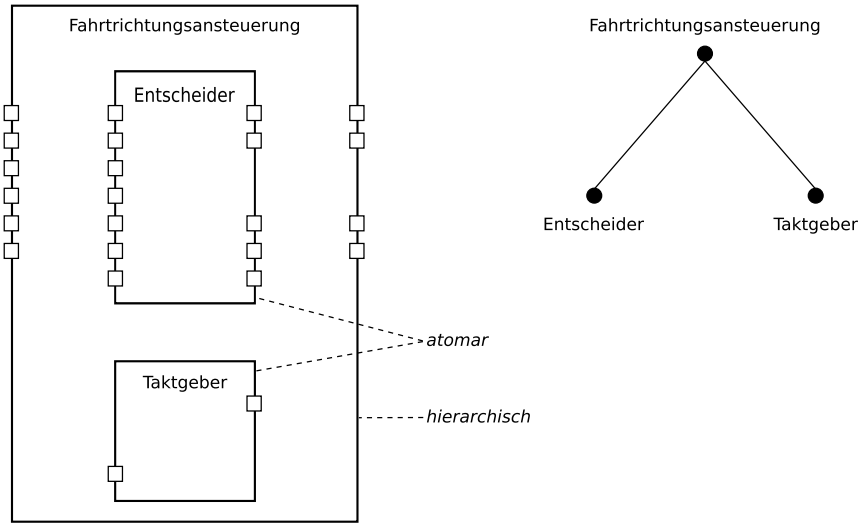
Ebenso wie Signale können Ports auch Operationen anbieten bzw. verlangen. Die Operation wird mit einem Namen sowie ihren Eingangs- und Ausgangsparametern beschrieben. Nach einem Operationsaufruf werden die Ausgangsparameter über den gleichen Port empfangen bzw. gesendet, über den der Aufruf angefordert bzw. empfangen wurde.

7.2 Hierarchisierung von Funktionen

Bisher wurde die äußere Struktur einer Funktion beschrieben und durch ihre Schnittstelle vollständig definiert. Jetzt wird die innere Struktur einer Funktion betrachtet: Eine Funktion ist entweder leer oder enthält weitere Funktionen, deren Zusammenspiel die Gesamtfunktion ausmacht. Eine leere Funktion, die damit keine innere Struktur, sondern nur noch Verhalten besitzt, wird als *atomare* Funktion bezeichnet. Eine Funktion mit Unterfunktionen ist eine *hierarchische* Funktion, da sie mit ihren Unterfunktionen eine Hierarchie bildet.

In Abbildung 27 sind die Funktionen *Entscheider* und *Taktgeber* atomare Funktionen. Die Funktion *Fahrtrichtungsanzeiger*, die beide atomaren Funktionen enthält, ist eine hierarchische Funktion.

Abbildung 27 Funktionen: atomar und hierarchisch.



Die bereits erwähnte Kapselung der Funktionen führt dazu, dass die hierarchische Verfeinerung – die Dekomposition – einer Funktion auf ihr Inneres beschränkt bleibt. Solange die Schnittstelle der Funktion unverändert bleibt, hat sich von außen nichts geändert. Damit können im Entwurfsprozess Funktionen grob über ihre Schnittstellen beschrieben und dann in späteren Verfeinerungsschritten dekomponiert werden.

Die Kommunikation zwischen Funktionen gleicher Ebene geschieht über Verbindungen zwischen ihnen. Die Kommunikation mit der umschließenden Funktion erfolgt über deren Ports. Damit bleibt auch bei hierarchischer Verfeinerung die Kapselung der Funktion erhalten.

Das Gegenstück zur Verfeinerung von Funktionen, die Komposition, fasst Funktionen zu größeren Funktionen zusammen. Die Komposition wird ebenfalls methodisch unterstützt, die oben skizzierten Einschränkungen und Betrachtungen bleiben analog gültig.

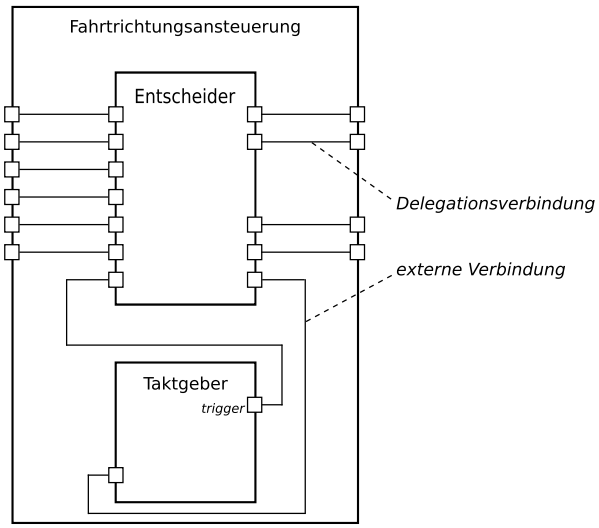
7.3 Verbindung von Funktionen

Die Verbindung von Funktionen definiert die Kommunikation der Funktionen. Durch das Verbinden der Ein- und Ausgangsports werden die Kommunikations-

beziehungen für den Signalfluss hergestellt. Wie in Abbildung 28 wird die Verbindung der Ports durch Linien zwischen den entsprechenden Ein- und Ausgangsports dargestellt.

Es werden externe Verbindungen und Delegationsverbindungen unterschieden. Externe Verbindungen verbinden zwei gleichrangig stehende Funktionen, im Bild die Funktionen *Entscheider* und *Taktgeber*. Delegationsverbindungen verbinden eine Funktion und die Ports der sie umgebenden Funktion, die Signale der umgebenden Funktion werden an die innere Funktion weitergeleitet, delegiert.

Abbildung 28 Verbindung von Funktionen.



Die Verbindung zweier Ports ist nur erlaubt, wenn die *Interfaces* der Ports »passen«. Das ist der Fall, wenn die zu verbindenden Ports konform sind. Es werden zwei Arten von Konformität unterschieden: Konformität der statischen Schnittstellen sowie Konformität der Protokolle.

Die Konformität der statischen Schnittstellen wird über deren Signale geprüft. Für die Konformität reicht es aus, wenn alle benötigten Signale auch geliefert werden. Das heißt, die Signale, die der Eingansport benötigt, müssen von den Ausgangsports geliefert werden, mit denen der Eingangsport verbunden ist.

Die Konformität der Protokolle sagt aus, dass die Protokollstatecharts der verbundenen Ports konform sein müssen. Das ist zunächst bei gleichen Protokollen der Fall. Die Bedingungen für Protokollkonformität, die über diese Protokollgleichheit hinausgehen, sind in MOSES noch nicht definiert worden.

Für Verbindungen gilt, dass Funktionsgrenzen nicht von Verbindungen durchstoßen werden dürfen. Muss über eine solche Grenze kommuniziert werden, ist an dieser Stelle ein Port zu modellieren, der als Schnittstelle fungiert. Damit bleibt die strikte Kapselung der Funktionen gewahrt.

7.4 Verhalten

Für die Integration von Funktionen in ein Funktionsnetzwerk reicht ein Modell der statischen Struktur allein nicht aus. Auch das dynamische Verhalten der Funktionen muss in angemessener Abstraktion repräsentiert werden, so dass die drei wesentlichen Fragen beantwortet werden können:

1 Wie verhält sich eine Funktion in ihrem Kontext?
2 Kann die Funktion aus ihrem Kontext herausgeschnitten und in ein anderes Netzwerk eingesetzt werden?
3 Leistet sie das gewünschte Verhalten auch in dem neuen Kontext?

Für das Einsetzen einer Funktion in ein Netzwerk wird ein *äußeres* Verhalten angegeben, das die Ein- und Ausgänge der Funktion durch zeitliche Informationen (Reihenfolgen, Zykluszeiten etc.) anreichert. Die Abhängigkeit der Ausgangs- von den Eingangssignalen wird durch das *innere* Verhalten der Funktion spezifiziert. Dazu können Zustände als Funktions- bzw. Betriebsmodi benutzt werden, um verschiedene Reaktionen auf Eingangssignale unterscheiden zu können. Verhaltensmodelle dienen also der schrittweisen Verfeinerung der Gesamtmodelle.

Im folgenden Abschnitt wird zunächst die Modellierung des inneren Verhaltens einer Funktion beschrieben. Anschließend wird die Erweiterung von Ein- und Ausgängen durch Verhaltensbeschreibungen diskutiert. Für die Modellierung werden Statecharts verwendet, Verhaltensstatecharts für das innere Verhalten und Protokollstatecharts für das äußere Verhalten.

Inneres Verhalten von Funktionen

Die Modellierung des inneren Verhaltens von Funktionen in Funktionsnetzwerken dient der Beschreibung des Ein-/Ausgabeverhaltens dieser Funktionen bezüglich ein- und ausgehender Signale. Dabei beschreiben die Verhaltensmodelle nur das prinzipielle Verhalten der Funktionen, ohne eine Realisierung vorwegzunehmen, wie sie z. B. durch die technische Architektur erfolgt. Die Beschreibung des inneren Verhaltens ist also kein Abstieg in die internen Details der Funktion, sondern liegt auf der gleichen Abstraktionsebene wie das statische Modell der Funktion.

Als Beschreibungsmittel werden Statecharts gemäß UML 2.0 verwendet. Bezüglich der Ereignisse und Bedingungen der Statecharts wird die Unterscheidung zwischen transienten und permanenten Signalen wirksam: Signale können zum einen dazu verwendet werden, bestimmte Ereignisse anzuzeigen, z. B. eine Eingabe vom Fahrer oder eine Warnmeldung, die von einer Überwachungsfunktion verschickt wurde. Solche Signale dienen im wesentlichen der Steuerung des Kontrollflusses. Das Verhalten der Funktion hängt davon ab, ob ein solches Steuerungssignal eintrifft oder nicht. In diesem Fall werden transiente Signale verwendet.

Andererseits können Signale dazu benutzt werden, Werte zu übertragen, wie z. B. aufbereitete Sensorwerte oder Vergleichswerte, die zur Plausibilisierung benutzt werden. Diese Signale realisieren vornehmlich den Datenfluss durch das System. Wenn man von der technischen Realisierung abstrahiert kann davon ausgegangen werden, dass es sich um permanente Signale handelt. Dies lässt sich in der weiteren Entwicklung mit verschiedenen Zeitmodellen unterlegen und entsprechend auf unterschiedliche Weise realisieren. In einem kontinuierlichen Zeitmodell bzw. einer Realisierung durch ein analoges Signal über einen Draht liegt das Signal tatsächlich kontinuierlich an, in einem diskreten Zeitmodell bzw. einem diskreten Signal über einen Bus wird das Signal in einer bestimmten Zykluszeit verschickt. Das logische Modell lässt beide Varianten offen.

Zusammenfassend ergeben sich für die Definition der Transitionen in der Verhaltensmodellierung durch Statecharts folgende Möglichkeiten:

Ereignis Eintreffen eines transienten Signals.

Bedingung Abfrage von (beliebig vielen) Zustandsvariablen und eingehenden permanenten Signalen, sowie dem Wert des eingetroffenen transienten Signals.

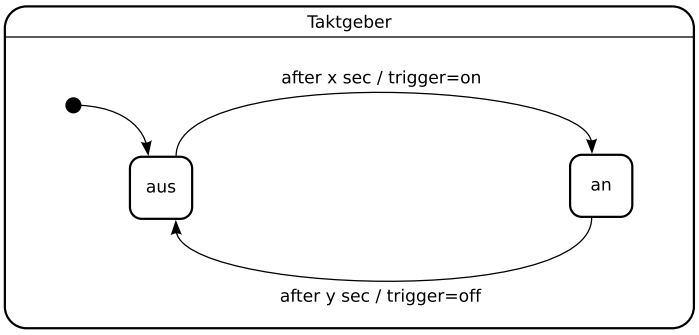
Aktion Zuweisungen an Zustandsvariablen und ausgehende permanente Signale sowie das Verschicken von transienten Signalen.

Abbildung 29 zeigt ein Beispiel für ein inneres Verhaltensmodell: den Taktgeber des Fahrtrichtungsanzeigers. Er besitzt zwei Zustände: *an* und *aus*, zwischen denen nach definierten Zeiten gewechselt wird. Bei jedem Wechsel wird das Signal *trigger* mit einem neuen Wert versehen und als Ereignis versendet.

Äußeres Verhalten von Funktionen

Komplementär zu der Beschreibung des inneren Verhaltens einer Funktion, d. h. der Beziehung von den Eingangssignalen zu den Ausgangssignalen innerhalb der

Abbildung 29 Inneres Verhaltensmodell einer Funktion



Funktion, wird mit dem äußeren Verhalten angegeben, wie sich die Funktion in ihrem Kontext verhält bzw. welche Bedingungen sie an ihre Umgebung stellt. Damit kann der Verhaltensanteil der Ein- und Ausgänge formuliert werden.

Die Modellierung des äußeren Verhaltens einer Funktion wird durch Protokollstatecharts an den Ports der Funktion dargestellt. Dabei kann ein Protokollstatechart für einen oder für mehrere Ports angegeben werden. Über die statische Information in den Schnittstellen hinaus wird hier angegeben, in welcher Reihenfolge und in welchen Zeitabschnitten Signale empfangen (Eingangsprotokoll) bzw. verschickt (Ausgangsprotokoll) werden können.

Protokollstatecharts sind wie Statecharts aufgebaut, mit folgenden Unterschieden: an den Transitionen eines Protokollstatecharts dürfen nur Ereignisse (Eintreffen eines Steuerungssignals) stehen, sowie Vor- und Nachbedingungen (Bedingungen an die Werte der Datensignale, Zustandsvariablen der Schnittstelle und Parameter des aktuellen Steuerungssignals). Die Nachbedingungen sind dabei Zusicherungen, die in dem Endzustand der Transition gelten. Protokollstatecharts enthalten keine Aktionen.

Verhalten in Funktionshierarchien

Das Verhalten von hierarchischen Funktionen wird über das Verhalten der inneren Funktionen definiert: Das Gesamtverhalten der äußeren Funktion setzt sich aus dem Einzelverhalten der inneren Funktionen zusammen, ohne dass zusätzliches Verhalten modelliert wird.

Damit ist es möglich, das Gesamtverhalten eines Systems genau zu spezifizieren. Probleme entstehen, wenn Funktionen komponiert bzw. dekomponiert werden sollen und dabei das Verhalten konform mitgezogen werden soll. Dieser Fall ist

noch offen, in der Literatur werden z. B. zu diesem Zweck Konformitätsgrade unterschieden. Sie werden in [Fra03a] ausführlich erläutert, an dieser Stelle wird darauf verzichtet. Für die Dekomposition werden im Dokument ebenfalls verhaltenskonforme Verfeinerungsmöglichkeiten beschrieben.

Fehlermodellierung

Die Fehlermodellierung ermöglicht es, die Fehlerfortpflanzung durch ein Funktionsnetzwerk zu modellieren und zu verfolgen. Dafür wurde ein Konzept erarbeitet, das diesem Zweck genügt.

Das Fehlermodell basiert auf vier Äquivalenzklassen für die Fehler, die im System auftreten können:

Signal okay (ok) beschreibt die Korrektheit eines Signals: das Signal ist vorhanden und sein Wert befindet sich im entsprechend definierten Wertebereich.

Wertfehler (W) definieren Fehler, die sich auf den Inhalt bzw. die Information eines Signals beziehen. Das können z. B. Über- oder Unterschreitungen eines Wertebereichs sein.

Protokollfehler (P) definieren Fehler im dynamischen Verhalten eines Signals. Das kann z. B. das Ausbleiben eines erwarteten Signals sein.

Signalfehler (F) sind generelle Fehler, die nicht in Wert- oder Protokollfehler unterschieden werden.

Mit Hilfe dieser Äquivalenzklassen können Fehlermatrizen aufgestellt werden, die Aussagen darüber enthalten, welche Fehlerklasse am Ausgang einer Funktion zu erwarten ist, wenn Fehler am Eingang der Funktion auftreten. Damit ist es dann möglich, die Fortpflanzung eines Fehlers über das gesamte Funktionsnetzwerk zu beobachten. Die Matrizen werden als »Erweiterte UND/ODER-Matrizen« bezeichnet, eine Abgrenzung gegenüber einfachen UND/ODER-Matrizen, die nicht mit Äquivalenzklassen arbeiten und beispielsweise im Werkzeug Cape/C als »Defektmatrizen« verwendet werden.

Tabelle 5 Erweiterte UND/ODER-Matrix (Beispiel).

		«Entscheider»		
Input	trigger	P	*	*
	leuchte_re_ok	*	W	*
	leuchte_li_ok	*	*	W
Output	blinker_re	P	W	*
	blinker_li	P	*	W

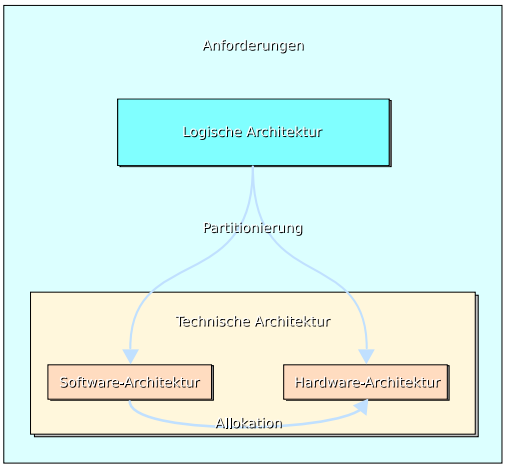
Die Matrizen sind spaltenweise zu lesen: In Tabelle 5 erzeugt ein Protokollfehler des Triggersignals einen Protokollfehler an beiden Blinkern, unabhängig von den anderen Signalen. Ein Wertfehler an *leuchte_re_ok* erzeugt einen Wertfehler an dem rechten Blinker. Ein Wertfehler an *leuchte_li_ok* pflanzt sich als Wertfehler des linken Blinkers fort.

8 Technische Architektur

Die technische Architektur (gelb in Abbildung 30) eines Systems besteht aus den Ressourcen, die verwendet werden, um die logische Funktionalität des Systems zu realisieren. Ressourcen sind zum Beispiel Hardwarebestandteile, wie die Recheneinheiten und Speicher der Steuergeräte sowie die Busse und sonstigen Verbindungen zwischen den Steuergeräten. Ressourcen sind auch Softwarebestandteile wie Routinen, Tasks und Betriebssystemdienste. Die technische Architektur wird als Gesamtressource zunächst unabhängig von der logischen Funktionalität modelliert. Dazu werden Modellierungskonzepte verwendet, die eine für die Entwicklung und Integration angemessene abstrakte Repräsentation der Architektur ermöglichen. Die Konzepte werden in [Fra03b] und [Fra04a] beschrieben.

Die Abbildung der logischen Architektur des Systems auf die technische Architektur geschieht in einem separaten zweiten Schritt, der Partitionierung, die im nachfolgenden Kapitel 9 vorgestellt wird.

Abbildung 30 Die technische Architektur.



Die technische Architektur besteht aus der Software- und der Hardwarearchitektur, die die entsprechenden Bestandteile getrennt modellieren. Die Verbindung zwischen den beiden Teilmodellen wird ebenfalls über die Partitionierung geschaffen. Diese Partitionierung, die die Soft- auf die Hardware abbildet, wird auch als Allokation bezeichnet.

8.1 Die Softwarearchitektur

Die Softwarearchitektur in MOSES beschränkt sich auf drei Kernelemente: Softwareelement, Routine und Task. Routinen sind die kleinsten ausführbaren Einheiten eines Systems. Sie entsprechen Prozeduren oder Methoden in Programmiersprachen.

Die Routinen werden zu Softwareelementen zusammengefasst, die installierbare Softwareeinheiten repräsentieren. Ein Softwareelement kann nur als Ganzes genutzt werden. Softwareelemente werden in hardwareabhängige Elemente (z. B. Gerätetreiber) und hardwareunabhängige Elemente (z. B. Anwendungssoftware) unterschieden. Diese Unterscheidung hat Auswirkungen auf den Freiheitsgrad bei der Allokation.

Orthogonal zur Gruppierung von Routinen in Softwareelemente werden Routinen in Tasks gruppiert, um die prozessorgerechte Abarbeitung zu modellieren. Dabei kann ein Task mehrere Routinen aufrufen oder eine Routine von mehreren Tasks aufgerufen werden.

Die Softwarearchitektur besitzt im Vergleich zu den anderen MOSES-Modellen die wenigsten Modellierungskonzepte und die wenigsten Metaklassen. Das rührt daher, dass zeitgleich zur Entwicklung der technischen Architektur die AUTOSAR-Initiative gestartet wurde, die ebenfalls eine Beschreibungssprache für technische Architekturen im Automobilbau definiert, jedoch wesentlich implementierungsnaher. Um Doppelentwicklungen zu vermeiden, wurde entschieden, die Softwareanteile der technischen Architektur nur soweit zu modellieren, dass die weitere Modellierung mit Hilfe des *Software Component Templates* von AUTOSAR vorgenommen werden kann. Die genaue Anbindung der Modelle wird zur Zeit im MOSAR-Projekt definiert.

8.2 Die Hardwarearchitektur

Die Hardwarearchitektur baut, wie die logische Architektur, auf dem Komponentengedanken auf. Hardware wird über Komponenten beschrieben, die hierarchisierbar sind und über Verbindungen miteinander kommunizieren können. Dabei werden Schnittstellen für die Komponenten definiert, über die eine vollständige Kapselung der Komponenten erreicht wird.

Auch die Hardwarearchitektur wurde parallel zu AUTOSAR entwickelt, allerdings war sie schon wesentlich weiter entwickelt, bevor AUTOSAR initiiert wurde. Daher

ist die Hardwarearchitektur feiner definiert, wurde aber trotzdem so entworfen, dass eine einfache Abbildung auf AUTOSAR möglich ist. Auch an dieser Stelle sei auf das MOSAR-Projekt verwiesen, das diese Abbildung definiert.

Die Hardwarearchitektur besteht aus drei Hauptbestandteilen:

1 Hardwareelementen,
2 Ports und
3 Verbindungen.

8.2.1 Hardwareelemente

Die Hardwareelemente sind die zentralen Elemente der Hardwarearchitektur. Sie entsprechen im Normalfall realen Hardwarebauteilen im Automobil. Es können zwar auch virtuelle Hardwarekomponenten gebildet werden, diese Möglichkeit wird an dieser Stelle jedoch nicht näher beschrieben.

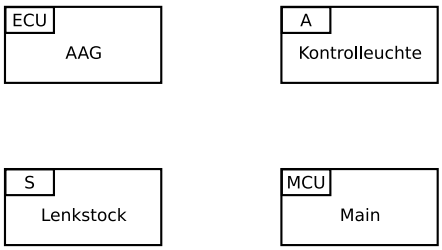
Hardwareelemente werden in Hardwarekomponenten (z. B. Steuergeräte) und innere Hardwareelemente (z. B. Prozessoren) unterschieden. Diese Unterscheidung dient dazu, hierarchisierbare Elemente (Hardwarekomponenten) von nicht weiter verfeinerbaren Elementen (inneren Elementen) zu trennen. Diese Unterscheidung hat praktische Gründe: Die Modellierung der Hardwarekomponenten ist vornehmliche Aufgabe eines Herstellers. Die inneren Elemente interessieren nur soweit, als sie für die Ressourcenplanung wichtig sind. Ihre Schnittstellen oder ihre innere Struktur sind jedoch für den Hersteller in der Modellierung nicht weiter wichtig.

Hardwarekomponenten

Hardwarekomponenten sind hierarchisierbare Komponenten, die Schnittstellen (Ports) besitzen und miteinander verbunden werden können. Sie bieten die Möglichkeit, reale Hardwarebauteile wie Steuergeräte, Aktoren, Sensoren aber auch Mikrocontroller zu modellieren. Abbildung 31 zeigt einige Beispiele für Hardwarekomponenten.

Als Hardwarekomponenten werden die Bauteile modelliert, die eine Schnittstelle und innere Struktur besitzen. Die innere Struktur wird über Komposition anderer Hardwareelemente gebildet, die Schnittstelle sind die Ports der Hardwarekomponente. Eine Hardwarekomponente ist ein abstraktes Bauteil, für die Modellie-

Abbildung 31 Hardwarekomponenten (Beispiele).



rung wird eine Spezialisierung einer Hardwarekomponente herangezogen. MOSES kennt Sensoren und Aktoren, die direkte Pendants in der Hardwarewelt besitzen. Auch Mikrocontroller, Steuergeräte und externer Speicher besitzen MOSES-Entsprechungen. Für alle anderen Hardwarebauteile, die als Komponenten modelliert werden sollen, steht das allgemeine Hardwaregerät zur Verfügung.

Die Komposition der Hardwarekomponenten ist einigen wenigen Regeln unterworfen: Steuergeräte können nur von Steuergeräten aggregiert werden. Eine solche Komposition ist virtuell, da im realen Fahrzeug Steuergeräte verbaubare Einheiten darstellen. Aktoren bzw. Sensoren können Aktoren bzw. Sensoren enthalten sowie MCUs und externen Speicher. Damit ist die Möglichkeit gegeben, z. B. Sensorcluster zu modellieren oder Sensoren mit eigenem Verhalten z. B. zur Signalaufbereitung zu versehen, ohne die Modellierungsmöglichkeiten zu weit von der Wirklichkeit anzusiedeln.

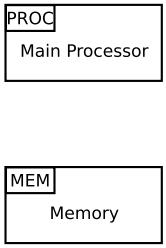
Innere Hardwareelemente

Die inneren Hardwareelemente sind Prozessoren, Speicher und sonstige Elemente. Sie zeichnen sich dadurch aus, dass sie keine Schnittstellen (Ports) und keine innere Struktur besitzen. Sie sind lediglich mit Attributen versehen, die zur Ressourcenabschätzung herangezogen werden können. Zwei innere Hardwareelemente sind in Abbildung 32 zu sehen: ein Prozessor sowie ein Speicher.

8.2.2 Hardwareports

Hardwareports sind, analog zur logischen Ports, die Schnittstellen der Hardwarekomponenten. Die Analogie geht allerdings nicht über diese allgemeine Feststellung hinaus, da Hardwareports viele Unterschiede zu logischen Ports aufweisen.

Abbildung 32 Innere Hardwareelemente (Beispiele).



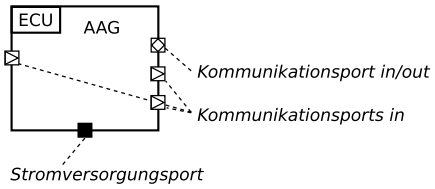
Hardwareports sind für Hardwarekomponenten die einzige Möglichkeit, mit der Umwelt bzw. dem Inneren zu kommunizieren. Das heißt, Hardwarenetze werden durch die Verbindung von Hardwareports (im Folgenden nur noch »Ports«) realisiert. Dazu besitzen Ports Verbindungsenden, an die Verbindungen geknüpft werden können. Diese Verbindungsenden werden typisiert, damit sind auch die Ports typisiert.

Bei Ports werden drei Grundarten unterschieden (Abbildung 33):

1 Kommunikationsports,
2 Stromversorgungsports und
3 Leistungsports.

Kommunikationsports sind Ports, über die Daten bzw. Nachrichten ausgetauscht werden, Stromversorgungsports übernehmen die Stromversorgung einer Komponente. Leistungsports sind eine Mischung aus den beiden vorigen Typen, sie übertragen Signale und übernehmen dabei gleichzeitig die Stromversorgung, z. B. werden frequenzmodulierte Signale über Leistungsports übertragen.

Abbildung 33 Hardwareports (Beispiele).

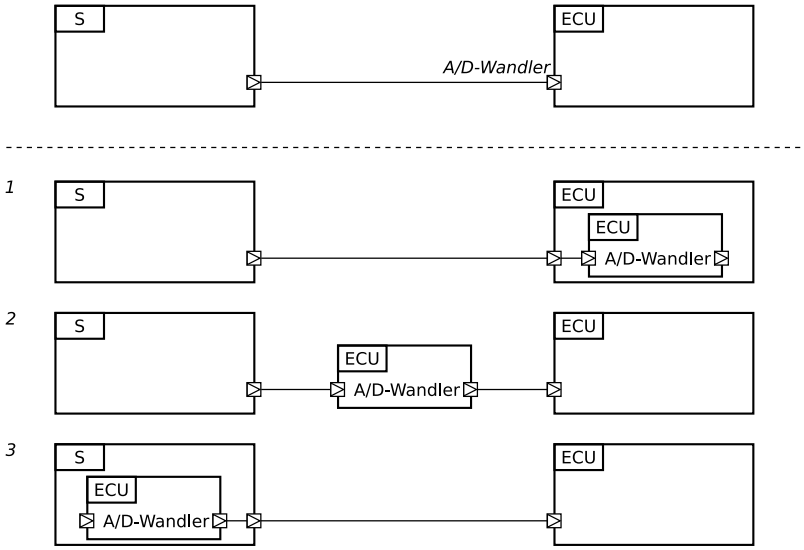


Ports in MOSES können »intelligent« sein. Das heißt, sie können Funktionalität enthalten, die für den Port wichtig ist. So müssen Ports, die über Busse verbunden werden das jeweilige Busprotokoll verstehen, die Signale entgegennehmen

und in eine interne Repräsentation umwandeln. Ein anderes Beispiel ist der A/D-Wandler von Abbildung 34. In MOSES (oben im Bild) übernimmt der Port selbst die Wandlung. In AUTOSAR z. B. muss dafür eine eigene Komponente modelliert werden. Dafür stehen drei Möglichkeiten zur Verfügung (unten im Bild):

1 Modellierung des A/D-Wandlers im Steuergerät
2 Modellierung des A/D-Wandlers als zusätzliches Steuergerät
3 Modellierung des A/D-Wandlers im Sensor

Abbildung 34 Auflösung intelligenter Ports.

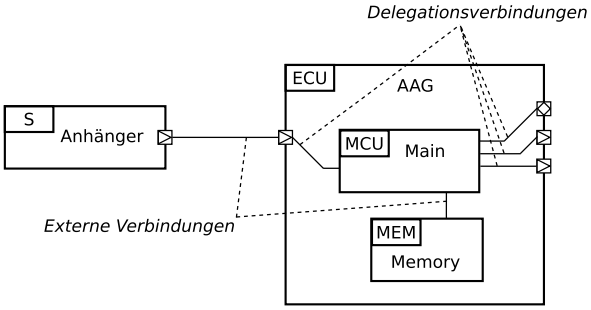


8.2.3 Verbindungen

In MOSES können alle Hardwareelemente miteinander verbunden werden. Die Verbindung von Hardwarekomponenten erfolgt über deren Ports, die Verbindung von inneren Komponenten erfolgt direkt. Zu diesem Zweck besitzen sowohl Ports als auch innere Komponenten Verbindungsenden, an die die Verbindung geknüpft wird.

Verbindungsenden und Verbindungen können typisiert werden. So kann bereits auf Modellierungsebene sichergestellt werden, dass nur gleich typisierte Verbindungsenden und Verbindungen miteinander verknüpft werden.

Abbildung 35 Verbindungen (Beispiele).



Je nachdem, ob hierarchisch gleiche oder unterschiedliche Elemente miteinander verbunden werden, werden externe Verbindungen oder Delegationsverbindungen unterschieden (Abbildung 35). Externe Verbindungen werden zwischen Elementen gleicher Hierarchieebene gezogen. Der Name rührt daher, dass diese Verbindungen außerhalb der zu verbindenden Elemente laufen. Delegationsverbindungen überspringen eine Hierarchiestufe und verbinden Hardwarekomponenten mit den von ihnen aggregierten Hardwareelementen.

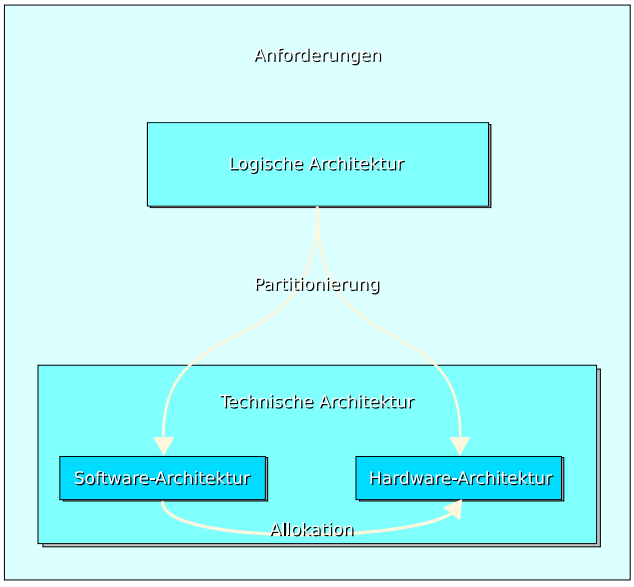
Für alle Verbindungen gilt, dass keine Elementgrenzen von Verbindungen übersprungen werden können. Muss eine solche Grenze durchstoßen werden, ist an dieser Stelle ein Port zu modellieren, der nach beiden Richtungen als Schnittstelle fungiert. Damit ist die strikte Kapselung der Elemente gewahrt.

An dieser Stelle sei noch auf eine Besonderheit der Verbindungsmodellierung hingewiesen: Während die Modellierung über Ports einen hohen Abstraktionsgrad voraussetzt, werden reale Hardwarebauteile über Stecker, konkret über deren Pins verbunden. Daher können Hardwarekomponenten Pins zugeordnet werden, die mit den Verbindungsenden der Ports in Beziehung gesetzt werden. So braucht die Abstraktionsstufe nicht verringert werden und die Probleme der Entwickler mit Pinbelegungen werden berücksichtigt.

9 Partitionierung

Die Partitionierung in MOSES (gelb in Abbildung 36) dient zwei unterschiedlichen Zwecken: Zum einen werden Funktionen auf die technische Architektur verteilt. Dieser Aspekt wird im Folgenden »Partitionierung«, also Ein- oder Aufteilung genannt. Zum anderen wird festgelegt, welche Software auf welcher Hardware laufen soll. Dieser Aspekt wird mit »Allokation«, also Verteilung oder Platzierung betitelt. Da beide Aspekte Einfluss aufeinander haben, wurde ihre Modellierung in MOSES zusammengefasst. Die Beschreibung der Partitionierung und Allokation ist in [Fra03b] und [Fra04a] zu finden.

Abbildung 36 Partitionierung.



Sowohl Partitionierung als auch Allokation sind Verbindungen zwischen unabhängigen Modellen. Das heißt, die verbundenen Modelle können aufgrund der Partitionierung tatsächlich unabhängig voneinander modelliert werden. Damit wird durch die Partitionierung die Unabhängigkeit der logischen Architektur von ihrer technischen Umsetzung gewährleistet. Die Allokation sichert die freie Verschiebbarkeit der Software auf unterschiedliche Hardware zu.

9.1 Partitionierung auf die technische Architektur

Die Partitionierung ist die Abbildung der logischen auf die technische Architektur. Dabei muss entschieden werden, welche Funktionen in Software realisiert werden und welche in Hardware. Damit werden Bedingungen aufgestellt, die von den Funktionen an die sie realisierenden technischen Elemente gestellt werden. Für Hardwareelemente bedeutet das zum Beispiel, dass genügend Ressourcen zur Verfügung gestellt werden müssen, um ein gewünschtes Zeitverhalten zu realisieren. Im Zusammenspiel mit der Allokation muss ein konsistentes Modell entstehen.

Partitionierung auf Hardware

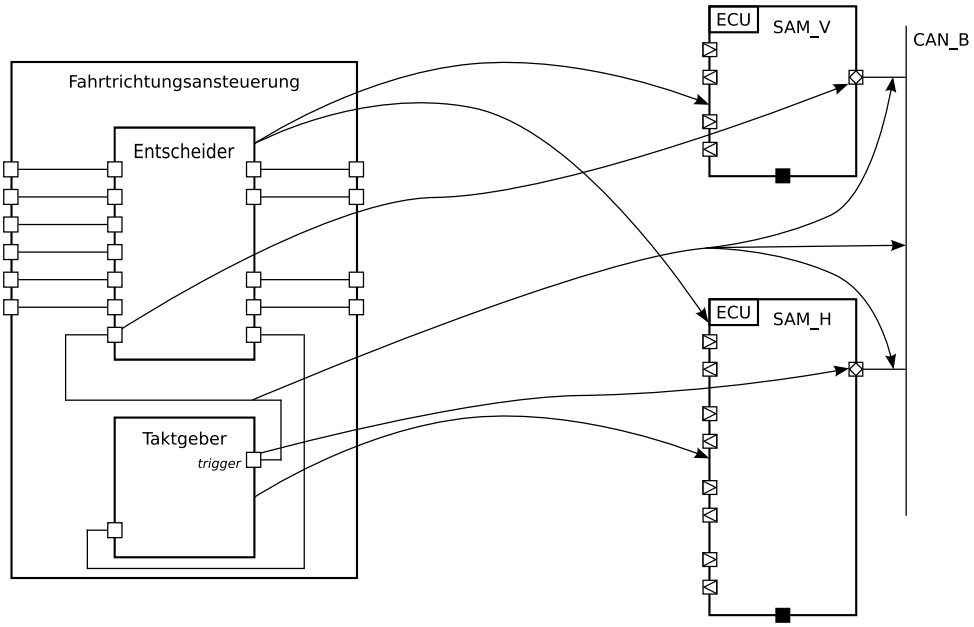
Die Partitionierung definiert eine Abbildung der logischen Elemente Funktion, Signal und Verbindung auf ihre technischen Entsprechungen. Für die Partitionierung auf Hardware sind das Hardwareelemente, Verbindungsenden und Verbindungspfade, bestehend aus Verbindungen und den zwischenliegenden Elementen.

Die Partitionierung von Funktionen auf Hardwareelemente ist eine einfache Abbildung (*Taktgeber* auf *SAM_H* in Abbildung 37), mehrere Funktionen können auf ein Hardwareelement partitioniert werden und umgekehrt können mehrere Hardwareelemente eine Funktion realisieren (*Entscheider* in Abbildung 37).

Die Partitionierung von Signalen ist etwas komplizierter, da Signale im Funktionsnetzwerk nur im Kontext der sie versendenden bzw. empfangenden Ports eindeutig sind. Daher werden Paare von Signalen und Ports auf Verbindungsenden partitioniert. Auch hier ist die Verbindung eine n:m-Relation. In Abbildung 37 wird das Signal *trigger* mit seinem Port auf die Hardware partitioniert.

Eine Verbindungspartitionierung ist nur dann notwendig, wenn der Pfad, über den ein Signal geschickt werden soll, nicht eindeutig aus der Signalpartitionierung berechnet werden kann. Dann muss der Pfad angegeben werden, der das Signal tatsächlich transportiert. Ein solcher Pfad ist im Hardwaremodell über die Kette der Hardwareelemente, ihrer Verbindungsenden und der Verbindungen anzugeben. Auf diesen Pfad wird das Signal und die betrachtete Verbindung partitioniert. In Abbildung 37 wurde die Verbindung zwischen *Entscheider* und *Taktgeber* der Deutlichkeit halber ebenfalls partitioniert, auch wenn diese Partitionierung eindeutig hätte generiert werden können.

Abbildung 37 Partitionierungsmöglichkeiten.



Partitionierung auf Software

Die Partitionierung der logischen Architektur auf die Software ist wesentlich einfacher. Es ist nur vorgesehen, Funktionen auf Routinen zu partitionieren. Auch diese Partitionierung ist eine n:m-Relation, das bedeutet, die Funktion kann mit mehreren Routinen realisiert werden, eine Routine kann aber auch mehrere Funktionen oder Teile von ihnen realisieren.

Diese Partitionierung ist deshalb so knapp, weil die Definition der AUTOSAR-Softwareelemente erst nach dem Ende des MOSES-Projekts konsolidiert wurde. Auch hier wird das MOSAR-Projekt die Partitionierung so weit verfeinern, dass z. B. Verbindungen zwischen Funktionen in Software nachzubilden sind etc.

Die Partitionierung ist dennoch hilfreich, da sie, wie im folgenden Abschnitt dargelegt wird, eine Überprüfung der Allokation zulässt.

9.2 Allokation der Software auf die Hardware

Die Allokation gibt an, welche Software auf welcher Hardware ausgeführt wird. Dabei kann nur die hardwareunabhängige Software verteilt werden, die hardwareabhängige Software ist fest mit ihrer Hardware verbunden.

Die Allokation betrifft die Softwareelemente und die Tasks. Softwareelemente, die Container der Routinen, werden auf diejenigen Hardwarekomponenten abgebildet, auf denen sie ausgeführt werden sollen. Dabei können nur ganze Softwareelemente verteilt werden, nicht einzelne Routinen. Das entspricht dem Gedanken der Softwareelemente als installierbare Softwareeinheit. Die Abbildung ist also eine 1:m-Relation, Softwareelemente werden nicht aufgeteilt, eine Hardwarekomponente kann jedoch mehrere Softwareelemente ausführen.

Für die Ausführung der Softwareelemente ist das Taskmodell zuständig, die Tasks werden separat auf Prozessoren verteilt. Auch ein Task kann nicht geteilt werden, Prozessoren können aber mehrere Tasks ausführen. Für die Konsistenz der Allokation muss sichergestellt sein, dass die Tasks nur auf Prozessoren verteilt werden, die in Hardwarekomponenten liegen, auf denen die Softwareeinheit läuft, die den Task enthält.

9.3 Partitionierung und Allokation

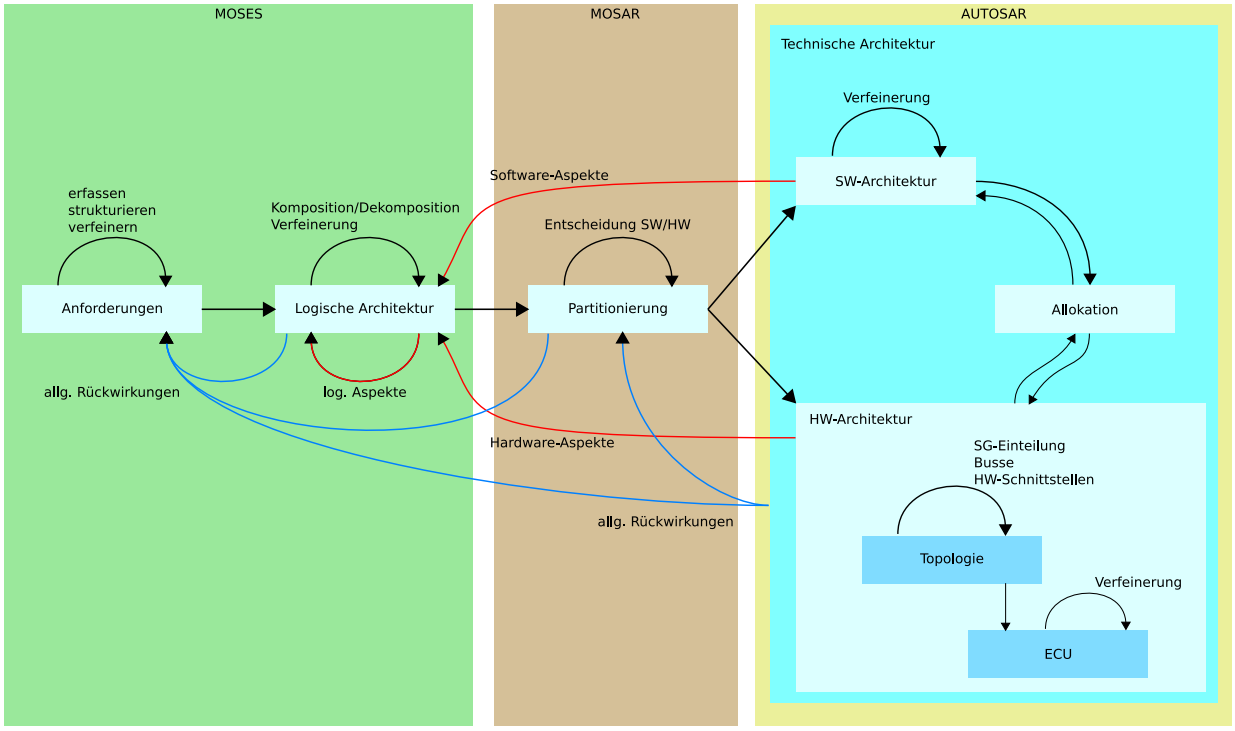
Die beiden unterschiedlichen Aspekte der Partitionierung müssen im entwickelten Modell zusammenpassen, also untereinander konsistent sein. Das muss überprüft werden, wenn Funktionen auf Software partitioniert werden und diese Software dann auf Hardware verteilt wird. Dann muss die Funktion natürlich auch auf die entsprechende Hardware verteilt werden. Diese Bedingung ist so wichtig, dass sie im Metamodell berücksichtigt wurde. MOSES lässt es nicht zu, Funktionen auf Software zu partitionieren, die auf anderer Hardware läuft als die Funktion.

10 AUTOSAR

Wie bereits mehrfach erwähnt, ist AUTOSAR parallel zu MOSES entstanden und definiert unter anderem eine technische Architektur für eingebettete Systeme im Automobil. Dabei werden Softwarekomponenten und Steuergeräte unterschieden, die in jeweils einem eigenen Template beschrieben werden. Zusätzlich werden die Verbindung beider Architekturen und systemweite Bedingungen, Netze etc. definiert, die im System Constraint Template bzw. in der System Description abgelegt werden.

Im Gegensatz zu MOSES ist AUTOSAR ein implementierungsnahes Modell, das als Generierungsmodell genutzt werden soll. Daher entspricht vor allem die implementierungsnahe technische Architektur von MOSES den Modellen von AUTOSAR. Höhere Abstraktionsebenen sind in AUTOSAR nicht vorgesehen.

Abbildung 38 MOSES – MOSAR – AUTOSAR.



MOSES wurde unabhängig von AUTOSAR entwickelt, ignoriert die dortigen Be-

mühungen aber nicht, sondern bietet Schnittstellen an, die eine Verbindung von MOSES-Modellen mit AUTOSAR-Modellen zulassen. Aufgrund der noch nicht vollständig abgeschlossenen Definition von AUTOSAR werden in diesem Kapitel die Möglichkeiten aufgezeigt, wie MOSES und AUTOSAR miteinander verknüpft werden können. Dabei kann noch nicht konkreter argumentiert werden, da, wie gesagt, die Metamodelle von AUTOSAR noch nicht stabil sind. Die Verbindung von MOSES und AUTOSAR wird derzeit im MOSAR-Projekt untersucht.

Abbildung 38 zeigt die Einbindung der AUTOSAR-Modelle in den MOSES-Kontext. Anforderungen werden wie bisher mit MOSES-Mitteln erfasst und daraus eine logische Architektur erstellt. Beide Architekturmodelle sind in AUTOSAR nicht definiert worden.

Danach wird die logische Architektur partitioniert. Diese Einteilung in Soft- und Hardware zielt direkt auf AUTOSAR-Modelle ab. Funktionen werden also auf die Elemente der *Software Component Description* und der *ECU Resource Description* partitioniert. Die entsprechenden MOSES-Modelle sind zwar abstrakter als ihre AUTOSAR-Entsprechungen, eine MOSES-Zwischenschicht würde aber die Modellierung unnötig aufblähen. Die Modellierung der technischen Architektur muss daher mit AUTOSAR-Mitteln vorgenommen werden.

Die Allokation und Busverteilung – der Systementwurf – werden vollständig in AUTOSAR durchgeführt. Damit sind beide Modelle verbunden, die mächtigeren Modellierungsmöglichkeiten von MOSES, z. B. bei Varianz, werden zugunsten der praktischeren Ausrichtung von AUTOSAR aufgegeben. Inwieweit sie nachgezogen werden müssen, kann erst im MOSAR-Projekt geklärt werden.

Teil III

Erweiterungen: Redundanz, Varianz, Domäne

Die bisher eingeführten Architekturmodelle dienten dazu, genau ein System zu spezifizieren. In der Praxis werden meist mehrere Fahrzeuge gleichzeitig modelliert:

Zum einen werden im praktischen Einsatz keine einzelnen Fahrzeuge modelliert, sondern Fahrzeugfamilien oder Baureihen, dafür werden Produktlinien und Domänen eingesetzt. Des Weiteren sollen Entwurfsentscheidungen wiederverwendet werden und diese Wiederverwendung nicht über *Copy & Paste*, sondern wohldefiniert über ein eigenes Repository erfolgen.

Ein zweiter zu modellierender Aspekt ist Redundanz, die zum einen dem System innewohnt, zum anderen durch Überlegungen z. B. zur Sicherheit eingeführt wird. Diese Redundanz muss explizit modelliert werden, da ansonsten im Modell nicht unterschieden werden kann, ob z. B. eine Funktion zweimal vorhanden ist oder ein- und dieselbe Funktion redundant ausführt.

Daher wurden die MOSES-Modelle um genau diese beiden Aspekte erweitert. Die Aspekte betreffen nicht jedes Teilmodell, die folgenden Kapitel erläutern zunächst die Modellierung von Redundanz und danach die Modellierung von Varianz.

In diesem Teil werden nur die modelltechnischen Konzepte erläutert, die prozessualen Auswirkungen und der Umgang mit Produktlinien und Domänenmodellen sind Inhalt von Teil IV.

11 Redundanz

Wie bereits erwähnt, resultiert Redundanz meist aus Sicherheitsüberlegungen. Funktionen müssen redundant ausgeführt werden, z. B. um sich gegenseitig zu überprüfen oder um Werte redundant zur Verfügung zu stellen. Auch Soft- und Hardware können redundant ausgelegt werden, meist resultiert diese Redundanz

aber aus der logischen Redundanz. Zu guter Letzt kann auch die Partitionierung redundant modelliert werden, wenn z. B. eine Funktion redundant ausgelegt werden soll, dies aber erst bei der Realisierung geschieht. Von der Redundanz sind modelltechnisch also nur die Anforderungen ausgenommen, in ihnen wird Redundanz gefordert und begründet, aber nicht modelliert.

Funktionale Redundanz schlägt sich in der logischen Architektur und in der Partitionierung nieder. In der logischen Architektur wird vermerkt, wenn die Logik nicht nur redundant realisiert werden soll, sondern die Auswertung der Redundanz signifikante logische Funktionalität benötigt. Das ist zum Beispiel der Fall, wenn Ersatzwerte benutzt werden und die Ersatzwerte aus redundaten Datenquellen stammen. Wenn hier für die Ersatzwertberechnung deutliche Funktionalität nötig ist, würde diese Redundanz in der logischen Architektur modelliert werden.

Die Redundanzmöglichkeiten der logischen Architektur wurden im MOSES-Rahmen untersucht, fanden aber keinen Eingang in das Metamodell. Die Ideen und Konzepte sind aber hinterlegt und können bei zukünftigen Erweiterungen berücksichtigt werden.

Redundanz, die explizit modelliert werden kann, ist für Partitionierungen festgelegt worden. Mit dieser Redundanz kann ausgedrückt werden, dass eine Funktion, ein Signal oder eine Verbindung redundant realisiert werden können. Damit wird die mehrfache technische Auslegung begründbar, ohne die logische Architektur ändern zu müssen. Diese Art der Redundanz reichte im MOSES-Rahmen aus, um Redundanz in einem praktikablen Ausmaß einsetzen zu können.

Redundanz in der technischen Architektur ist fast immer von logischer Seite über die Partitionierung getrieben. Daher wurde auch für technische Architekturen kein eigenes Redundanzmodell entwickelt, um die Modelle übersichtlich zu halten. Die von redundanter Partitionierung eingeführte Varianz auf technischer Ebene kann anhand genau dieser Partitionierung als redundant erkannt werden, so dass kein Informationsverlust auftritt.

Zusammenfassend beschränkt sich die Erweiterung der MOSES-Modelle um Redundanz aus pragmatischen Gründen auf Redundanz in Partitionierungen. Redundanz auf logischer Ebene wurde untersucht und kann bei Bedarf in das Metamodell nachgezogen werden, wobei vorher eine Abschätzung des Nutzens vorgenommen werden sollte.

12 Varianz

Neben der Redundanz wurden die MOSES-Modelle auch um das Konzept der Varianz erweitert. Varianz tritt auf, wenn mehrere Fahrzeuge – Baureihen oder Produktlinien – in einem Modell beschrieben werden sollen. Beispiele für mögliche Produktlinien sind in Abbildung 39 zu sehen. Die gezeigten Produktfamilien sind fiktive Beispiele, keine realen Produktfamilien bei BMW.

Abbildung 39 Produktlinien (fiktiv, Bilder: BMW [bmw]).



Die Anwendung der Varianzkonzepte, die im anschließenden Teil IV erläutert werden, beschäftigen sich mit den Prozessen, die bei der Anwendung von Varianz beachtet werden müssen. Dort werden auch die verschiedenen Modelle – Produktmodelle und Produktlinienmodelle – näher vorgestellt.

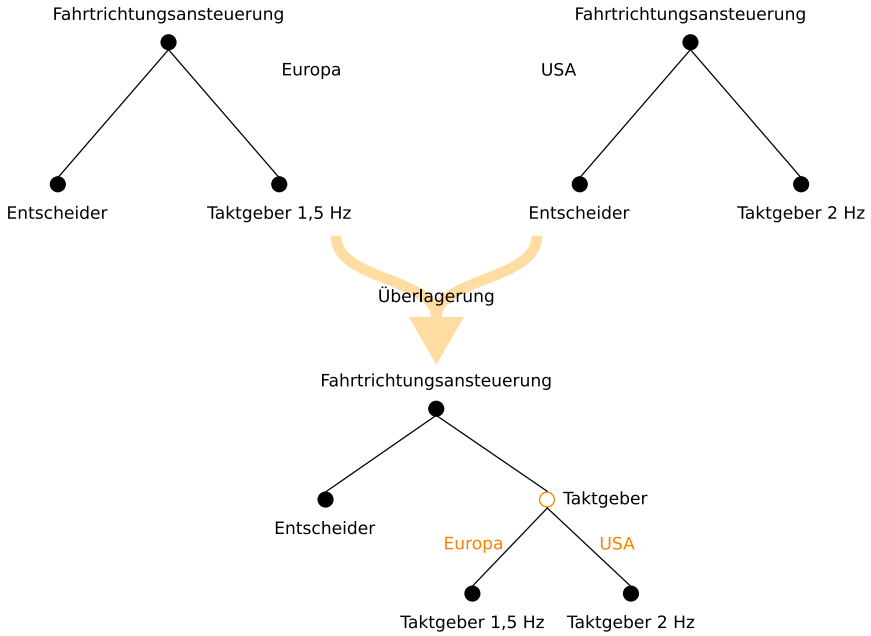
12.1 Überlagerung und Ausleitung

Zum Verständnis erfolgt an dieser Stelle ein kurzer Vorgriff auf die Anwendung von Varianz, damit die nachfolgenden Modellierungskonzepte besser verstanden werden.

Varianz entsteht, wie bereits erwähnt, wenn mehr als ein Produkt modelliert werden soll. Das kann man sich so vorstellen, dass die Produktmodelle der einzel-

nen Produkte übereinandergelegt werden. Durch dieses *Überlagern* kann man nun die Modelle vergleichen und Gemeinsamkeiten sowie Unterschiede herausarbeiten. Die Gemeinsamkeiten können unverändert als Produktlinienmodell übernommen werden, da diese Modellteile für alle Produkte gleich sind. Abbildung 40 zeigt zwei unterschiedliche Produktmodelle für Fahrtrichtungsanzeiger, die zu einem Produktlinienmodell überlagert werden.

Abbildung 40 Überlagerung.



Die Unterschiede müssen jetzt in das Produktlinienmodell eingearbeitet werden. Dabei sind zwei grundsätzliche Strategien unterscheidbar:

1 Vereinheitlichung der Unterschiede
2 Kennzeichnung der Unterschiede

Die Vereinheitlichung wird gewählt, wenn die Unterschiede so gering sind, dass sie ohne größeren Aufwand beseitigt werden können. Das hat Auswirkungen auf die Produkte, die vom Entwickler eingeschätzt werden müssen.

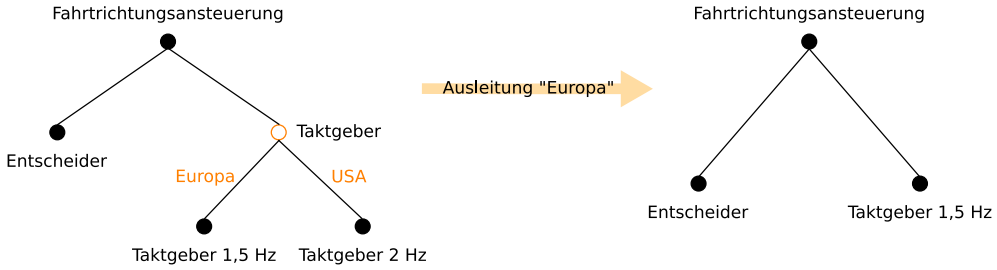
Die zweite Strategie, die Kennzeichnung der Unterschiede, führt zu varianten Modellen. Die Stellen, an denen Unterschiede auftreten, werden explizit als *Variantzpunkte* gekennzeichnet. Unterhalb der Varianzpunkte werden die unterschiedli-

chen Ausprägungen der Modelle, die *Varianten* aufgeführt. Dabei wird festgehalten, in welchem Produkt welche Variante gelten soll, die Varianten werden *annotiert*.

Im Beispiel von Abbildung 40 wurde ein Varianzpunkt *Taktgeber* eingeführt, unter den die zwei Varianten mit 1,5 bzw. 2 Hz liegen. Als Annotation dient hier die Länderausstattung.

Mit *Ausleitung* wird die Tätigkeit bezeichnet, die aus einem Produktlinienmodell ein einziges Produktmodell erzeugt. Dabei wird angegeben, welches Produktmodell gewünscht wird. Anhand der Kennzeichnung, die beim Überlagern annotiert wurde, kann an den Varianzpunkten entschieden werden, welche Variante in das zu erzeugende Produktmodell gehört und welche nicht. In Abbildung 41 werden die europäischen Fahrtrichtungsanzeiger aus dem Produktlinienmodell ausgeleitet.

Abbildung 41 Ausleitung.



Die Begriffe Überlagerung und Ausleitung wurden an dieser Stelle nur grob eingeführt. Produktlinienmodelle können auch direkt variant modelliert werden, ohne dass vorher Produktmodelle vorhanden sein müssen. Es ist auch möglich, aus Produktlinienmodellen wiederum Produktlinienmodelle auszuleiten, wenn noch nicht alle Varianten beseitigt werden. Diese Vorgehensweise hat aber keinen Einfluss auf die am Ende entstehenden varianzfreien Produktmodelle.

12.2 Anforderungsmodell

MOSES-Modelle verwenden zwei Konzepte, um Varianz darzustellen: Im Anforderungsmodell werden Relationen zwischen den Anforderungen benutzt, um Varianz zu modellieren. Die restlichen Modelle enthalten ein Varianzkonzept, dass auf Varianzpunkten mit Varianten und Optionen basiert.

Anforderungen sind Sätze, die in eine Struktur eingeordnet sind. Sie drücken eindeutige Sachverhalte – Anforderungen – aus, die vom System erfüllt werden müssen. Innerhalb einer Anforderung gibt es daher keine Möglichkeit mit *oder* oder *vielleicht* Varianz auszudrücken. Für diesen Zweck wurden in MOSES mehrere Ausdrucksmittel geschaffen:

- das Notwendigkeitsattribut der Konkretisierungsrelation
- die Annotation von Dimensionsattributen
- die *needs*-Relation
- die *excludes*-Relation

Das Notwendigkeitsattribut der Konkretisierungsrelation

Die Aufteilung der Anforderungen in Kunden- und Systemanforderungen bedingte, dass eine Relation zwischen Kunden- und Systemanforderungen eingeführt wurde: die Konkretisierung. Diese Konkretisierungsrelation bekommt nun ein Notwendigkeitsattribut. Dieses Attribut sagt aus, ob eine Systemanforderung für die Konkretisierung der Kundenanforderung obligatorisch (notwendig) oder nur optional ist. Obligatorische Anforderungen werden auch als Serienausstattung, optionale als Sonderausstattung bezeichnet.

Damit müssen auf Seiten der Systemanforderungen die obligatorischen Systemanforderungen gewählt werden, wenn die zugehörige Kundenanforderung gewählt wurde. Bei optionalen Systemanforderungen besteht eine weitere Wahlmöglichkeit. Wichtig ist dabei, dass die Systemanforderungen nur im Kontext der Konkretisierungsrelation optional sind. Da sie mehreren Kundenanforderungen zugeordnet werden können, können sie durchaus für eine Kundenanforderung optional sein, aber für eine andere obligatorisch.

Tabelle 6 Notwendigkeitsattribute (Beispiele).

Kundenanforderung	Systemanforderung	Notwendigkeit
Die Fahrtrichtungsanzeiger werden durch Betätigen des Lenkstockhebels aktiviert.	Die Fahrtrichtungsanzeiger [links/rechts] werden durch Betätigen des Lenkstockhebels nach [oben/unten] aktiviert.	obligatorisch
Ein Anhänger kann angehängt werden.	Eine Anhängerdisplayleuchte ist vorzusehen.	optional
	Ein Anhängerstecker ist vorzusehen.	optional

Die Annotation von Dimensionsattributen

Das Notwendigkeitsattribut kennzeichnet zwar Serien- und Sonderausstattung, es wird jedoch noch nicht beschrieben, für welche Fahrzeuge diese Kennzeichnung gilt. Dafür wurden Dimensionsattribute geschaffen, die zusätzlich zu den Notwendigkeitsattributen an Konkretisierungsrelationen annotiert werden.

Dimensionsattribute kennzeichnen die Fahrzeuge, für die diese Anforderung zu berücksichtigen ist. Ein Dimensionsattribut in MOSES ist ein 6-Tupel, das zur Zeit stark auf die Bedürfnisse von BMW zugeschnitten ist, jedoch in den Dimensionen schnell angepasst werden kann. Es umfasst folgende Dimensionen:

Tabelle 7 Dimensionen in MOSES.

Dimension	Beispiel
Entwicklungsbaureihe	{3er, 5er, . . . }
Karosserietyp	{Limousine, Touring, . . . }
Position Lenkrad	{LL, LR}
Motorvariante	{Benzin/1596, Diesel/1995, . . . }
Getriebe	{Automatik, SMG, . . . }
Länderausführung	{Europa, USA, Asien}

Ein Beispiel für ein Dimensionsattribut nach Tabelle 7 ist:
`(5er, Cabrio, LL, Benzin/1995, Automatik, Europa)`

Die Dimensionsattribute sind nun vollständig zu annotieren, das heißt, für jede Konkretisierung sind alle Dimensionsattribute anzugeben, die für sie relevant sind. Da diese Art der Annotation zu sehr vielen Attributen führt, die an eine Relation zu schreiben sind, können abkürzende Schreibweisen benutzt werden:

Tabelle 8 Abkürzende Notation für Dimensionsattribute.

Bedeutung	Notation	Beispiel
Wertebereich von *x* bis *y*	x – y	Benzin/1596 – Benzin/1995
Aufzählung von Werten	x; y; z	Europa; Asien
beliebige Werte	*	

Das Beispiel in Tabelle 9 greift Tabelle 6 vereinfacht auf, die Kundenanforderungen wurden aus Platzgründen weggelassen. Die erste Anforderung wird als Serienausstattung für alle Fahrzeuge definiert. Der Einbau des Anhängersteckers ist Sonderausstattung. In Europa wird zusätzlich noch eine Anhängerdisplayleuchte verbaut.

Tabelle 9 Komplette Annotation (Beispiele).

Systemanforderung	Notwendigkeit	Dimension
Die Fahrtrichtungsanzeiger [links/rechts] werden durch Betätigen des Lenkstockhebels nach [oben/unten] aktiviert.	obligatorisch	(*, *, *, *, *, *)
Ein Anhängerstecker ist vorzusehen.	optional	(*, *, *, *, *, *)
Eine Anhängerdisplayleuchte ist vorzusehen.	optional	(*, *, *, *, *, Europa)

Damit können die Anforderungen an die Fahrzeuge über die Dimensionen genau beschrieben und eingeordnet werden. Eine automatisierte Ausleitung aufgrund der Dimensionsattribute ist möglich, dafür ist ein Ziel-Dimensionsattribut anzugeben, mit dem die vorhandenen Attribute verglichen werden. Findet sich das Zielattribut in der Menge der annotierten Attribute, so kann die damit verbundene Systemanforderung in das Produktmodell übernommen werden.

Die abkürzende Notation ist mit Bedacht einzusetzen, da vor allem in Verbindung mit der *excludes*-Relation inkonsistente Modelle annotiert werden können, wenn die Wertebereiche von Relationen nicht sorgfältig getrennt wurden.

Die *needs*-Relation

Die Attributierung der Konkretisierung dient der Auswahl von Anforderungen nach gewünschtem Fahrzeug. Dabei wurde noch nicht berücksichtigt, dass Anforderungen einander bedingen oder ausschließen können. Diese Beziehungen werden über die zwei Relationen *needs* und *excludes* ausgedrückt.

Die *needs*-Relation drückt aus, dass eine Relation eine andere benötigt, um sinnvoll genutzt werden zu können. Die *needs*-Relation ist gerichtet und trifft nur Aussagen in eine Richtung:

Tabelle 10 *needs*-Relation (Beispiel).

Systemanforderung	Relation	Systemanforderung
Die Fahrtrichtungsanzeiger [links/rechts] werden durch Betätigen des Lenkstockhebels nach [oben/unten] aktiviert.	*needs*	Ein Lenkstockhebel wird eingebaut.

Die Benutzung der Fahrtrichtungsanzeiger erfordert einen Lenkstockhebel. Dieser Hebel kann jedoch unabhängig von der Blinkfunktion eingebaut werden (z. B. für Lichtumschaltung).

Für die Ausleitung und damit für die Annotation bedeutet die *needs*-Relation eine Einschränkung, denn die von einer Anforderung benötigte Anforderung muss mit der ersten Anforderung zusammen ausgeleitet werden. Ist das nicht der Fall, ist die Annotation nicht konsistent, es könnten ungültige Produktmodelle entstehen.

Die *excludes*-Relation

Die *excludes*-Relation ist das Gegenteil der *needs*-Relation. Zwei Anforderungen, die in einer *excludes*-Relation zueinander stehen, dürfen nicht gemeinsam in einem Produktmodell auftreten. Sie sind gemeinsam nur in einem Produktlinienmodell zulässig. Im Gegensatz zur *needs*-Relation wirkt die *excludes*-Relation in beide Richtungen.

Tabelle 11 *excludes*-Relation (Beispiel).

Systemanforderung	Relation	Systemanforderung
Die Farbe der Leuchten ist weiß.	*excludes*	Die Farbe der Leuchten ist gelb.

Werden laut Tabelle 11 weiße Leuchten verbaut, können nicht gleichzeitig gelbe verbaut werden und umgekehrt. Wie die *needs*-Relation hat auch die *excludes*-Relation Einfluss auf die Annotation. Zwei Anforderungen, die in einer *excludes*-Relation zueinander stehen, dürfen nicht gemeinsam in einem Produktmodell auftreten.

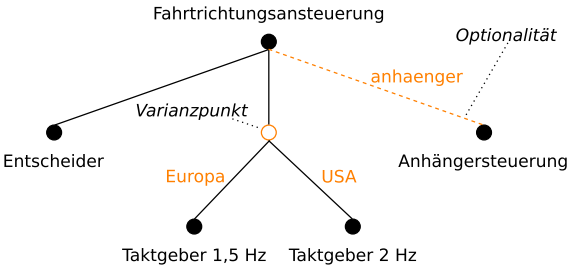
12.3 Varianzpunkte

Wie bereits erwähnt, verwenden die logische Architektur, die technische Architektur und die Partitionierung das Konzept der Varianzpunkte, um Varianz zu beschreiben. Dieses Konzept basiert auf dem Gedanken, dass Varianz nur unterhalb bestimmter Varianzpunkte oder als Optionalität auftreten darf. Das Konzept ist generisch einsetzbar, für die verschiedenen Architekturen muss nur definiert werden, welche Architekturelemente in welcher Weise variieren dürfen. Daher wird im Folgenden kurz das Konzept an sich vorgestellt und nachfolgend die Umsetzung in den Architekturmodellen.

12.3.1 Das Varianzpunktkonzept

Das Varianzpunktkonzept sieht zwei Arten von Varianz vor: Optionalität und XOR-Varianz (Abbildung 42). Optionalität bedeutet, dass ein als optional gekennzeichnetes Element entweder im ausgeleiteten Produktmodell vorkommt oder nicht. XOR-Varianz bedeutet, dass unterhalb eines Varianzpunkts Varianten deklariert werden, von denen genau eine im ausgeleiteten Produktmodell benutzt werden muss.

Abbildung 42 Varianzpunkte und Optionalität.



Die Optionalität bezieht sich immer auf ein Element und alle seine Unterelemente. Dabei ist ein Element nur in seinem Kontext optional, das heißt, es ist als Unterelement seines aggregierenden Elements optional. Die Annotation der Ausleitungsinformationen erfolgt also an der Relation (Kante) von Oberelement zu Unterelement. Damit ist es möglich, dass ein Element in einem Kontext optional sein kann und in einem anderen nicht.

Als Varianzpunkte werden diejenigen Elemente bezeichnet, unterhalb derer zwischen mehreren Varianten gewählt werden kann. Ein Varianzpunkt gibt dabei die Hülle vor, die alle Varianten erfüllen müssen, um an Stelle des Varianzpunkts eingesetzt zu werden. Ein Varianzpunkt kann also als Typ gesehen werden und seine Varianten als Instanzen, diese Analogie ist allerdings nur eine Denkhilfe. Die Ausleitungsinformationen werden an die Beziehungen zu den Varianten annotiert.

Annotation

Als Ausleitungsinformationen werden Entscheidungsprädikate annotiert. Anhand der Entscheidungsprädikate kann entschieden werden, welche Variante bzw. ob eine Option gewählt wird. Ein Entscheidungsprädikat gibt an, auf welche Menge von Fahrzeugidentifikatoren (VID = *vehicle id*) es zutrifft. Diese VIDs charakterisieren konkrete Fahrzeuge. Ein VID ist eine Liste von Ausstattungsmerkmalen, die in einem Fahrzeug verbaut werden. Diese Ausstattungsmerkmale teilen sich in

typbestimmende Merkmale und Sonderausstattungen. Damit sind alle Informationen, die für die Ausleitung eines Fahrzeugs nötig sind, vorhanden: Serien- und Sonderausstattung.

Tabelle 12 Fahrzeugidentifikatoren (VIDs).

Beschreibung	VID
Europäer	europa = (*, *, *, *, *, Europa)
Europäer mit Sportpaket	((*, *, *, *, *, Europa), Sportpaket) (europa, Sportpaket)
Benziner	benziner = (*, *, *, Benzin/*, *, *)
Europäische Benziner	europa AND benzin

Im Gegensatz zu den Dimensionsattribute der Anforderungen sind die Entscheidungsprädikate, die sich auf die VIDs beziehen, logisch kombinierbar. Das heißt, es kann mit logischen Operatoren gearbeitet werden, die zum einen die Annotation für Menschen verständlicher machen, zum anderen automatische Ausleitung unterstützen.

Entscheidungszeitpunkt

Der Zeitpunkt, an dem die Varianz aufgelöst wird, ist in der Praxis eindeutig festzulegen, um den Ausleitungsprozess geordnet in den Produktentstehungsprozess einzugliedern. Andere Modelle sehen dafür explizite Metamodellelemente vor. MOSES definiert den Auflösungszeitpunkt nicht, diese Aufgabe wird dem Prozessmodell überlassen. Das hat vor allem pragmatische Gründe: Zum einen ist eine solche Annotation vom vorhandenen Entwicklungsprozess zu unterstützen, damit Informationen nicht redundant gehalten werden. Zum anderen wird das Modell um zusätzliche Informationen erweitert, die erhoben und verwaltet werden müssen.

Beide Gründe führten dazu, dass der Entscheidungszeitpunkt nicht explizit berücksichtigt wurde. Wenn das Modell erweitert wird, kann auch hier eine andere Entscheidung gefällt werden.

Relationen zwischen Elementen

Die Relationen zwischen den Elementen, wie sie bei den Anforderungen definiert wurden, gibt es im Varianzpunktkonzept nicht. Die Modellierung sieht vor, Entscheidungen über die vorhandenen Mittel Varianzpunkt und Optionalität zu fällen. Trotzdem kann der Fall auftreten, dass Elemente aus verschiedenen Hier-

archien einander bedingen oder ausschließen. Dann gibt es drei Möglichkeiten, damit umzugehen:

1 Einführung eines Varianzpunkts

  Ein neuer Varianzpunkt wird eingeführt, wenn das Modell es erlaubt. Das heißt, die in Varianz zueinander stehenden Elemente sollten auch im Modell etwas miteinander zu tun haben, da sonst nicht nachvollziehbare Varianzpunkte entstünden. Mit neuen Varianzpunkten können gegenseitige Ausschlüsse modelliert werden.

2 Implizite Relationen über horizontale Verbindungen

  Elemente, die einander bedingen, stehen oft in horizontaler Verbindung miteinander, das heißt, sie kommunizieren miteinander. Ist das der Fall, so kann über geschickte Modellierung dieser Verbindungen erreicht werden, dass diese beiden Elemente gemeinsam ausgeleitet werden.

3 Relation über Annotation

  Kann keine der beiden oben genannten Alternativen verwendet werden, so können immer noch entsprechende Annotationen an die im Beziehung stehenden Elemente gesetzt werden. Bei einander bedingenden Elementen wären das gleiche Annotationen, bei sich ausschließenden Elementen logisch einander ausschließende Annotationen. Dabei ist darauf zu achten, dass bei Änderungen die jeweils komplementäre Annotation auch geändert wird. Dazu muss notiert werden, welche Annotationen in Beziehung miteinander stehen.

12.3.2 Logische Architektur

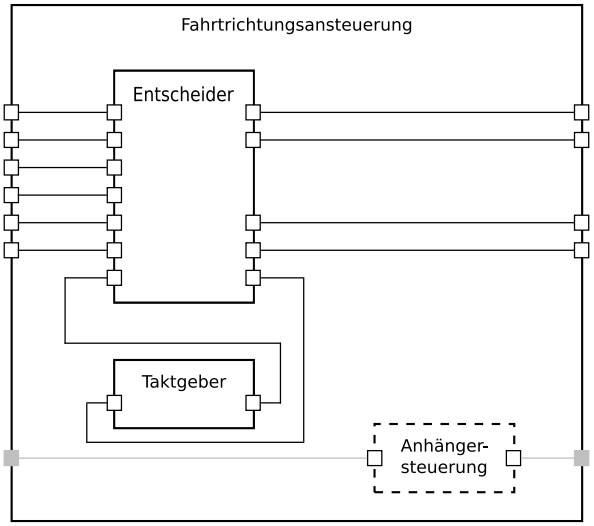
In der logischen Architektur können alle Elemente variant sein:

- Funktionen
- Ports
- Verbindungen
- Signale bzw. Operationen

Sie unterliegen einigen Einschränkungen, um die Varianz handhabbar zu halten. Diese Einschränkungen sind im Metamodell-Dokument ausführlich beschrieben, an dieser Stelle werden nur die wichtigsten Bedingungen genannt.

Funktionen können jegliche Varianz enthalten, sie unterliegen als zentrale Elemente der logischen Architektur keinen Einschränkungen. Variante Ports können nur an Funktionen vorkommen, die selber variant sind. Analoges gilt für variante Operationen und Signale, auch sie können nur bei varianten Ports vorkommen. Variante Verbindungen kennen solche Abhängigkeiten nicht, für sie gilt jedoch, dass die Anfangs- und Endports zueinander varianter Verbindungen gleich sein müssen.

Abbildung 43 Varianz und abhängige Ports (Beispiel).



Ein weiteres Konzept soll an dieser Stelle vorgestellt werden: das Konzept der abhängigen Ports. Abhängige Ports wurden eingeführt, um horizontale Abhängigkeiten modellieren zu können, ohne die Varianz zu »hoch« in der Hierarchie modellieren zu müssen. Sei der Fall gegeben, dass zwei Funktionen miteinander über Oberfunktionsgrenzen hinweg kommunizieren. Dafür werden an den Oberfunktionen entsprechende Ports vorgesehen. In einem anderen Produktmodell kommunizieren diese Funktionen nicht miteinander. Der zugehörige Varianzpunkt müsste über die beiden Oberfunktionen gesetzt werden, die jedoch eigentlich nicht zueinander variant sind. In MOSES kann die Varianz nun für die beiden Unterfunktionen modelliert werden, der nur in einem Modell vorhandene Port wird als abhängiger Port modelliert. Jetzt kann schon auf der oberen Hierarchieebene erkannt werden, dass ein Port abhängig von einer Entscheidung in einer tieferen Hierarchie ist.

Abbildung 43 zeigt ein Beispiel für eine optionale Funktion *Anhängersteuerung*. Sie kommuniziert über Ports der Funktion *Fahrtrichtungsansteuerung* mit anderen Funktionen. Diese Ports sind nur vorhanden, wenn die Funktion *Anhängersteue-*

rung gewählt wird; daher sind diese Ports abhängig von dieser Funktion. Im Bild werden abhängige Ports (und deren Verbindungen) grau gezeichnet.

12.3.3 Technische Architektur

Die Varianz der technischen Architektur bleibt auf den Hardwareteil der technischen Architektur beschränkt. Dies hat pragmatische Gründe, da die Varianz von Software im Rahmen von MOSES nicht betrachtet wurde.

Die technische Architektur sieht Varianz vor für:

- Hardwareelemente
- Verbindungsenden
- Verbindungen

Es fällt auf, dass Ports nicht in die Varianz einbezogen werden. Das ist nicht nötig, da die Portverbindungsenden als Schnittstellen der Verbindungen variant sein können. Damit können Ports über die Verbindungsenden Anschlussvarianz bekommen.

Der Treiber bei der Varianz auf technischer Seite sind vor allem betriebswirtschaftliche und produktionsbedingte Überlegungen. So ist Herstellern an der Identifikation von Gleich- und Ähnlichteilen gelegen. Des Weiteren spielt die Varianz von Topologiemodellen in der Definition von Bussystemen eine starke Rolle. Diese Motivation wurde in MOSES aufgenommen und für die Varianzdefinition benutzt. Damit wird Varianz eingeteilt in mögliche Varianz bei Gleichteilen, Ähnlichteilen und sogenannten Andersteilen.

Tabelle 13 Varianzmöglichkeiten der Hardware.

Klassifikation	Mögliche Varianz
Gleichteile	Attributvarianz von Hardwareelementen eines Steuergeräts Attributvarianz des gesamten Steuergeräts Attributvarianz von Hardwareports des Steuergeräts Anschlussvarianz von Hardwareports des Steuergeräts bei gleichen Pins
Ähnlichteile	Gleichteilvarianz sowie optionale Hardwareelemente (und damit abhängige Ports) optionale Ports
Andersteile	jegliche Varianz möglich

12.3.4 Partitionierung

Die Varianz von Partitionierungen sieht keine Optionalität vor, da Funktionen, die nicht partitioniert werden, auch nicht nötig sind. Damit ist nur noch XOR-Varianz über Varianzpunkte interessant. Damit werden Varianten modelliert, bei denen Funktionen in unterschiedlichen Fahrzeugen auf unterschiedlichen Steuergeräten laufen. Diese Varianz ist so wichtig, weil sich damit der Kreis der frei verteilbaren Funktionen schließt. Um Funktionen wirklich frei zu verteilen, müssen sie variant partitioniert werden können.

Damit ist die eine Seite der Varianz beschrieben – variante Partitionierungen. Auf der anderen Seite ergeben sich durch die Varianz der logischen und technischen Architektur Auswirkungen auf die Bedeutung der Partitionierung, wenn variante Anteile als Quelle oder Ziel beteiligt sind. Dafür wurden folgende Bedeutungen festgelegt:

Tabelle 14 Partitionierung varianter Elemente.

Quelle	Ziel	Bedeutung
Varianzpunkt	beliebig	alle Varianten der partitionierten Funktion werden auf das Ziel partitioniert, müssen also auf diesem laufen können
beliebig	Varianzpunkt	*all:* die Funktion wird auf alle Varianten partitioniert, sie muss auf allen Varianten lauffähig sein *some:* die Funktion wird auf mindestens eine Variante partitioniert, sie muss auf dieser Variante lauffähig sein

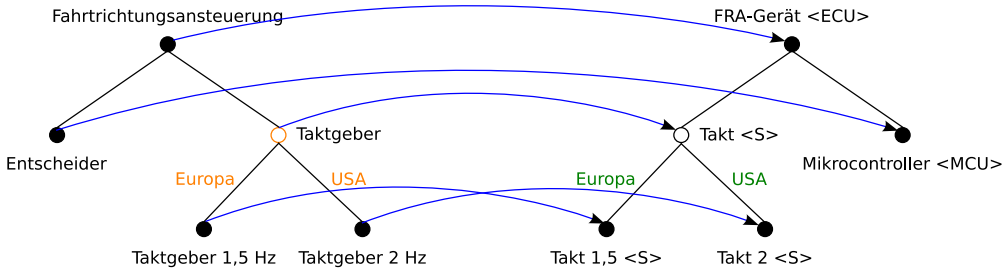
Die zweite Variante (*some*) ist nur für mehrere partitionierte Funktionen sinnvoll, für die ein allgemeines Partitionierungsziel angegeben wird, das für einige Funktionen in nicht allen Varianten als Ziel gültig ist.

12.4 Annotation im Kontext der Partitionierung

Die Annotation der Entscheidungsprädikate erfolgt in der logischen Architektur, in der technischen Architektur und in der Partitionierung. Die Informationen, die annotiert werden, sind teilweise abhängig voneinander, teilweise unabhängig. Das kann dazu ausgenutzt werden, annotierte Informationen gegeneinander auf Konsistenz zu prüfen oder um fehlende Annotationen in den Modellen zu ergänzen.

Ist z. B. die Annotation der logischen Architektur bereits erfolgt, so kann über die Partitionierung diese Information in die technische Architektur übernommen werden. Die Idee dahinter ist, dass eine Partitionierung bedeutet, dass eine Funktion auf einer bestimmten Hardware läuft. Wird diese Funktion über ein Entscheidungsprädikat ausgewählt, muss auch die Hardware vorhanden sein, auf der sie laufen soll. Daher können die Annotation übernommen bzw. die bestehende Annotation überprüft werden.

Abbildung 44 Übernahme der Annotation aus der logischen Architektur.



In Abbildung 44 werden die Taktgeberfunktionen auf die entsprechenden Varianten der technischen Architektur partitioniert. Da die Annotation der technischen Architektur bereits erfolgte (orange), kann die Annotation in die technische Architektur übernommen werden (grün).

Anders liegt der Fall bei der Rückrichtung. Hardware kann eingebaut werden, ohne dass eine bestimmte Funktion darauf laufen muss. Wenn z. B. die Funktion variant partitioniert wird, kann die Hardware verbaut werden und diese Funktion trotzdem nicht darauf laufen. Daher kann die Partitionierungsinformation nicht übernommen werden. Es ist lediglich eine negative Aussage möglich: wenn das Hardwareelement nicht ausgewählt wird, kann auch die Funktion nicht darauf laufen.

13 Domäne

Die dritte Erweiterung der Modelle ist die Modellierung von Domänenmodellen. Domänenmodelle speichern langfristiges Erfahrungswissen und stellen Bausteine, Teillösungen oder Konzepte zur Verfügung. Daher müssen in Domänenmodellen auch Teilmodelle abgelegt werden können, die als genau diese Bausteine oder Teillösungen fungieren.

Auch hier wurden für Anforderungen und die restlichen Modelle unterschiedliche Ansätze gewählt. Für Anforderungen wird das Kontextkonzept verwendet, für die anderen Modelle das Platzhalterkonzept. Für Partitionierungen wurden keine expliziten Domänenmodellerweiterungen geschaffen.

13.1 Anforderungen

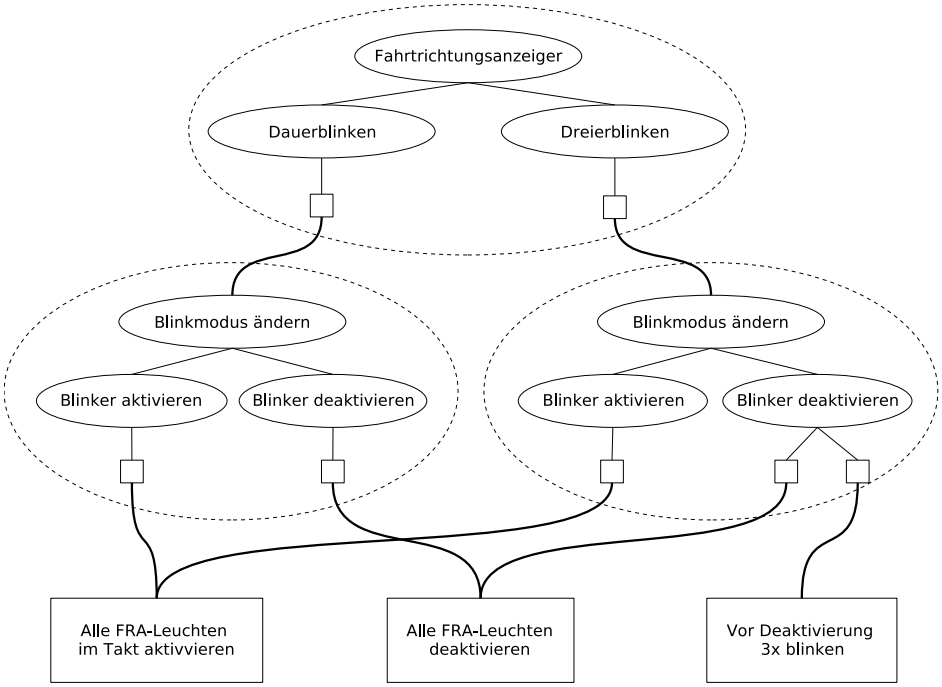
Anforderungen im Domänenmodell sind wie im Produktmodell Bäume. Sie entstehen aus den Produktmodellen, indem sogenannte »Kontexte« ausgezeichnet werden. Ein Kontext besteht aus gekennzeichneten Strukturknoten, die zu einem bestimmten Bereich, dem Kontext, gehören. In Abbildung 45 werden drei Kontexte ausgezeichnet: *Fahrtrichtungsanzeiger* sowie zweimal *Blinkmodus ändern*.

Die Anforderungen werden anhand der Kontexte in Teilmodelle zerschnitten. Die Verbindung zwischen Teilmodellen selbst und zwischen Teilmodellen und den Anforderungen erfolgt über spezielle Relationen. Jetzt ist es möglich, Anforderungen verschiedenen Kontexten zuzuordnen oder Kontexte untereinander auszutauschen, sie zu verfeinern, zu vergröbern etc. Die Anforderungen werden also unabhängig von den Kontexten gehalten und somit flexibel wiederverwendbar. Gleichzeitig bieten die Kontextbäume Musterlösungen für bestimmte Probleme, die wiederverwendet werden können.

Im Domänenmodell werden zusätzliche Bearbeitungsinformationen erhoben. Für jedes Element – Strukturierungsknoten oder Anforderung – kann ein Bearbeitungsstatus angegeben werden, der vom Domänenverwalter gesetzt wird und aussagt, ob die veröffentlichte Teilversion stabil ist oder sich noch in Bearbeitung befindet.

Eine Information, die aus den Produktmodellen übernommen werden muss, ist die Konfigurationsinformation, die annotiert wurde. Für Anforderungsmodelle

Abbildung 45 Kontexte und Anforderungen.



sind das die Dimensionsattribute, die in das Domänenmodell einfließen sollen. Im Domänenmodell liegen bewährte Lösungen, also Anforderungen bzw. Strukturknoten, die in mehreren Produkten oder Produktlinien verwendet wurden. Genau diese Information wird in Historylisten verwaltet, dem Analogon zu den Dimensionsattributen. In einer Historyliste werden alle Dimensionsattribute gesammelt, mit denen das Element annotiert wurde.

13.2 Platzhalter

Die Modelle der logischen und technischen Architektur verwenden das Platzhalterkonzept (*hook*-Konzept). Dabei werden im Domänenmodell Platzhalter definiert, an deren Stelle Elemente mit passender Schnittstelle eingesetzt werden können.

Wie bei den Anforderungen besteht das Problem, Teilmodelle im Domänenmodell abzulegen, ohne zu viele innere Beziehungen zu verlieren. Dazu werden die Teilmodelle identifiziert, die in der Domäne abgelegt werden sollen und aus dem

Gesamtmodell herausgeschnitten. Das oberste Element – die Wurzel des Teilmodells – dient als Anknüpfungspunkt des Teilmodells nach oben. Nach unten können zwei Fälle auftreten: entweder wurden Unterelemente abgeschnitten oder nicht. Wenn Unterelemente abgeschnitten wurden, wird an ihrer Stelle ein Platzhalter für die ursprüngliche Funktion modelliert. Damit bleibt die Schnittstelle erhalten, welche Funktion eingesetzt werden kann. Mit diesem Platzhalter können nun die Wurzeln von Teilmodellen verbunden werden, die an diese Stelle passen.

Mit Hilfe dieser Konzepte können Domänenmodelle aufgebaut werden. Die Teilmodelle entsprechen Musterlösungen, die in Produktmodellen verwendet werden können. Bei der Verwendung können die Teilmodelle mit oder ohne die ihnen anhängenden Modelle benutzt werden. Weiterhin können die benutzten Teilmodelle als Muster verwendet werden, dann werden sie im Produktmodell verändert. Die Teilmodelle können auch als Bausteinreferenz verwendet werden, dann werden sie so benutzt, wie sie sind, ohne Änderungen zuzulassen. Einen genaueren Überblick über die Verwendung der Modelle sowie Beispiele gibt Abschnitt 14.3 im folgenden Teil IV.

Platzhalterelemente

Die logische Architektur kann an den Funktionen aufgetrennt werden. Dafür werden Funktionsplatzhalter zur Verfügung gestellt. Infolge einer solchen Auftrennung können auch die Ports zu Platzhaltern werden.

Die technische Architektur kann an den Komponenten aufgetrennt werden, es entstehen Komponentenplatzhalter. Analog zur logischen Architektur können dann auch Ports zu Platzhaltern werden.

Teil IV

Produktlinien und Domänenmodelle

MOSES unterstützt nicht nur Produktmodelle, sondern auch die Modellierung von Produktlinien und Domänenmodellen. Nachdem im vorigen Teil III die Modellierungselemente für Produktlinien (Varianz) und Domänenmodelle (Kontexte und Platzhalter) vorgestellt wurden, erläutert dieser Teil den Umgang mit den Modellerweiterungen.

In MOSES werden drei Ebenen von Modellen unterschieden, Domänenmodelle, Produktlinienmodelle und Produktmodelle.

Domänenmodelle sind die allgemeinste, abstrakteste Beschreibung. Sie beschreiben Domänen, das heißt, Klassen von gemeinsamen Modellen. Im Automobilbau ist die Domäne die Gesamtheit aller Fahrzeuge des Herstellers, das Domänenmodell enthält also das fahrzeugübergreifende Wissen des Herstellers.

Eine Stufe tiefer stehen die *Produktlinienmodelle*. Sie modellieren ganze Produktlinien, also Fahrzeuge, die eine gemeinsame Basis besitzen und deshalb gemeinsam modelliert werden. Aus Produktlinienmodellen sollen automatisiert konkrete Produktmodelle ausgeleitet werden.

Produktmodelle sind Fahrzeugmodelle. Sie modellieren genau ein Fahrzeug ohne Varianz. Sie dienen als Grundlage zur Fertigung eines Fahrzeugs.

Tabelle 15 Übersicht der Modelle.

Modell	Fokus	Modellierungsmittel
Domänenmodell	Domäne (gesamter Wissensbereich)	wie Produktlinienmodell plus Platzhalter
Produktlinienmodell	Fahrzeugfamilien	wie Produktmodell plus Varianz
Produktmodell	Fahrzeuge	Komponenten, Strukturierung, Hierarchie

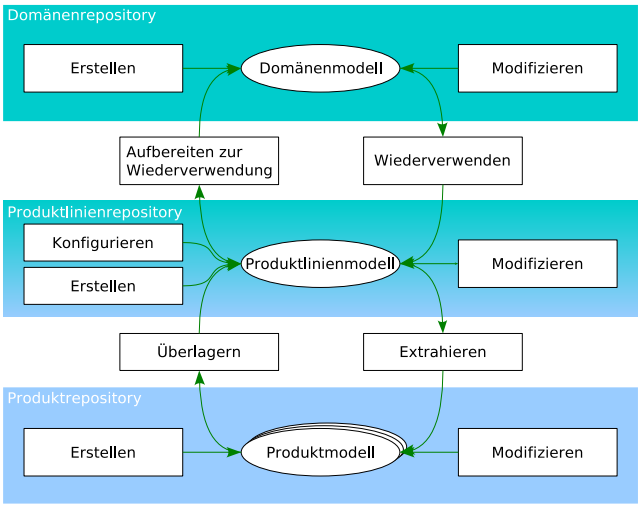
Damit sind die Modellarten statisch und inhaltlich charakterisiert. Der Übergang von einer Modellart zur anderen ist aber noch nicht erläutert worden und Gegenstand des folgenden Kapitels.

14 Umgang mit Produktlinien und Domänenmodellen

Zur Beschreibung des Umgangs mit Produktlinien und Domänenmodellen gehört vornehmlich eine Beschreibung der Prozesse, wie von einem Modell zum anderen gelangt werden kann. Zusätzlich müssen die Aktivitäten erläutert werden, die die Modelle erstellen bzw. ändern.

Wie im vorigen Kapitel erwähnt, besitzen die verschiedenen Modellarten nicht nur gemeinsame Modellelemente, sie bauen aufeinander auf. Das heißt, Informationen können von einem Modell zum anderen übernommen werden. Diese Informationsübernahme muss Regeln folgen, damit in jedem Modell Konsistenz herrscht. Ein Überblick über die möglichen Aktivitäten zwischen den Modellen ist in Abbildung 46 zu sehen.

Abbildung 46 Modelle und Aktivitäten.

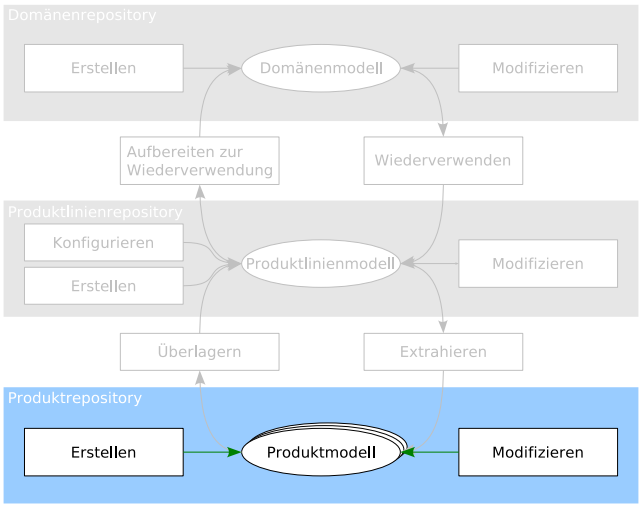


Die Beschreibung der Aktivitäten erfolgt von unten beim Produktmodell beginnend über das Produktlinienmodell bis nach oben zum Domänenmodell. Dabei wird in der Beschreibung davon ausgegangen, dass die höheren Modelle noch unbekannt sind, zusätzliche Modellierungsmöglichkeiten werden also nach Einführung der entsprechenden Modelle erläutert.

14.1 Das Produktmodell

Das Produktmodell ist das Modell eines Fahrzeugs. Es enthält keine Varianz oder Platzhalter. Damit ist das Produktmodell das Modell, das bisher als Fahrzeugmodell modelliert wurde. In Abbildung 47 sind die Aktivitäten für Produktmodelle hervorgehoben.

Abbildung 47 Aktivitäten für Produktmodelle.



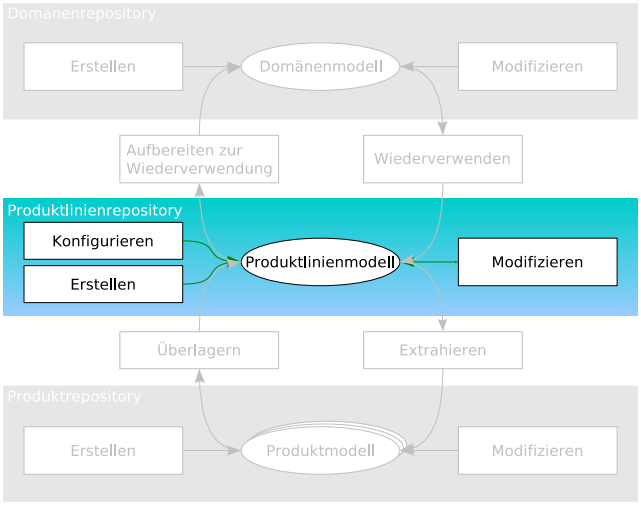
Produktmodelle können entworfen und geändert werden. Der Entwurf erfolgt entweder von Grund auf neu oder durch Modifikation bekannter Modelle. Es gibt typischerweise viele verschiedene Produktmodelle, so viele, wie Fahrzeuge produziert werden sollen. Dabei sind die Verbindungen zwischen den einzelnen Produktmodellen nicht modelliert, das heißt, Änderungen in einem Produktmodell sind unabhängig von gleichen Elementen in anderen Produktmodellen. Für Produktmodelle sind Produktverantwortliche zuständig, beispielsweise Entwickler, Systemverantwortliche oder Integratoren.

14.2 Das Produktlinienmodell

Das Produktlinienmodell ist das Modell einer Fahrzeugfamilie. Welche Produkte zu einer Produktlinie zusammengefasst werden, liegt im Ermessen des Herstellers. Oft wird gefordert, dass möglichst viele Gleich- oder Ähnlichteile in einer Produktlinie erfasst werden. Ein Produktlinienmodell enthält Varianz, aber keine

Platzhalter. In Abbildung 48 sind die Aktivitäten für Produktmodelle hervorgehoben.

Abbildung 48 Aktivitäten für Produktlinienmodelle.



Das Produktlinienmodell kann erstellt oder modifiziert werden. Die Pflege des Produktlinienmodells wird von einem Produktlinienverantwortlichen vorgenommen. Der Produktlinienverantwortliche ist für die Konsistenz des Produktlinienmodells verantwortlich. Er achtet insbesondere darauf, dass nicht nur das Modell sondern auch die Annotation konsistent ist.

Die Erstellung eines Produktlinienmodells kann von Grund auf neu erfolgen. Dieser Weg wird beschritten, wenn eine Fahrzeugfamilie tatsächlich neu entwickelt werden soll. Meist sind aber schon Produktmodelle oder Teillösungen vorhanden, die in einem Produktlinienmodell zusammengeführt werden sollen. So kann man die vorhandene Modellierung der Produkte für das Produktlinienmodell nutzen. Dieser Weg führt von den Produktmodellen zum Produktlinienmodell und wird als »Aufbereiten zur Wiederverwendung« bezeichnet. Das Aufbereiten identifiziert Gemeinsamkeiten und Unterschiede von Produktmodellen und überführt sie in Varianz und Annotation. Sind die Modelle genügend aufbereitet, entscheidet der Produktlinienverantwortliche über die Aufnahme der Modelle in das Produktlinienrepository.

Die Modifikation der Produktlinienmodelle kann ebenfalls auf zwei Wegen erfolgen: innerhalb des Produktlinienmodells oder über Aufarbeitung geänderter Produktmodelle. Auch die Modifikation ist vom Produktlinienverantwortlichen auf Konsistenz der Modelle und Annotation zu überwachen.

Ein Produktlinienmodell kann lediglich die Überlagerung verschiedener Produktmodelle sein, üblicherweise werden mit Varianz aber mehr Produktmodelle möglich als vor der Überlagerung. Das ist durchaus erwünscht, da neue Produktmodelle ausgeleitet werden können. Es ist jedoch sicherzustellen, dass die ausleitbaren Produktmodelle sinnvolle Modelle sind oder die Annotation ist zu überarbeiten.

Ein Nutzen des Produktlinienmodells ist die automatische Ausleitung von Produktmodellen. Dafür wird eine Identifikation eines Fahrzeugs dem Produktlinienmodell übergeben, von diesem ausgewertet und das entsprechende Produktmodell ausgeleitet. Ist kein Fahrzeug identifiziert worden, sondern z. B. eine Fahrzeugfamilie, kann noch nicht jegliche Varianz des Produktlinienmodells eliminiert werden, das entstehende ausgeleitete Modell ist immer noch ein Produktlinienmodell. Auf dieses teilausgeleitete Modell kann wiederum eine neue Ausleitung angewendet werden, so lange, bis die Varianz entfernt wurde.

Natürlich können die ausgeleiteten Produktmodelle weiter genutzt, das heißt modifiziert werden. Dabei ist darauf zu achten, ob die Änderungen lokal für das Produktmodell gelten sollen oder über Aufarbeitung wieder in das Produktlinienmodell zurückfließen sollen.

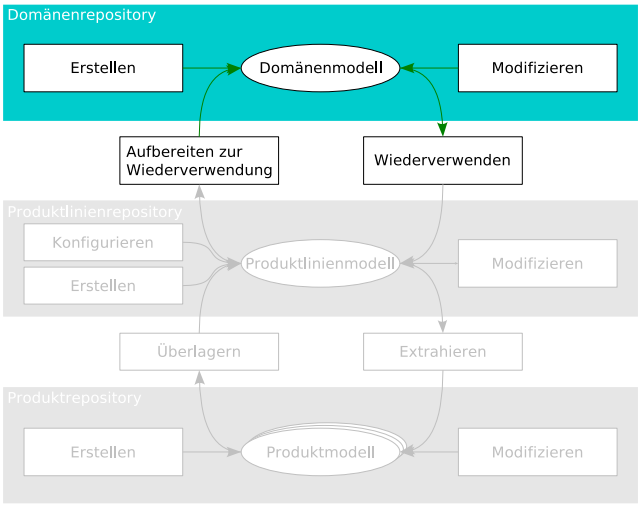
14.3 Das Domänenmodell

Das Domänenmodell ist das Modell der gesamten Domäne. Das heißt, es enthält das Wissen, das über die Domäne vorliegt. Damit ist das Domänenmodell nicht einfach eine Überlagerung von Produktlinienmodellen. Diese Überlagerung wäre zu komplex und nicht wartbar. Ein Domänenmodell soll vielmehr Muster und Teillösungen zur Verfügung stellen, die für die gesamte Domäne gelten. Ein Domänenmodell wird vom Domänenverantwortlichen betreut. Er entscheidet, welche Informationen in das Modell gehören und in welcher Weise sie modelliert werden. Die nachfolgend beschriebenen Aktivitäten für Domänenmodelle zeigt Abbildung 49.

Erstellen und Modifizieren

Die Erstellung des Domänenmodells erfolgt aufgrund der Erfahrungen mit den Produkt- und Produktlinienmodellen. In beiden werden generelle Lösungen, also Teilmodelle, identifiziert, die für die gesamte Domäne genutzt werden können. Diese identifizierten Teilmodelle werden extrahiert und, soweit nötig, mit Platzhaltern und Verbindungen zu anderen Domänenelementen versehen. Dann können die Teilmodelle in das Domänenmodell übernommen werden.

Abbildung 49 Aktivitäten für Domänenmodelle.



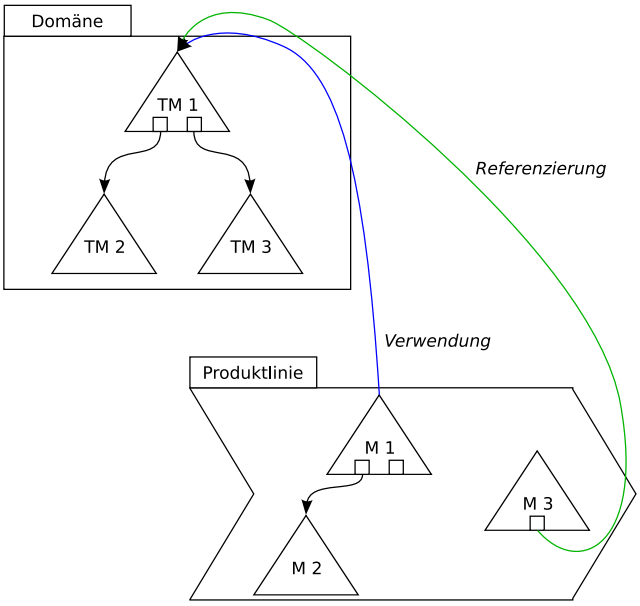
Die Modifikation des Domänenmodells folgt allgemeinen Modellierungsprinzipien. Es ist darauf zu achten, dass Änderungen Einfluss auf alle Modelle des Herstellers haben, sie sollten daher in der Praxis ausreichend erprobt und bewährt sein. Änderungen können direkt im Domänenmodell erfolgen. Da dabei aber die Test- und Absicherungsmöglichkeiten gering sind, ist der bessere Weg, die zu ändernden Modelle in ein Produktmodell zu überführen, dort zu modifizieren und zu testen und nach der Fertigstellung wieder in das Domänenmodell zu überführen.

Benutzung von Domänenelementen

Domänenelemente können in Produkt- und in Produktlinienmodellen wiederverwendet werden. Dabei werden zwei Strategien unterschieden: Verwendung und Referenzierung (Abbildung 50).

Bei der *Verwendung* werden die Domänenelemente in die unteren Modelle kopiert. Dabei wird ein Link auf das verwendete Domänenelement belassen, der jedoch nur zur späteren Identifikation des Elements dient. Die Teilmodelle, die an Platzhaltern des Domänenelements hängen, können beim Kopieren ebenfalls kopiert werden, die Platzhalter können aber auch freigelassen werden. Das so verwendete Element kann jetzt frei modifiziert werden. Insbesondere frei gelassene Platzhalter müssen gefüllt werden. Die Modifikation hat keinen Einfluss auf das Domänenmodell. Umgekehrt haben auch Änderungen im Domänenmodell keinen Einfluss auf die verwendeten Elemente.

Abbildung 50 Verwendung und Referenzierung.



In Abbildung 50 wird das Teilmodell *TM 1* als Modell *M 1* verwendet. Dabei wurde Teilmodell *TM 2* als Modell *M 2* mitverwendet, der Platzhalter, an dem Teilmodell *TM 3* hing, bleibt frei und wird später besetzt.

Anders arbeitet die *Referenzierung*. Wird ein Domänenelement in einem unteren Modell referenziert, wird eine Referenz auf das Domänenmodell angelegt. Über diese Referenz ist das Domänenelement mitsamt der an Platzhaltern hängenden Teilmodelle in das untere Modell eingebunden. Es kann im unteren Modell nicht geändert werden, da es nur eine Referenz ist. Soll das Domänenelement geändert werden, ist es zu verwenden, zu ändern und die Änderung in das Domänenmodell zurückzuschreiben. Änderungen im Domänenmodell wirken sich dabei auf alle Referenzen des Domänenelements aus.

In Abbildung 50 referenziert Modell *M 3* das Teilmodell *TM 1* der Domäne. Damit wird das gesamte Teilmodell inklusive der Teilmodelle *TM 2* und *TM 3* an Stelle des Platzhalters von Modell *M 3* gesetzt. Dabei bleibt das referenzierte Teilmodell in der Domäne und kann so nicht verändert werden.

Teil V

Benutzung von MOSES in der Praxis

Dieser Teil beschäftigt sich mit der praktischen Anwendung von MOSES. Dabei werden vier verschiedene Aspekte vorgestellt, die während des MOSES-Projekts aufgefallen sind.

Zunächst wird das Sichtenkonzept vorgestellt, das erläutert, welche verschiedenen Sichten auf ein Modell eingenommen werden können. Danach wird kurz diskutiert, wie Versionierung mit MOSES-Modellen zusammenpasst. Anschließend wird der MOSES-Prozess vorgestellt, den Abschluss bildet die Umsetzung der MOSES-Methodik in die Praxis.

15 Sichten

Das Sichtenkonzept wird ausführlich in MOSES 3-2 [Fra04a] vorgestellt. Es handelt sich dabei um ein Konzept, wie die hierarchischen Modelle der logischen und technischen Architektur betrachtet werden können. Die Erörterung ist als Hinweis zum Umgang mit den Modellen zu verstehen sowie als Anforderung an ein zukünftiges Werkzeug zur Modellierung der MOSES-Modelle.

Im Sichtenkonzept werden zwei Arten von Sichten unterschieden: Lupensichten und Projektionssichten. Lupensichten betrachten zusammenhängende Ausschnitte eines Modells, Projektionssichten wählen den zu betrachtenden Ausschnitt anhand eines Kriteriums aus.

Lupensichten

Lupensichten zeigen Ausschnitte eines Modells, die wie mit einer Lupe vergrößert bzw. verkleinert werden können. Je nach Stärke der Lupe werden mehr oder weniger Modellteile bzw. Details angezeigt. Eine Lupensicht ist mit dem aus grafischen Werkzeugen bekannten *Zoom* vergleichbar.

Der Zweck einer Lupensicht ist die Betrachtung eines zusammenhängenden Ausschnitts eines Modells. Das heißt, sowohl die horizontalen als auch die vertikalen Verbindungen des betrachteten Modells werden in der Sicht mit angezeigt. Der Nutzer kann Teilausschnitte vergrößern und verkleinern, er kann auch horizontal zu benachbarten Komponenten navigieren.

Da der Kontext der betrachteten Modellelemente dem des Originalmodells entspricht, können die Elemente in Lupensichten geändert und angepasst werden, ohne über die normale Modellierung hinausgehende Nebeneffekte befürchten zu müssen.

Projektionssichten

Eine Projektionssicht zeigt Modellemenente an, die einem bestimmten Kriterium genügen. Dieses Kriterium ist vom Nutzer wählbar und bezieht sich auf eine prüfbare Eigenschaft eines Modellelements. So können beispielsweise alle Komponenten eines bestimmten Verantwortlichen angezeigt werden. Ein Kriterium kann auch die Partitionierung sein, so dass alle Funktionen angezeigt werden können, die auf ein bestimmtes Steuergerät partitioniert wurden.

Im Gegensatz zur Lupensicht sind Projektionssichten nicht mehr in der Modellhierarchie angesiedelt. Die angezeigten Modellelemente können auf unterschiedlichen Hierarchieebenen liegen und über Hierarchiegrenzen miteinander verbunden sein. Die Verbindung zwischen den Elementen, so sie existiert, wird ohne die dazwischenliegenden Hierarchiegrenzen als durchgehend angezeigt.

Damit ist die Möglichkeit gegeben, Modelle aufgrund anderer Kriterien zu betrachten, als die der Hierarchisierung zugrunde liegen. Das dient zwei Zwecken: Erstens können so Modelle aus verschiedenen Blickwinkeln betrachtet und bewertet werden. Zweitens können Projektionssichten als Sekundärhierarchien betrachtet werden, die orthogonal zur Primärhierarchie des Originalmodells liegen. Diese Sekundärhierarchien können attributiert oder mit Partitionierungsinformationen angereichert werden. Insbesondere für verteilte Funktionen kann die Verteilung virtuell aufgehoben werden.

Wichtig beim Umgang mit Projektionssichten ist die fehlende Kontextinformation. Damit wirken sich Änderungen am Modell unter Umständen anders aus als erwartet, weil viele Modellelemente ausgeblendet wurden.

Ergo

Zusammenfassend bieten Sichten die Möglichkeit, Modelle aus verschiedenen Blickwinkeln zu betrachten, die unter Umständen orthogonal zur originalen Mo-

dellierung stehen. Damit sind zusätzliche Modellierungsmöglichkeiten gegeben, die bei unbedachter Anwendung leicht zu fehlerhaften Modellen führen können, bei korrekter Anwendung aber große zusätzliche Möglichkeiten der Modellierung bieten.

Die vorgestellten Sichtenkonzepte sind unabhängig von den Architekturmodellen, das Sichtenmodell selbst steht orthogonal zu den Architekturmodellen. Damit werden Sichten hauptsächlich zu einer Werkzeug- denn einer Modellierungsfrage.

16 Versionierung

Die Versionierung gehört zu den prozessbegleitenden Methodiken, die im Verlauf des Entwicklungsprozesses auf alle Teilmodelle angewendet werden können. MOSES-Modelle selbst bieten kein Versionierungskonzept an, sondern stützen sich auf vorhandene, etablierte Prozesse. Im Rahmen der Modellierung treten manchmal Missverständnisse auf, was Versionen sind, was Varianten und wo die Unterschiede zwischen beiden liegen. Des Weiteren ist die Frage zu klären, ob Produktmodelle mehrere Versionen eines Modellelements enthalten dürfen oder nicht.

Diese Fragen wurden in MOSES 4-2 [Fra05b] kurz behandelt, die Ergebnisse werden hier ebenso kurz vorgestellt. Wie eingangs erwähnt, wird Versionierung parallel zur Modellentwicklung betrieben. Welche Modellteile versioniert werden, ist dem Nutzer der Modelle überlassen. Diese Entscheidung richtet sich nach dem Anwendungsgebiet und der Erfahrung der Nutzer. Meist werden Modellteile wie Funktionen oder Steuergeräte versioniert, im Domänenmodell Teilmodelle oder Bausteine.

Die Versionierung muss auf Ebene der Architekturmodelle erfolgen und sollte auch Modellebenen nicht überschreiten. So sollten logische und technische Architektur getrennt voneinander versioniert werden, da sie aus Versionierungssicht nichts miteinander zu tun haben. Gleiches gilt für Produkt-, Produktlinien- und Domänenmodelle. Diese Modelle sind unabhängig voneinander, was die Versionen ihrer Modelle betrifft: sie können unabhängig voneinander verändert werden, können also auch unabhängig versioniert werden.

Das Problem der Unterscheidung von Versionen und Varianten tritt hauptsächlich in Bezug auf die Verwendung von Teilmodellen des Domänenmodells auf. Dieses Problem und seine Lösung kann am Beispiel von Abbildung 51 gut dargestellt werden: Ein Teilmodell *TM 1* in einem Domänenmodell habe die Versionsnummer V1. Es wird in einem Produktmodell in dieser Version verwendet. Jetzt soll das Teilmodell weiterentwickelt werden.

Dazu wird es in einem Projekt verwendet und dort modifiziert. Die Modifikation kann über mehrere Versionen gehen, in unserem Beispiel ist das Teilmodell in Version V14 des Projekts fertig. Dann werden die Änderungen in das Domänenmodell übernommen, hier ist die neue Version die Version V2 des Teilmodells. Im Produktmodell wird nun Version V2 eingesetzt werden müssen, da Version V1 ersetzt wurde.

Abbildung 51 Versionen.

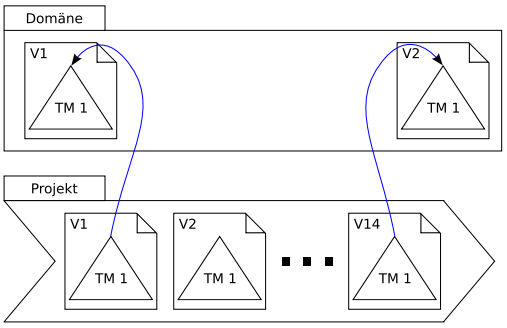
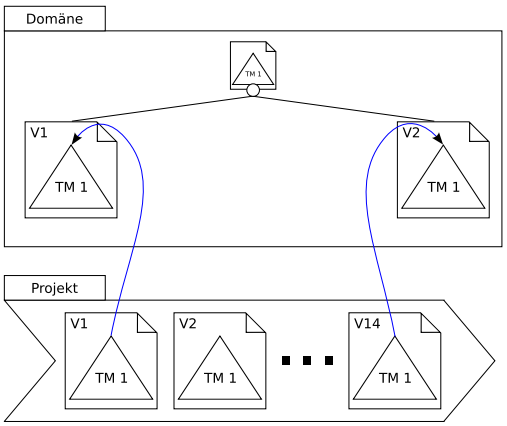


Abbildung 52 Versionen und Varianz.



Sind beide Versionen (V1 und V2) aber noch gültig (beispielsweise Prozessoren, von denen noch Restbestände eines alten Typs vorhanden sind), so kann ein Varianzpunkt im Domänenmodell eingeführt werden, unterhalb dessen die Versionen als Varianten modelliert werden. Dann kann im Produktmodell frei zwischen Version V1 und Version V2 gewählt werden (Abbildung 52). Die Einführung der Variante ist wohlgemerkt ein expliziter Modellierungsschritt, normalerweise überschreiben neue Versionen alte komplett. Damit sind die Möglichkeiten, Versionen und Varianten zu mischen, auf ein überschaubares und modellierbares Maß beschränkt worden.

17 Prozess

Der MOSES-Prozess gibt an, welche Schritte in welcher Reihenfolge auszuführen sind, um zu MOSES-Modellen zu kommen. Ein solcher Prozess wurde bisher nicht explizit definiert. Für Teilprobleme, wie z. B. die Aufarbeitung von Wissen für Domänenmodelle, liegen Teilprozessbeschreibungen vor, ein Gesamtprozessmodell fehlt. Das liegt daran, dass der Fokus im MOSES-Projekt auf den Architekturmodellen lag und der Prozess nicht als explizites Ziel definiert wurde.

Im Lauf des Projekts haben sich jedoch Vorgehensweisen und Prozessaspekte herausgeschält, die es ermöglichen, einen MOSES-Prozess zu skizzieren, wenngleich nicht formal zu definieren. Dabei wird ein artefaktgetriebener Prozess beschrieben, dessen Aktivitäten dazu dienen, bestimmte Artefakte, nämlich die Architekturmodelle herzustellen. Damit ist es möglich, den skizzierten Prozess in bestehende Prozessmodelle, seien es Produktentstehungsprozesse oder Vorgehensmodelle, einzubetten.

Die Erstellung von MOSES-Modellen wird in drei unterschiedlichen Stufen betrachtet, die im Folgenden vorgestellt werden:

1 MOSES-Prozess
2 Berücksichtigung von Varianz
3 Aufarbeitung von Domänenwissen

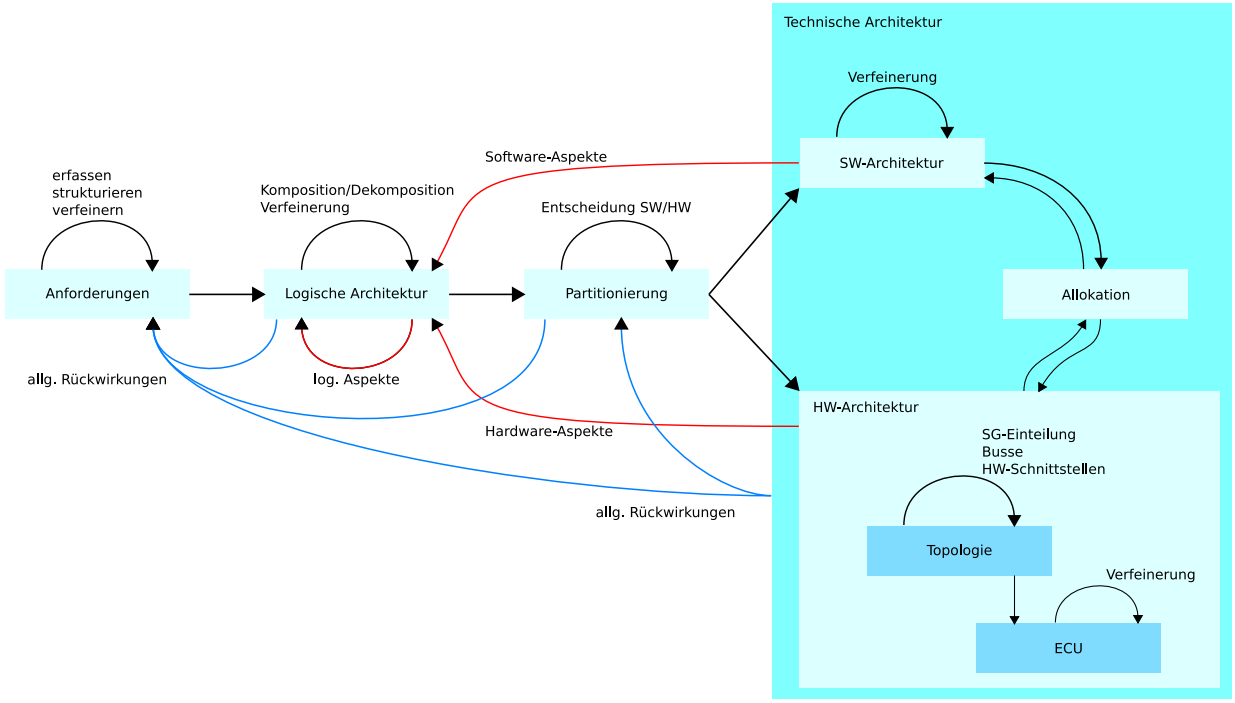
17.1 Der MOSES-Prozess

Der MOSES-Prozess beschreibt die Erstellung der MOSES-Modelle ohne die Berücksichtigung von Varianz oder Domänenaspekten. Er beschreibt also den einfachsten Weg zu MOSES-Produktmodellen. Dabei wird iteratives Vorgehen unterstützt.

Im Folgenden wird der Prozess anhand von Abbildung 53 sequentiell beschrieben. In der Praxis kann bei entsprechendem Vorwissen auch an anderer Stelle begonnen werden.

Eine Neuentwicklung beginnt mit der Modellierung von Anforderungen. Sie werden mit den Methoden des Anforderungsmanagements erfasst, strukturiert, mo-

Abbildung 53 Skizze des MOSES-Prozesses.



delliert und analysiert. Dabei können die Anforderungen verfeinert und in sich verändert werden.

Sind die Anforderungen fertig modelliert, wird aus den Anforderungen heraus die logische Architektur entwickelt. Dafür werden zunächst die Hauptfunktionen mit Hilfe der Kundenanforderungen festgelegt. Diese Funktionen werden verfeinert und soweit modelliert, dass die Blätter des Funktionsbaums atomar sind. Eine atomare Funktion zeichnet sich dabei vor allem dadurch aus, dass sie vollständig auf eine Softwarekomponente oder ein Steuergerät partitionierbar ist. Das Ergebnis der logischen Modellierung ist eine partitionierbare logische Architektur, ein Funktionsnetzwerk. Sollten sich in diesem Schritt Änderungen an den Anforderungen beispielsweise durch Verfeinerungen von Funktionen ergeben, so sind diese im Anforderungsmodell nachzutragen.

Im Rahmen einer aspektorientierten Modellierung sind gleichzeitig die Aspekte zu modellieren, die auf logischer Ebene berücksichtigt werden müssen. Das können z. B. rechtliche Aspekte oder Sicherheitsaspekte sein. Diese Verfeinerung wird im Modell explizit im Sinne einer aspektorientierten Modellierung als aspektgetriebene Verfeinerung gekennzeichnet.

Nach Fertigstellung der logischen Architektur kann diese partitioniert, das heißt in Software und Hardware aufgeteilt werden. Damit kann die technische Architektur, bestehend aus Softwarearchitektur und Hardwarearchitektur, modelliert werden. Auf Softwareseite werden die Softwarekomponenten modelliert, die später die Funktionen realisieren sollen. Auf Hardwareseite wird zunächst ein Topologiemodell erstellt, das die Einteilung der Steuergeräte und ihre Verbindung modelliert. Dieses Topologiemodell wird genutzt, um die Steuergerätearchitektur aufzubauen und zu modellieren.

Die Modellierung der technischen Architektur hat Rückwirkungen auf die vorhergehenden Architekturen: Die Partitionierung ist von Verfeinerungsschritten betroffen und muss eventuell analog verfeinert werden. Die Rückwirkung auf die logische Architektur geschieht über Aspekte, die erst modelliert werden können, wenn die technische Architektur vorhanden ist. Das sind zum Beispiel Überwachungsaspekte, die erst modelliert werden können, wenn klar ist, welche Hardware überwacht werden soll. Diese Aspekte werden wiederum mit den Mitteln einer aspektorientierten Modellierung eingebunden, um die Unabhängigkeit der ursprünglichen logischen Architektur zu wahren.

Sowohl Partitionierung als auch die Modellierung der technischen Architektur können Einfluss auf die Anforderungen haben, z. B. wenn zu testende oder zu überprüfende Anforderungen eingefügt werden müssen.

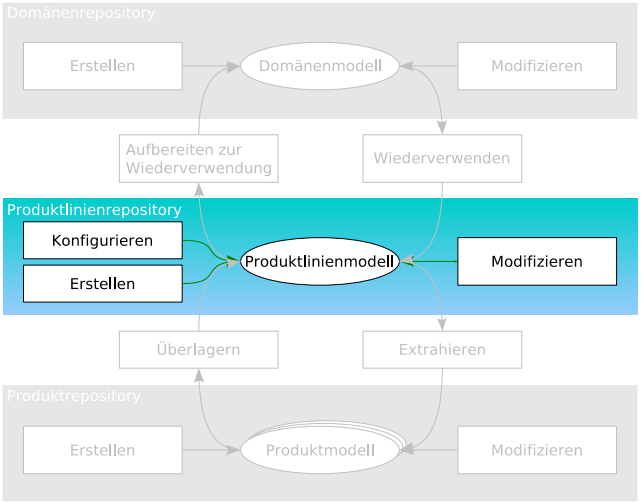
Zu guter Letzt wird nach der Fertigstellung der technischen Architektur die Allokation, das heißt die Verteilung der Software auf die Hardware, modelliert.

Der beschriebene Prozess kann in bestehende Prozesse eingebettet werden, da nur angegeben wurde, welche Artefakte in welcher Reihenfolge zu erstellen sind. Ein Tailoring z. B. des V-Modells im Hinblick auf diese Artefakte ist ohne Probleme möglich. Diese enge Einbettung ist nicht zuletzt der Tatsache geschuldet, dass das gesamte MOSES-Projekt praxisnah aufgezogen wurde und sich an bestehenden Prozessen orientierte.

17.2 Berücksichtigung von Varianz

Die Berücksichtigung von Varianz bedeutet die Erstellung von Produktlinienmodellen. Die Beziehung zwischen Produktmodellen und Produktlinienmodellen ist bereits in Abschnitt 14.2 erläutert worden: Produktmodelle werden überlagert und aus dieser Überlagerung Varianzpunkte definiert. Die Aktivitäten wurden ebenfalls bereits vorgestellt, sie sind in Abbildung 54 noch einmal dargestellt.

Abbildung 54 Aktivitäten der Produktlinienmodelle.

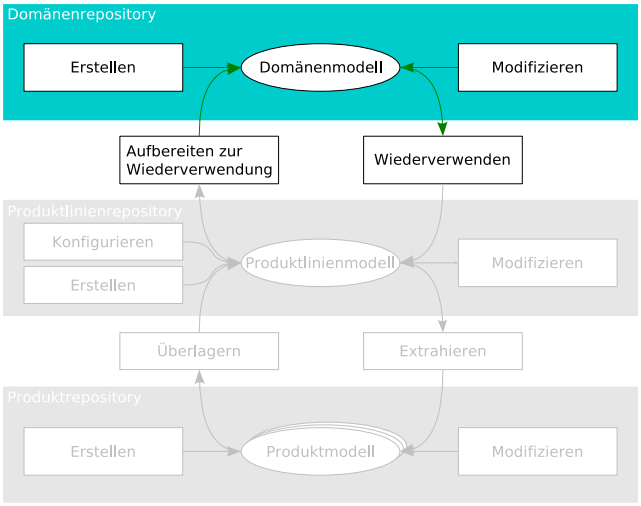


Im MOSES-Prozess kann Varianz für jedes Architekturmodell definiert werden. Dafür muss für jedes Modell festgelegt werden, an welcher Stelle ein Varianzpunkt festgelegt wird. Dann wird die Art der Varianz festgelegt: Optionalität, Auswahlvarianz, Attributvarianz, etc. Danach werden die Varianten beschrieben und annotiert. Die Annotation beschränkt sich bei MOSES auf die Auswahlinformationen, Auflösungszeitpunkte werden nicht modelliert, können aber als Anforderungen beschrieben werden.

17.3 Aufarbeitung von Domänenwissen

Die Aufarbeitung von Domänenwissen wird nicht als Erweiterung des MOSES-Prozesses modelliert. Sie findet in einem eigenen Prozess statt, der parallel zum MOSES-Prozess läuft und vom Domänenverantwortlichen gesteuert wird. Der Prozess wurde bereits in Abschnitt 14.3 beschrieben und wird daher an dieser Stelle nicht wiederholt. Die Abbildung 55 zeigt noch einmal die nötigen Aktivitäten.

Abbildung 55 Aktivitäten der Domänenmodelle.



18 Umsetzung in die Praxis

Die Umsetzung von MOSES in die Praxis bedeutet, den Produktentstehungsprozess mit MOSES-Modellen zu gestalten. Das bedeutet, sowohl die angefertigten Modelle (Artefakte) als auch die Prozesse müssen entsprechend umgestaltet werden. Das ergibt bei vollständiger Umsetzung der MOSES-Methodik viele Änderungen, die in den laufenden Prozess eingebracht werden müssen. Daher wird in diesem Kapitel vorgestellt, welche Veränderungen in welchem Ausmaß durchgesetzt werden müssen, um MOSES einzusetzen. Dabei wird auch vorgestellt, wie MOSES schrittweise umgesetzt werden kann.

18.1 Rollen für MOSES

In MOSES werden die bisherigen Rollen des Entwicklungsprozesses erweitert und neue Rollen definiert. Tabelle 16 zeigt die Übersicht über die von MOSES benötigten Rollen.

Tabelle 16 Rollen der MOSES-Methodik.

Rolle	Beschreibung
Anforderungsverantwortlicher	Der Anforderungsverantwortliche ist dafür zuständig, die Anforderungsmodellierung zu planen und zu überwachen.
Anforderungsersteller	Der Anforderungsersteller führt die Anforderungsmodellierung durch, dies umfasst insbesondere Erstellung, Erfassung und Strukturierung von Anforderungen. Die Ergebnisse des Anforderungserstellers werden dem Anforderungsverantwortlichen geliefert.
Systemintegrator	Der Systemintegrator ist für die Planung und Überwachung des Systementwurfs zuständig. Dazu gehört neben den Entwurfsprozessen auch die Integration der fertigen Subsysteme in das Gesamtsystem. Zunächst ist der Systemintegrator für die Erstellung der logischen und technischen Architektur zuständig. Dazu erarbeitet er mit dem Funktionsnetzverantwortlichen, dem Softwareverantwortlichen und dem Steuergeräteverantwortlichen jeweils deren Architekturen. Dann übergibt er die Architekturen an die Verantwortlichen und integriert deren Ergebnisse in das Gesamtsystem.
Funktionsnetzverantwortlicher	Der Funktionsnetzverantwortliche ist für die Planung und Überwachung der Erstellung der logischen Architektur zuständig. Er überwacht die Aufteilung der logischen Architektur in Teilfunktionen und integriert die von den Funktionsverantwortlichen gelieferten Funktionen in das Funktionsnetz.

Rolle	Beschreibung
Funktionsentwickler	Der Funktionsentwickler ist für den Entwurf einer Funktion zuständig. Er bekommt vom Funktionsnetzverantwortlichen die Schnittstellen seiner Funktion geliefert, modelliert die Funktion und liefert sie zurück.
Softwareverantwortlicher	Der Softwareverantwortliche überwacht und leitet die Erstellung der Software-Architektur. Er bekommt vom Systemintegrator die zu implementierenden Funktionen und ist für deren Umsetzung zuständig. Dabei obliegt es ihm, die Einteilung der Softwarekomponenten vorzunehmen.
Softwareentwickler	Der Softwareentwickler implementiert eine Softwarekomponente oder Teile davon.
Steuergeräteverantwortlicher	Der Steuergeräteverantwortliche überwacht und leitet die Erstellung der Hardwarearchitektur. Dabei verfeinert er eine gegebene oder selbst erstellte Hardwaretopologie und teilt diese in Steuergeräte auf, die dem Steuergeräteentwickler in Auftrag gegeben werden. Die fertigen Geräte werden in die Hardwarearchitektur integriert.
Steuergeräteentwickler	Der Steuergeräteentwickler entwirft und baut das ihm zugeteilte Steuergerät.
Produktlinienverantwortlicher	Der Produktlinienverantwortliche ist für die Pflege des Produktlinienmodells zuständig. Da üblicherweise das Produktlinienmodell häufig auch durch Entwickler geändert wird, hat der Produktlinienverantwortliche eher organisatorische Aufgaben: er muss einen Prozess zur Pflege und Konsistenzerhaltung des Produktlinienmodells definieren und dessen Einhaltung überwachen.
Domänenverantwortlicher	Der Domänenverantwortliche ist für die Pflege des Domänenmodells verantwortlich. Wie der Produktlinienverantwortliche hat er Prozesse zu definieren und deren Einhaltung zu überwachen, so dass das Domänenmodell konsistent bleibt.

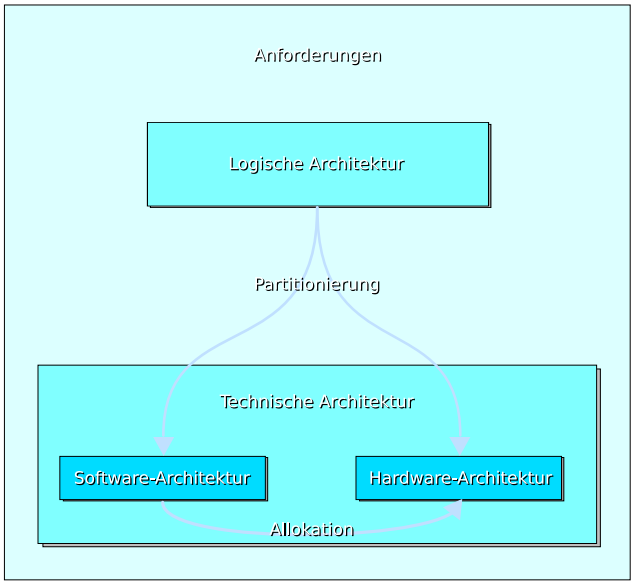
Die Übersicht zeigt zunächst alle Rollen, die für die MOSES-Methode wichtig sind. In der konkreten Umsetzung können mehrere Rollen von der gleichen Person ausgeführt werden oder Rollen können von Organisationseinheiten übernommen werden. Wichtig ist in diesem Zusammenhang eine eindeutige Festlegung der die Rollen ausführenden Personen.

Welche Rollen im Prozess umgesetzt werden müssen, richtet sich nach den Modellen, die von MOSES übernommen werden. Wenn z. B. zunächst nur die Anforderungsmodellierung etabliert werden soll, sind lediglich die beiden zugehörigen Rollen Anforderungsverantwortlicher und Anforderungsersteller in den Prozess zu integrieren.

18.2 Einführung der MOSES-Modelle

An dieser Stelle werden die MOSES-Modelle noch einmal unter dem Gesichtspunkt der Einführung in den Entwicklungsprozess vorgestellt. Alle MOSES-Modelle sind in der bereits bekannten Abbildung 56 zu sehen.

Abbildung 56 Das MOSES-Gesamtmodell.



Das Anforderungsmodell ist die Grundlage für die nachfolgenden Modelle. Die Anforderungsmodellierung verbessert alle Bereiche der Prozesskette, da Anforderungen für alle Bereiche erhoben werden können. Dieser Gedanke hat sich mittlerweile auch bei vielen Herstellern durchgesetzt, so dass die Notwendigkeit der Anforderungsanalyse nicht mehr bezweifelt wird. Ob Anforderungen nach MOSES erhoben werden oder nach einem anderen Schema, ist für den Nutzen der Anforderungen nicht entscheidend. Das heißt, von den MOSES-Aktivitäten sollte die Anforderungsmodellierung als erstes in den Prozess integriert werden.

Das weitere Vorgehen hängt stark von den vorhandenen Modellen und Prozessen ab. Wenn schon ein Funktionsbegriff als logische Abstraktion der Funktionalität etabliert ist, sollte die Funktionsnetzmodellierung eingeführt werden. Diese Modellierung fehlt häufig und bietet bereits nach kurzer Einarbeitungszeit großen Nutzen, da das Verständnis des Gesamtsystems rapide zunimmt.

Ist der Funktionsbegriff noch nicht etabliert, muss entschieden werden, ob die da-

für notwendigen Schulungen in Kauf genommen werden. Die Einführung einer logischen Architektur mit Funktionen ist in frühen Phasen schnell gewinnbringend. Die dafür notwendige Abstraktion muss allerdings erlernt werden.

Unabhängig von der logischen Architektur sollten Modelle für die technische Architektur eingeführt werden. Dieser Schritt ist oft vom Verständnis her unproblematisch, da Soft- und Hardwaremodelle oft bekannt sind und eingesetzt werden. Lediglich die abteilungsübergreifende Etablierung einer einzigen Methode stellt oft ein Problem dar. Die Motivation der Einführung ist aber erheblich leichter als bei der logischen Architektur, da oft auf bekannte Modelle verwiesen werden kann. Mit der Einführung von AUTOSAR werden in Zukunft Modelle der technischen Architektur zwingend eingesetzt.

Die Partitionierung im Sinne der Verteilung der logischen auf die technische Architektur kann erst eingeführt werden, wenn logische und technische Architekturmodelle etabliert sind. Die dadurch erzielte Verfolgbarkeit von Änderungen und Entscheidungen führt schnell zur Akzeptanz der neuen Modelle.

Damit ist seitens der Modelle eine stufenweise Einführung der MOSES-Modelle vorstellbar und zweckmäßig. Es sollte mit denjenigen Modellen begonnen werden, die am vertrautesten sind oder die den schnellsten Nutzen versprechen.

Parallel zu der reinen Einführung der Modelle können der Produktlinienansatz sowie der Domänenansatz verwirklicht werden. Dabei bietet es sich an, mit dem Produktlinienansatz zu beginnen und diesem früh in die Praxis zu überführen. Er erfordert wenig Umdenken seitens der Nutzer, im Gegenteil: Die Möglichkeit, Varianz modellieren zu können, wird oft dankbar angenommen.

Anders sieht es mit Domänenmodellen aus. Die Funktion von Domänenmodellen als Bausteinreservoir kann früh eingeführt werden, dabei darf die Rolle des Domänenverwalters aber nicht unterschätzt werden. Wird das Domänenkonzept zu früh und unkoordiniert eingeführt, verkommt das Domänenrepository zu einer Ansammlung aller Bausteine, ohne die besondere wissensextrahierende Rolle eines Domänenmodells ausfüllen zu können. Dabei ist gerade der langfristige Ansammlungsprozess von Domänenwissen die Grundlage eines funktionierenden Domänenrepositories.

18.3 Organisatorische Einführung

Die organisatorische Einführung von MOSES in den Produktionsprozess muss u. a. folgende Aktivitäten berücksichtigen:

- Einführung der Modelle
- Einführung der Rollen
- Anpassung der Prozesse
- Einführung neuer Werkzeuge

Die Einführung der Modelle wurde im vorigen Abschnitt bereits besprochen, es bleibt noch zu ergänzen, dass die Modelle auch abgelegt und bearbeitet werden müssen. Dafür werden die Rollen aus Abschnitt 18.1 in den Entwicklungsprozess integriert. Der Prozess muss dahingehend angepasst werden, dass die Rollen nicht nur definiert werden, sondern ihr Wirkungsbereich und eventuell zusätzlich anfallende Aktivitäten berücksichtigt werden.

Gleichzeitig sind Werkzeuge zu etablieren, die mit den neuen Modellen umgehen können, das heißt, sie erstellen und ändern können. Es ist zweckmäßig, die Modelle zentral zu speichern, dafür muss eine Ablage geschaffen werden. Der Zugriff auf diese zentrale Ablage muss mit den Werkzeugen möglich sein, optimalerweise werden dabei die Rollen in ihren spezifischen Anforderungen unterstützt. Ein Zugriffs- und Änderungsmanagement über die Modelle muss etabliert werden.

Nicht zuletzt sind die sozialen Auswirkungen zu berücksichtigen: Mitarbeiter müssen geschult werden, sowohl in der Benutzung der neuen Modelle als auch in der Benutzung der Werkzeuge. Die Rollen sind zu erklären, zu etablieren und zu leben.

An dieser Stelle wurden nur die wichtigsten Punkte aufgezählt, die bei der Einführung der MOSES-Modelle zu berücksichtigen sind. Die Aufzählung ist nicht vollständig, ein Hersteller weiß selbst genau, wie schwer es ist, Prozesse zu ändern oder neue Methoden zu etablieren. Der sofort sichtbare Nutzen der MOSES-Methodik rangiert von kurzfristig bei der Modellierung der Funktionsnetzwerke oder Anforderungen bis zu langfristig wie bei Domänenmodellen. Daher sind die Auswirkungen zu analysieren und die Modelle Schritt für Schritt einzuführen.

Literatur

[bmw] *BMW AG*. http://www.bmw.de/

[Fra02] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *MOSES 1: Anforderungsmodellierung und Anforderungsmanagement im Domain Engineering. Eine Methodik für die BMW Group*. November 2002. – Projektabschlussbericht

[Fra03a] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *MOSES 2: Modellierung von hierarchischen, vernetzten Funktionen. Eine Methodik für die BMW Group*. Juli 2003. – Projektabschlussbericht

[Fra03b] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *MOSES 3.1: Architekturmodellierung, Teil 1, Metamodell für technische Architekturmodelle und Partitionierung von logischen Funktionen*. September 2003. – Projektabschlussbericht

[Fra04a] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *MOSES 3.2: Architekturmodellierung, Teil 2, Erweiterte Modellierung und Bewertung von Partitionierungen*. Januar 2004. – Projektabschlussbericht

[Fra04b] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *MOSES 4.1: Validierung und Umsetzung der modellbasierten Entwicklungsmethodik MOSES, Teil 1: Konfiguration, Produktlinien- und Domänenmodelle für Funktionsnetzwerke*. September 2004. – Projektabschlussbericht

[Fra05a] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *Das MOSES-Gesamtmetamodell*. April 2005. – Projektabschlussbericht

[Fra05b] Fraunhofer-Institut für Software- und Systemtechnik, Abteilung Verlässliche Technische Systeme: *MOSES 4.2: Validierung und Umsetzung der modellbasierten Entwicklungsmethodik MOSES, Teil 2: Produktlinien- und Domänenmodelle technischer Architekturen*. April 2005. – Projektabschlussbericht