Modellierung variantenreicher Funktionsnetze im Automotive Software Engineering

Cem Mengi Antonio Navarro Perez Christian Fuß

Lehrstuhl für Informatik 3 - Softwaretechnik
RWTH Aachen, Ahornstr. 55, 52074 Aachen
{mengi | antonio | cfuss}@i3.informatik.rwth-aachen.de

Abstract: *Funktionsnetze* gehören während des Systementwurfs im modellgetriebenen Top-Down-Entwicklungsprozess von Fahrzeugsoftware zu den ersten Artefakten; sie werden aus den Spezifikationen der Anforderungen abgeleitet und sind die Grundlage der Architekturspezifikation. In der Praxis haben sich Funktionsnetze aber noch nicht konsequent durchgesetzt. Dort arbeiten Fahrzeughersteller vorrangig mit umfangreichen und technisch-detaillierten Spezifikationen wie Kommunikationsmatrizen. Angesichts des stetig steigenden Grades an Variabilität und Komplexität in heutigen Fahrzeugfamilien ist dieses Vorgehen zunehmend unzweckmäßig. Wir schlagen das Konzept der logischen Funktionsnetz-Familien vor, um diesem Problem zu begegnen. Dieses Konzept besteht aus 1) einer Notation für Funktionsnetze, 2) einer Notation für Variabilität in Form adaptierter Feature-Modelle und 3) zugehörigen Modellierungsmethodiken.

1 Einleitung

Die Automobilindustrie bietet derzeit ihren Kunden beim Kauf eines Fahrzeugs individuelle Ausstattungsmöglichkeiten, wie beispielsweise Parkassistenten, Regensensoren und Komfort-Schließsysteme. Die Vielfalt an softwarebasierten Sonderausstattungen und ihre Kombinationen ermöglichen es, eine große Anzahl an Fahrzeugvarianten zu konfigurieren.

Für Fahrzeughersteller bedeutet dies, dass im Verlauf des E/E-Entwicklungsprozesses *Varianten* bzw. *Variationspunkte*, die unterschiedliche Varianten hervorrufen, in den Entwicklungsartefakten explizit und geeignet erfasst werden müssen [PBvdL05, CN01].

Die Existenz von Varianten erstreckt sich dabei über den gesamten E/E-Entwicklungsprozess. Varianten können in der Anforderungsspezifikation, in der Systemspezifikation, im Architekturentwurf, im Code, aber auch in der Test- und Integrationsphase vorhanden sein. Darüber hinaus entstehen Varianten auch im Laufe der Produktion und in der Betriebs- und Wartungsphase, so dass im gesamten Produktlebenszyklus, der bei einem Fahrzeug ca. 20-25 Jahre andauert, sehr viele und unterschiedliche Ausprägungen von Varianten entstehen können.

Ein Artefakt der Systemspezifikation ist das so genannte *Funktionsnetz* [WR06], das zu den ersten Artefakten eines *Top-Down*-Spezifizierungsprozesses gehört. Es modelliert auf

[MPF09] C. Mengi, A. Navarro Perez, C. Fuß
Modellierung variantenreicher Funktionsnetze im Automotive Software Engineering.
7. Workshop Automotive Software Engineering (ASE 09).
In: Proceedings INFORMATIK 2009. GI Edition - Lecture Notes in Informatics (LNI), Vol. P154, 2009.
www.se-rwth.de/publications

einer *logischen* Ebene die einzelnen Funktionen des Systems, die aus den Anforderungen abgeleitet werden, sowie ihre Kollaborationsstrukturen und gegenseitigen Abhängigkeiten. Funktionsnetze bilden damit eine Brücke zwischen der Anforderungsspezifikation und der technischen Systemspezifikation. Mit ihrer Hilfe können Entwurfsentscheidungen für den Grobentwurf diskutiert und getroffen werden. Sie dienen den Entwicklern als gemeinsames *Verständismodell* und als *Kommunikationsgrundlage*, auch über Entwicklergruppen hinweg. Abbildung 1 zeigt ein schematisches Beispiel für ein Funktionsnetz.

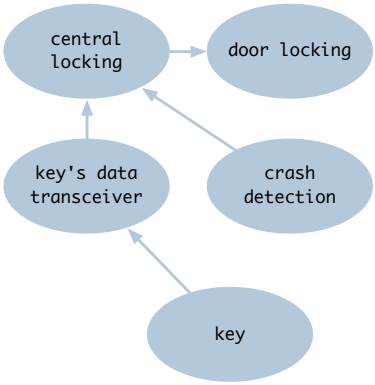

Abbildung 1: Ein schematisches Funktionsnetz

Dafür sollten bereits bei der Modellierung der Funktionsnetze geeignete *Modellierungs-Richtlinien* aufgestellt und beachtet werden, um ein in allen Köpfen konsistentes Verständnis und Vokabular zu erreichen. Des Weiteren müssen Variationspunkte angemessen erfasst werden, um die in der Anforderungsspezifikation formulierten Varianten einzubringen und sie in Hinblick auf die Systemspezifikation behandeln zu können.

In der Praxis hat sich die Top-Down-Spezifikation mit Funktionsnetzen noch nicht flächendeckend durchgesetzt. Dort beherrscht das herkömmliche *Bottom-Up*-Verfahren die Vorstellung des Spezifikationsprozesses, bei dem das elektronische System des Fahrzeugs durch Komposition einzelner konzeptionell und technisch geschlossener Komponenten entsteht. Fahrzeughersteller benutzen dort so genannte Kommunikationsmatrizen an Stelle von Funktionsnetzen. Die Fahrzeugfunktionalität ist dort auf einer *technischen* Ebene mit einer vorgreifenden Vorstellung der Partitionierung auf Steuergeräte detailliert und ausgearbeitet festgehalten. Kollaborationsstrukturen und Abhängigkeiten sind nicht explizit modelliert, da die Kommunikation des Systems bereits hier als Kommunikation über eine technische Bus-Topologie verstanden wird. Entwurfsentscheidungen sind vorrangig durch implizites technisches Wissen motiviert, das von technischen Spezifikationen bestehender Fahrzeugmodelle geerbt wird. Abbildung 2 zeigt einen (nachgestellten und sehr kleinen) Ausschnitt aus einer solchen Kommunikationsmatrix in ihrer tabellarischen Darstellung.

Varianten werden in Kommunikationsmatrizen nicht explizit festgehalten. Stattdessen werden in der Praxis vollständige Kommunikationsmatrizen erstellt, die alle möglichen Varianten enthalten. Wo dieses Vorgehen zu Konflikten führt, werden informelle Notizen hin-

Funktion	Nachricht	Signal	Träger	Richtung
Zentralverriegleung_Master	Zentralverriegelung_FV_Status	Zentralverriegelung_FV_Aktator_Stellung	ACS-BUS	Out
Zentralverriegleung_Master	Zentralverriegelung_FV_Status	Zentralverriegelung_FV_Aktator_Sperre	ACS-BUS	Out
Zentralverriegleung_Master	Zentralverriegelung_FV_Status	Zentralverriegelung_FV_Kontakt_Bedienung	ACS-BUS	Out
Zentralverriegleung_Master	Zentralverriegelung_FV_Status	Zentralverriegelung_FV_ID_Trnsmt	ACS-BUS	Out
MW_Antenna	MW_Antenna_Sig_CRS	ID_HRR_CS_Ref	HW	In
Kontakt_FV	Kontakt_FV	Kontakt_CLS_1	EL	In
Kontakt_FV	Kontakt_FV	Kontakt_CLS_1_gz	EL	Out
Kontakt_FV	Kontakt_FV	Kontakt_CLS_2	EL	In

Abbildung 2: Ein kleiner Ausschnitt einer Kommunikationsmatrix

zugefügt. Konkrete Ausprägungen des Systems werden erzeugt, indem nicht-gebundene Varianten gestrichen werden.

Kommunikationsmatrizen haben aufgrund ihrer technisch-detaillierten Natur den Vorteil, in kurzer Zeit zu System-Prototypen überführt werden zu können, mit denen erste Simulationen ausgeführt werden können. Dieser Vorteil wird aber mit sehr komplexen und unübersichtlichen Funktionsnetzen erkauft, die durch ihre Unverständlichkeit begleitende Dokumentation erfordern. Im Rahmen einer Kooperation mit einem Fahrzeughersteller haben wir Einblick in die Kommunikationsmatrizen eines seiner Fahrzeugmodelle erhalten. In Abbildung 3 sind die dort beschriebenen Funktionen und ihre kommunizierten Signale für eine einzelne Subsystemkomponente, für ein Subsystem und schließlich für vier Subsysteme graphisch dargestellt. Funktionen sind über Kanten mit Signalen verbunden, die sie über den Bus senden und empfangen.

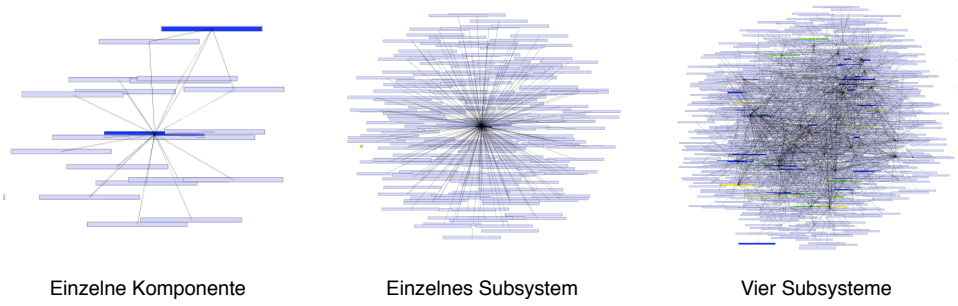

Abbildung 3: Graphische Darstellung von Funktionsnetzen für eine Subsystemkomponente, ein Subsystem und vier Subsysteme

Es ist sofort ersichtlich, dass die schiere Komplexität der Kommunikationsmatrix sie als Verständnismodell disqualifiziert. Sie kann in Folge dessen nicht als Kommunikationsgrundlage dienen. Auch als Unterstützung für Entwurfsentscheidungen ist sie untauglich, da ihre Komplexität und das Fehlen explizit modellierter Abhängigkeiten schnelle und unkomplizierte explorative Veränderungen nicht erlaubt. Stattdessen ist es praktisch unmöglich, die Auswirkungen von Änderungen an einer Funktion auf andere Funktionen nachzuvollziehen.

Zudem geht das in der Anforderungsphase gewonnene Wissen über Varianten praktisch

verloren. Variabilität als zentrales Konzept droht aus dem Bewusstsein der Entwickler gedrängt zu werden. Ihrer Bedeutung für den Spezifikationsprozess wird damit keine Rechnung getragen. Stattdessen werden Varianten als alternative Komponenten verstanden, die in Anlehnung an das Bottom-Up-Vorgehen relativ entkoppelt von ihrer Systemumgebung hinzugefügt oder entfernt werden können. Gegenwärtige Entwicklungen und Erkenntnisse zeigen aber, dass die Automobilindustrie in Zukunft eher von einem abstrakten Funktionsbegriff getrieben sein wird, bei dem zwischen Funktion und Partitionierung auf Steuergeräte unterschieden wird [AUT, CCG+07, PBKS07, Bro06]. Die zunehmende Komplexität, Vernetztheit und Verschmierung von Fahrzeugfunktionalität über verschiedene ursprünglich unabhängige Fahrzeugaspekte lässt eine einfache Abbildung auf Steuergeräte nicht mehr zu.

In diesem Artikel schlagen wir in Abschnitt 2 eine Notation für Funktionsnetze vor, mit der wir Kommunikationsmatrizen als Grundlage der Systemspezifikation ablösen wollen. Wir ergänzen diese Funktionsnetz-Sprache um eine Modellierungsmethodik, die Entwicklern den Übergang von Kommunikationsmatrizen zu Funktionsnetzen erleichtern und festgefahrene Bottom-Up-Denkweisen auflösen soll. Diese Methodik besteht aus Modellierungs-Richtlinien, die wir an bewährte Konzepte der Softwaretechnik anlehnen, und aus dem Vorschlag eines Antimuster-Kataloges. Um der Variantenproblematik zu begegnen, greifen wir in Abschnitt 3 auf das Konzept kardinalitäts-basierter Feature-Modelle [CHE04] [KCH+90] zurück und schlagen eine an Funktionsnetze angepasste Notation des zugrunde liegenden Variabilitätsverständisses vor. Das Ergebnis beider Vorschläge sind *Funktionsnetz-Familien*.

2 Funktionsnetze

Funktionsnetze sind Artefakte der logischen Systemarchitektur. Bei ihrer Modellierung gehen wir von einem abstrakten Funktionsbegriff aus. Funktionen sind hier keine Bauteile einer konkreten Realisierung, sondern abstrakte Konzepte. Sie abstrahieren von Realisierungsdetails wie der Partitionierung auf Hard- und Softwarekomponenten, der Anzahl ihrer Instanzen im realen System, Details ihrer Umgebung und Abläufen im System. Funktionsnetze modellieren diese Funktionen und die Kollaborationsstrukturen, in denen Funktionen durch gegenseitige Kommunikation (Signale) die Gesamtfunktionalität des Systems bilden.

2.1 Eine Notation für Funktionsnetze

Funktionen und Kollaboration zwischen Funktionen sind damit die Kernelemente einer Notation für Funktionsnetze. Unsere Notation besteht aus *Funktionen*, *Ports* und *Verbindungen*. Abbildung 4 zeigt ein beispielhaftes Funktionsnetz in dieser Notation.

Funktionen werden als Blöcke dargestellt, die einen aussagekräftigen Namen in natürlicher Sprache besitzen.

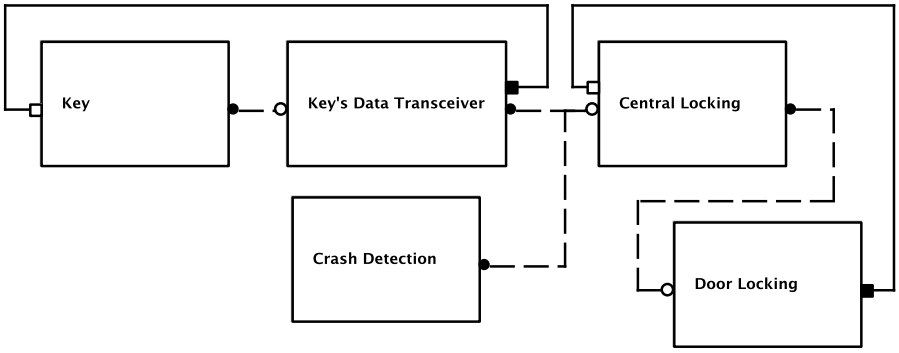

Abbildung 4: Unsere Notation für Funktionsnetze

Ports hängen Funktionen an und stellen den Kommunikationsbedarf einer Funktion dar. Schwarz ausgefüllte Ports modellieren ausgehende Kommunikation, weiß ausgefüllte Ports modellieren eingehende Kommunikation. Ports besitzen einen Signaltyp, der das Signal kennzeichnet, das über den Port kommuniziert werden kann. Ein Port kann jeweils nur einen Signaltyp kommunizieren. Zusammen bilden Ports das Interface ihrer Funktion.

Verbindungen verbinden Ports. Dabei gehen Verbindungen immer von einem ausgehenden Port zu einem eingehenden Port. Diese beiden Ports müssen den gleichen Signaltyp besitzen. Unter diesen Bedingungen können Ports über beliebig viele Verbindungen verbunden sein. Es gibt zwei Arten von Verbindungen, die jeweils ein anderes Kommunikationsparadigma darstellen. Gestrichelte Linien stehen für Steuerverbindungen, durchgezogene Linien für Datenverbindungen. Über Steuerverbindungen kann die sendende Funktion bei der empfangenden Funktion Verhalten auslösen. Über Datenverbindungen kann die sendende Funktion permanent Daten zur Verfügung stellen. Diese beiden Paradigmen beziehen sich auf die logische Semantik einer Kommunikationsbeziehung zwischen Funktionen. Sie sind nicht mit der technischen Umsetzung von Kommunikation zu verwechseln, wie sie z.B. AUTOSAR [AUT] als Client-Server- und Sender-Receiver-Kommunikation definiert.

Im pathologischen Beispiel in Abbildung 4 steht die Funktion `Key` für die Funktionalität, die man mit einem Autoschlüssel verbindet. Die Funktion `Key's Data Transceiver` steht für die Funktionalität, die die Kommunikation zwischen einem Autoschlüssel und dem Auto ermöglicht. Die Steuerverbindungen zwischen beiden modelliert hier die Anforderung des Schlüssels an das Fahrzeug, die Türen zu entriegeln. Die Datenverbindung zwischen beiden modelliert ein vom Fahrzeug ausgehendes Funksignal, das Schlüsseln mitteilt, dass sie in Reichweite ihres Fahrzeuges sind.

In der Literatur findet sich kein Standard für die Darstellung von Funktionsnetzen. Die meisten Variationen lehnen sich an Komponentendiagramme an und kennen ebenfalls Funktionen, Ports und Verbindungen. Wir haben unsere Notation daran angelehnt und auf das für ein flexibles Verständnismodell Notwendige reduziert.

2.2 Modellierungsmethodik für Funktionsnetze

Die Softwaretechnik kennt Entwurfsprinzipien für Software-Architekturen, die als allgemeine Richtlinien beim Entwurf dienen [LL05]. Wir wollen einige dieser Prinzipien auf die Modellierung von Funktionsnetzen anwenden. Im Kern handelt es sich dabei um *Abstraktion* und *Partitionierung*.

Abstraktion ist die fundamentale Methode zur Beherrschung und Reduktion von Komplexität. Durch die Anwendung von Abstraktion werden die Funktionen des Systems nahe an den in der Anforderungsanalyse gewonnen Anforderungen, anstatt nahe an der technischen Realisierung des Systems modelliert. Dazu partitionieren wir die Gesamtfunktionalität des Systems auf Grundlage der Semantik einzelner Funktionen, anstatt ihrer späteren Verteilung auf Steuergeräte. Die so ausgebildeten Funktionen sollen generalisiert und reduziert sein, also nur in einem sinnvollen Maße spezifisch sein. Redundanz soll dabei möglichst vermieden werden.

Partitionierung ist das Prinzip der Unterteilung des Funktionsnetzwerkes in relativ unabhängige Funktions-Teilmengen; also in *Partitionen*. Dadurch gliedern wir das Dokument, erhöhen die Übersicht und erleichtern das Verständnis. Zusätzlich erleichtern wir durch die Partitionierung des Systems die Definition von Varianten.

Das Ziel ist ein Entwurf, der durch seine Einfachheit möglichst verständlich und flexibel ist. Durch geringe Komplexität und geringe Redundanz können Entwurfsideen leichter diskutiert und skizziert werden, ohne Behinderung durch verfrüht angesprochene technische Details. Durch die logische Nähe zur ebenfalls abstrakten Anforderungsspezifikation sollen Inkonsistenzen zwischen Anforderungen und Entwurf möglichst früh ersichtlich werden.

Abbildung 5 zeigt, wie eine beispielhafte Menge von Funktionen, wie sie in einer Kommunikationsmatrix spezifiziert sind, durch Anwendung von Abstraktion verständlicher modelliert werden kann. Die Funktionen für das Schließen der Türen haben in diesem Beispiel für jede Fahrzeugtür die gleiche Semantik. Die Anzahl der Fahrzeugtür-Instanzen im konkreten Fahrzeug ist damit bereits ein technischer Aspekt, der von weitergehenden Parametern abhängen kann. Die vier Funktionen der Fahrzeugtüren können zu einer Funktion reduziert werden. Ebenso können die Steuerverbindungen zu einer Verbindung reduziert werden. Um die variable Anzahl der Instanzen zu modellieren, wird der Signaltyp mit der Anzahl der Türen parametrisiert. Die Entwurfsidee, dass die Zentralverrieglung *alle vorhanden Türen* entriegeln kann, ist so übersichtlicher und eindeutiger modelliert.

Abbildung 5 demonstriert indirekt auch unseren Ansatz zur Anwendung von Partitionierung. Die Funktion der Zentralverrieglung, taucht in verschiedenen Kontexten auf. Neben der Kernidee des Ent- und Verriegeln von Türen ist sie beispielsweise auch Teil des Sicherheitssystems, das Türen nach einem Unfall entriegeln soll, oder Teil der Funkkommunikation mit dem Autoschlüssel. Wenn wir die Menge aller Funktionen strukturell partitionieren, sodass jede Funktion eindeutig einer Partition der Gesamtmenge zugeordnet wird, werden wir dieser „Verschmiertheit“ von Funktionen auf verschiedene, heterogene und überlappende Kontexte nicht gerecht.

Wir schlagen daher das Konzept der *Sichten* vor. Eine Sicht ist eine Teilmenge von Funk-

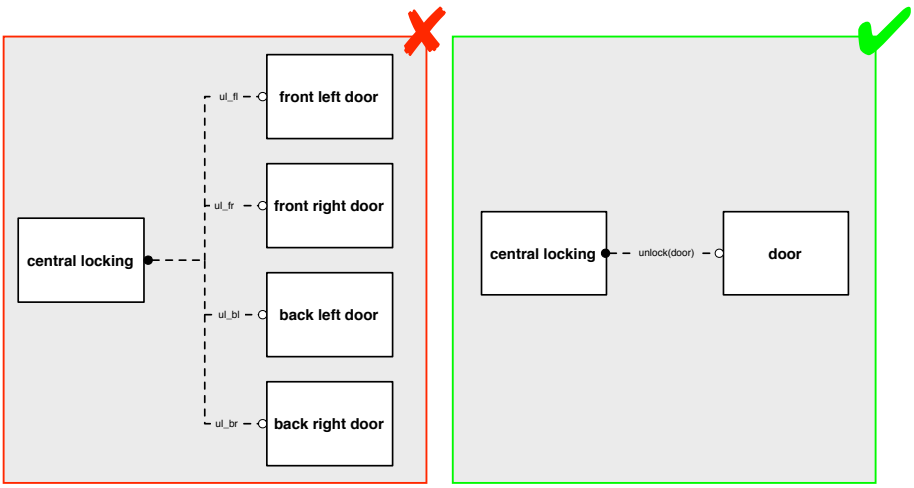

Abbildung 5: Anwendung von Abstraktion

tionen, die in einem für das Verständnis relevanten Zusammenhang stehen. In einer Sicht werden nur diejenigen Ports und Verbindungen dargestellt, die Funktionen innerhalb einer Sicht verbinden. Strukturell bleibt die Funktion aber weiterhin Teil des kompletten Funktionsnetzes. So gewinnen wir Übersicht, ohne unzutreffende strukturelle Gliederungen vornehmen zu müssen. Abbildung 5 ist selbst ein Beispiel für eine solche Sicht.

Partitionierung kann auch durch eine hierarchische Dekomposition von Funktionen in Teilfunktionen angewendet werden. Funktionen können dabei eine innere, aus Teilfunktionen bestehende Struktur – eine Art Sub-Funktionsnetz – besitzen. Diese Anwendung gewinnt mit der schrittweisen Verfeinerung des Grobentwurfes an Zweckmäßigkeit; in unserem Kontext des frühen Grobentwurfes, den wir in diesem Kontext behandeln, verzichten wir aber auf eine ausführliche Diskussion dieses Hilfsmittels.

Wir versprechen uns durch die Anwendung dieser Prinzipien eine verbesserte Verständlichkeit des Funktionsnetzes für Entwickler, eine gemeinsame Basis für die Kommunikation zwischen Entwicklern und ein flexibles und leicht zu erweiterndes Funktionsnetz, das mit seiner Umwelt und unterschiedlichen Anforderungen skalieren kann. Ein ausführlicheres Beispiel zur Verbesserung von Funktionsnetzen findet sich in [MA09].

2.3 Modellierungs-Richtlinien für Funktionsnetze

Die im vorherigen Abschnitt vorgestellten Modellierungsprinzipien sind vielfältig interpretierbar und umsetzbar. Für die Praxis benötigen wir deswegen Richtlinien, die Entwicklern konkrete Handlungsratschläge zu Modellierungsfragen geben. Als Notation für

Name	Redundanz bei Fahrzeugtüren
Symptom	Eine Funktion im Kontext von Fahrzeugtüren ist redundant vorhanden. Ihre redundanten Entitäten modellieren ein vielfaches Vorhandensein der Funktion für jede Fahrzeugtür. Ihre Entitäten unterscheiden sich nicht in ihrer allgemeinen Semantik, sondern nur in einem Bezeichner, der die Entität in der Menge alle Entitäten der Funktion kennzeichnet.
Lösungsmuster	Die Entitäten der Fahrzeugtüren werden zu einer Entität zusammengefasst. Gleiches geschieht mit allen Verbindungen und allen verbundenen Funktionen.

Abbildung 6: Ein Beispiel für ein Antimuster

Name	Gleichsetzung von Sensoren und Informationsquelle
Symptom	Das System benötigt Informationsquelle(n) für eine bestimmte Menge von Statusinformationen. Diese Informationsquelle(n) werden implizit mit Sensoren gleichgesetzt, die in der technischen Realisierung die Informationen ermitteln (Temperatursensoren, Kontakte, ...).
Lösungsmuster	Die Informationsquelle sollte anhand semantisch zusammenhängender Informationen und unabhängig von ihren Sensoren modelliert sein. Beispiel: Der Status der Türschlösser wird von einer einzigen Funktion ”Fahrzeugverriegelung” angeboten, statt von mehreren Funktionen, die auf physische Kontakte an jedem physischen Schloss abbilden.

Abbildung 7: Ein weiteres Beispiel für ein Antimuster

solche Richtlinien schlagen wir einen Katalog von *Antimustern* vor.

Antimuster sind in gewisser Weise die Antagonisten zu dem bekannten Konzept der Entwurfsmuster. Sie bestehen aus einem Symptom und einem Lösungsmuster. Symptome bezeichnen Strukturen und Eigenschaften in Funktionsnetzen, die möglicherweise auf eine schlechte Modellierung hinweisen. Lösungsmuster bieten Vorschläge an, wie diese Symptome beseitigt werden können. Zum einen zeigt dieser Katalog auf, welche Fehler man vermeiden sollte. Zum anderen dient er zur Unterstützung bei der Migration von Kommunikationsmatrizen zu Funktionsnetzen.

Ein Antimuster für Funktionsnetze besteht aus einem Namen, der Beschreibung des Symptoms und der Beschreibung des Lösungsmusters. Sie können dabei sowohl allgemeine Probleme, als auch sehr konkrete Probleme aus bestimmten Fahrzeugdomänen festhalten. Abbildung 6 und Abbildung 7 zeigen Beispiele für solche Antimuster.

Wir bevorzugen für unseren Kontext Antimuster gegenüber Entwurfsmustern. Entwurfsmuster können nur dann formuliert werden, wenn sie sich als eine Art „best practice“ in der Praxis bewährt haben. In unserem Fall existiert eine solche Praxis praktisch nicht, denn ein Top-Down-Entwicklungsprozess mit Funktionsnetzen hat sich bei Fahrzeugherstellern

noch nicht flächendeckend durchgesetzt. Entwurfsmuster bringen immer die Gefahr mit sich, dem Entwickler „Schauklappen“ gegenüber alternativen Entwurfsideen aufzusetzen. Diese Gefahr kann man nur für bewährte Entwurfsmuster rechtfertigen.

Antimuster haben eine andere psychologische Wirkung: Sie sind Warnungen für bestimmte Entwurfsprobleme *nachdem* man entworfen hat, geben aber keine konkrete Handlungsanweisung *bevor* man entwirft. Sie können zudem mit den Erfahrungen begründet werden, die durch Kommunikationsmatrizen und die Probleme mit ihrem Umgang gewonnen wurden. Wir schlagen sie deswegen als einen ersten Schritt auf dem Weg zu Modellierungs-Richtlinien vor. Entwurfsmuster können später als zweiter Schritt folgen.

3 Variabilität in logischen Funktionsnetzen

Im einleitenden Abschnitt haben wir dargelegt, wie Variabilität in Kommunikationsmatrizen durch die Bildung kompletter Systeme berücksichtigt wird. Durch die fehlende explizite Darstellung von Variabilität fehlt allerdings ein klares, einheitliches und konsistentes Verständnis von Variabilität, das mit Dokumenten anderer Phasen verglichen werden kann. Durch unsere in Abschnitt 2 vorgeschlagene Modellierungsmethodik mit Funktionsnetzen verlieren wir diese Variabilitäts-Information. Wir ergänzen Funktionsnetze darum um eine Notation für Variabilität, die wir an Feature-Modelle anlehnen. Als Ergebnis erhalten wir logische Funktionsnetz-Familien.

3.1 Funktionsnetze und Feature-Modelle

Für die Analyse der Anforderungen haben sich Feature-Modelle als orthogonales Variabilitätsmodell bewährt [KCH+90, PBvdL05]. Feature-Modelle sind nicht explizit auf die Anforderungsanalyse beschränkt; ihre Darstellung kann für andere Zwecke aber unzweckmäßig sein. Für Funktionsnetze wünschen wir uns die Möglichkeit, Varianten von Funktionen mitsamt ihrer Ports und Verbindungen übersichtlich festhalten zu können. Zudem wollen wir für Funktionen auf die Darstellung einer hierarchischen Dekomposition von Varianten, wie man sie für Features in Feature-Modellen findet, verzichten.

Wir schlagen deswegen die Notation der *funktionalen Feature Modelle* vor, das auf dem Variabilitätsverständnis von kardinalitäts-basierten Feature-Modellen basiert, aber für Funktionsnetze angepasst ist. Auf Modellebene übernehmen wir dabei die Semantik und Ausdrucksstärke von kardinalitäts-basierten Feature-Modellen. Abbildung 8 illustriert den Aufbau dieser Notation.

Variationspunkte und die ihnen zugehörigen Varianten werden in einer flachen Baumstruktur angeordnet. Die Wurzel enthält eine Referenz auf das Funktionsnetz, zu dem das Variabilitätsmodell gehört. Die Kinder der Wurzel enthalten Referenzen zu Variationspunkten in diesem Funktionsnetz; in unserem Fall können nur Funktionen Variationspunkte sein. Variationspunkte besitzen eine beliebige Anzahl an möglichen Varianten für die Funktion. Die am Variationspunkt vorhandene Variabilität wird dabei durch Kardinalitäten zwischen

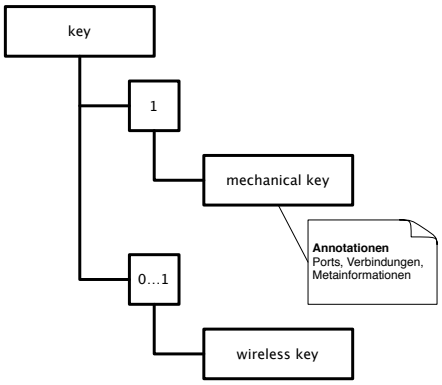

Abbildung 8: Grundlegende Struktur von funktionalen Feature Modellen

Variationspunkt und Varianten beschrieben.

Im Beispiel in Abbildung 8 ist die abstrakte Funktion `Key` ein Variationspunkt. Er kann zur Bindungszeit an einen konventionellen mechanischen Schlüssel oder an eine Kombination aus mechanischem Schlüssel und Funkschlüssel gebunden werden. Die Variante des mechanischen Schlüssels ist mit einer Kardinalität von `1` gekennzeichnet, was sich in Anlehnung an die in der Literatur verbreiteten Variabilitäts-Klassifikation mit `mandatory` übersetzten lässt. Die Variante des Funkschlüssels ist mit einer Kardinalität von `0...1` gekennzeichnet, was sich mit `optional` übersetzten lässt. Kardinalitäten erlauben im Gegensatz zu diesen Begriffen eine umfangreichere und genauere Beschreibung von Variabilität.

Funktions-Varianten sind als vollständige Funktionen spezifiziert; d.h. sie werden zusammen mit ihren Ports und den daran anhängenden Verbindungen definiert. Solche Ports und Verbindungen sind damit indirekt selbst variabel. Unterscheiden sich bestimmte Varianten untereinander nur in der Zielfunktion einiger ihrer Verbindungen, so fassen wir sie trotzdem als eigenständige Varianten auf und definieren sie vollständig.

Durch die Restriktion auf eine flache Baumdarstellung können wir die Ports und Verbindungen, die mit einer Funktionsvariante implizit gewählt werden, übersichtlicher darstellen. Trotzdem bewahren wir die Ausdruckskraft von Feature-Modellen mit unbegrenzter Tiefe, da wir auch Varianten als Variationspunkte auffassen und notieren können, wie Abbildung 9 zeigt.

Analog zu konventionellen Feature-Modellen besteht die Möglichkeit, Varianten über Constraints in Beziehung zu setzen. Constraints erlauben die Definition von bedingenden oder ausschließenden Abhängigkeiten zwischen Varianten. Zusammen mit den Verbindungs-Anforderungen ermöglicht dies eine automatisierte Überprüfung, ob eine gewünschte Konfiguration möglich ist, oder nicht. Eine konkrete Constraints-Sprache ist noch Gegenstand gegenwärtiger Überlegungen.

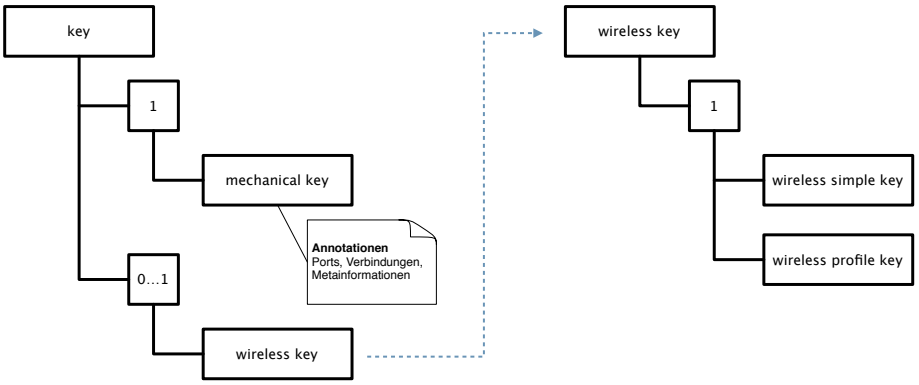

Abbildung 9: Abbildung von Hierarchien auf funktionale Feature Modelle

3.2 Metainformationen in funktionalen Feature-Modellen

Funktionsnetze sind statische Modelle. Will man sie als Grundlage für frühe Prototypen und Simulationen nutzen, wie es bei Kommunikationsmatrizen gängige Praxis ist, benötigt man zusätzliche Informationen über die dynamische Kommunikation zwischen Funktionen; insbesondere interessieren Abläufe und zeitliche Anforderungen. Diese Abläufe und Anforderungen sind abhängig von der konkreten Konfiguration von Funktionsvarianten (und impliziten Signalvarianten). Varianten entstehen und ändern sich vor allem während des Grobentwurfes mit Funktionsnetzen. Es sollte dort möglich sein, auch die von Varianten abhängigen Simulationsparameter dort notieren zu können, um eine zeitgleiche explorative Konstruktion von Prototypen zu erleichtern.

Funktionale Feature-Modelle bieten diese Möglichkeit in Form von Annotationen, die Varianten hinzugefügt werden können. Unter Annotationen verstehen wir hierbei Schlüssel/Wert-Paare, mit dem sich beliebige Informationen auszeichnen und festhalten lassen. Dabei kann es sich beispielsweise um Parameter handeln, mit denen auf dem Funktionsnetz basierende Matlab/Simulink-Dokumente [Mat] konfigurieren lassen.

4 Werkzeugunterstützung

Um die Umsetzung unseres Konzeptes als Werkzeug zu evaluieren, entwickeln wir gegenwärtig einen Werkzeug-Prototypen. Das Werkzeug soll eine simultane Sicht auf das Variabilitätsmodell und das zugehörige Funktionsnetz bieten (siehe Abbildung 10). Es soll in seiner ersten Iteration die Bearbeitung des Funktionsnetzes, sowie das Hinzufügen und Verwalten von Variationspunkten ermöglichen. Im Anschluss soll es das Verknüpfen mehrerer Funktionsnetze und Partitionen und ihre zugehörigen Variabilitätsmodelle unterstützen. Das Werkzeug soll Dokumente anderer Werkzeuge importieren und exportieren

und so eine Anbindung der Funktionsnetz-Dokumente an Werkzeuge des Anforderungs- und Variantenmanagements wie *pure::variants* [pur] und Werkzeuge zur Simulation wie Matlab/Simulink und Matlab/Stateflow ermöglichen.

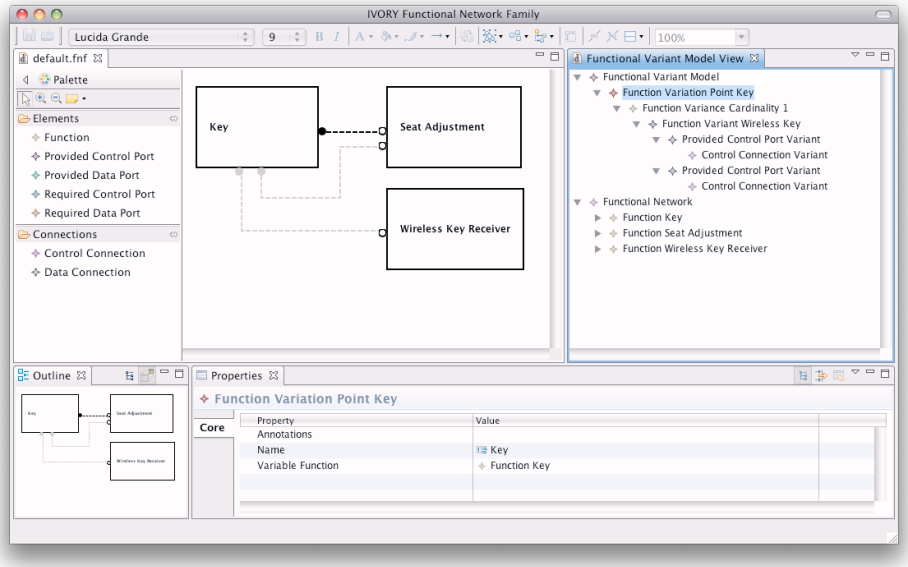

Abbildung 10: Funktionsnetz-Familien-Prototyp

5 Zusammenfassung

Wir haben die Herausforderung der Automobilindustrie beschrieben, von einem technisch orientierten Systementwurf zu einem logischen anforderungsorientierten Systementwurf übergehen zu müssen. Wir haben für das Systementwurf-Dokument der Funktionsnetze eine Notation und eine zugehörige Modellierungsmethodik beschrieben, in deren Rahmen Richtlinien zur Erstellung von Funktionsnetzen erhoben werden können. Auf diese Weise erzielen wir eine Reduktion der Komplexität und erhöhen die Verständlichkeit auf logischer Systemebene.

Weiterhin haben wir eine für Funktionsnetze angepasste Notation kardinalitäts-basierter Feature-Modelle vorgeschlagen, um Variabilität in Funktionsnetzen explizit darstellen zu können und sie so zu Funktionsnetz-Familien zu erweitern. Hierdurch werden Variationspunkte explizit identifiziert und Variantenmanagement ermöglicht. Wir haben kurz dargestellt, wie sich solche Funktionsnetz-Familien an Simulations-Werkzeuge anbinden lassen. Somit wird es weiterhin möglich sein, Funktionsnetze als Basis für Simulationen zu verwenden. Die Konzepte werden anhand eines Prototypen evaluiert.

Literatur

[AUT] AUTOSAR Website. http://www.autosar.org.

[Bro06] Manfred Broy. Challenges in automotive software engineering. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, Seiten 33–42, New York, NY, USA, 2006. ACM.

[CCG+07] Philippe Cuenot, DeJiu Chen, Sebastien Gerard, Henrik Lonn, Mark-Oliver Reiser, David Servat, Carl-Johan Sjostedt, Ramin Tavakoli Kolagari, Martin Torngren und Matthias Weber. Managing Complexity of Automotive Electronics Using the EAST-ADL. In *ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems*, Seiten 353–358, Washington, DC, USA, 2007. IEEE Computer Society.

[CHE04] Krzysztof Czarnecki, Simon Helsen, und Ulrich Eisenecker. Formalizing Cardinality-based Feature Models and their Staged Configuration. 2004.

[CN01] Paul Clements und Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, August 2001.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak und A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Bericht, Carnegie-Mellon University Software Engineering Institute, November 1990.

[LL05] Jochen Ludewig und Horst Lichter. *Software Engineering*. Dpunkt.Verlag GmbH, July 2005.

[MA09] Cem Mengi und Ibrahim Armaç. Functional Variant Modeling for Adaptable Functional Networks. In *Third International Workshop on Variability Modelling of Software-intensive Systems*, Seiten 83–92, 2009.

[Mat] Mathworks Website. http://www.mathworks.com/.

[PBKS07] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger und Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. In *FOSE '07: 2007 Future of Software Engineering*, Seiten 55–71, Washington, DC, USA, 2007. IEEE Computer Society.

[PBvdL05] Klaus Pohl, Günter Böckle und Frank J. van der Linden. *Software Product Line Engineering : Foundations, Principles and Techniques*. Springer, September 2005.

[pur] pure systems Website. http://www.pure-systems.com.

[WR06] Henning Wallentowitz und Konrad Reif, Hrsg. *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen*. Vieweg, Wiesbaden, 2006.