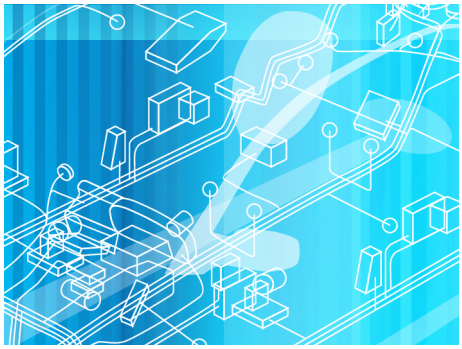
Fraunhofer Institut
Software- und
Systemtechnik

Anwendung von Kohäsions- und Kohärenzmetriken auf VEIA-Architekturmodelle zur Bewertung von Produktlinien

Stefan Mann



ISST-Bericht 86/08
Juni 2008

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

ISSN 0943-1624

Das Projekt VEIA

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

Am Projekt sind vier Partner beteiligt:

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

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

Autor

Dieses Dokument wurde geschrieben von:

Stefan Mann (Fraunhofer ISST).

Zusammenfassung

Ziel des Projekts VEIA ist es, auf der Grundlage der Konzepte der Produktlinientechnik eine Methode für die verteilte Entwicklung und Integration von Automotive-Systemen zu erarbeiten. Mit Hilfe von Architekturbeschreibungen auf unterschiedlichen Abstraktionsebenen wird eine Produktlinie entworfen. Die Bewertung der Entwürfe zielt darauf ab, die Qualität der Entwürfe sicherzustellen und eine Entscheidungsgrundlage für eine optimale Realisierungsstrategie zu liefern. Hierzu ist die Anwendung von Metriken vorgesehen.

In dem Bericht wird betrachtet, wie ausgewählte Metriken auf VEIA-Architekturmodelle angewendet werden können. Hierzu gehört die Analyse, welche Modelle betrachtet werden, welche Informationen als Eingaben für die Metriken erhoben werden und inwieweit diese Daten in einer Modellierung schon vorliegen oder ob sie erst noch berechnet werden müssen. Für diese Untersuchung wurden Metriken zur Bewertung von Kohäsion und Kohärenz in Produktlinienarchitekturen ausgewählt.

Neben einer detaillierten Diskussion der ausgewählten Metriken erfolgt die Analyse, welche Daten für die Metriken erhoben werden müssen und wie sie erhoben bzw. berechnet werden können. Sie dient als Anforderungsspezifikation für eine Werkzeugunterstützung, die um die Spezifikation der Eingabedaten für die PSSF- sowie für die Kohäsions- und Kohärenz-Metriken in Form von DTDs ergänzt wird.

Die diskutierten Metriken zu Kohäsion und Kohärenz werden auf eine Funktionsmodellierung der Fallstudie »Condition-Based Service« (CBS) angewendet.

Inhalt

1 Einleitung

Ziel des Projekts VEIA ist es, auf der Grundlage der Konzepte der Produktlinientechnik eine Methode für die verteilte Entwicklung und Integration von Automotive-Systemen zu erarbeiten. Mit Hilfe von Architekturbeschreibungen auf unterschiedlichen Abstraktionsebenen wird eine Produktlinie entworfen. Die Bewertung der Entwürfe zielt darauf ab, die Qualität der Entwürfe sicherzustellen und eine Entscheidungsgrundlage für eine optimale Realisierungsstrategie zu liefern. Hierzu ist die Anwendung von Metriken vorgesehen.

1.1 Verwendung von Metriken im VEIA-Referenzprozess

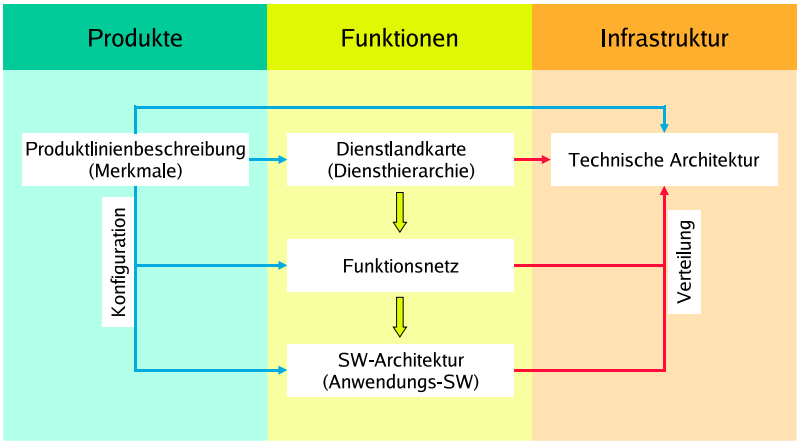
Im VEIA-Referenzprozess [GEKM07] wurden die Artefakte zur Erfassung und Modellierung von Architektursichten festgelegt, die den Entwurf von Produktlinien von Automotive-E/E-Systemen unterstützen (siehe Abbildung 1):

- Produktlinienbeschreibung durch Produktmerkmale,
- Dienstlandkarte,
- Funktionsnetz,
- Softwarearchitektur,
- Technische Architektur,
- Konfigurationsmodelle und
- Verteilungsmodelle.

Mit Hilfe dieser Artefakte wird ein zu entwickelndes System in den unterschiedlichen Sichten auf unterschiedlichen Abstraktionsniveaus beschrieben. Der Nutzen der Artefakte liegt zum einen in der präzisen Beschreibung des Systems. Darüber hinaus sind mit Hilfe der Architekturmodelle vielfältige Bewertungen des Systems möglich.

Zu den Architekturmodellen gehören die Artefakte »Funktionsnetz«, »Softwarearchitektur« und »Technische Architektur«. Diese Modelle sind hierarchisch aufgebaute, komponentenbasierte Architekturmodelle, die um Varianzinformationen angereichert wurden, um Produktlinien auch auf Architekturebene komprimiert darstellen zu können.

Abbildung 1 Artefakte des VEIA-Referenzprozesses.



Bewertungen als Analyseaktivitäten im Referenzprozess wurden in [GKM07a] diskutiert. Die Ermittlung der Kennzahlen, die für die Bewertung der Entwürfe verwendet werden, erfolgt durch Metriken. Metriken zur Bewertung von Architekturen bzw. Produktlinien wurden bereits in [GKM07b] untersucht. In diesem Bericht wurden ausführlich die PSSF-Metriken [Kie06] diskutiert, andere Metriken wurden nur vorgestellt.

1.2 Ziel des Berichts

In dem vorliegenden Bericht wird betrachtet, wie weitere Metriken auf VEIA-Architekturmodelle angewendet werden können. Hierzu gehört die Analyse, welche Modelle betrachtet werden, welche Informationen als Eingaben für die Metriken erhoben werden und inwieweit diese Daten in einer Modellierung schon vorliegen oder ob sie erst noch berechnet werden müssen.

Für diese Untersuchung wurden Metriken zur Bewertung von Kohäsion und Kohärenz in Produktlinienarchitekturen ausgewählt. Neben einer gegenüber dem Bericht [GKM07b] detaillierteren Diskussion der ausgewählten Metriken erfolgt die Analyse, welche Daten für die Metriken erhoben werden müssen und wie sie erhoben bzw. berechnet werden können. Sie dient als Anforderungsspezifikation für eine Werkzeugunterstützung, die um die Spezifikation der Eingabedaten für die PSSF- sowie für die Kohäsions- und Kohärenz-Metriken in Form von DTDs ergänzt wird.

Des Weiteren wird die Bewertung des Fallbeispiels mit Hilfe von Metriken fortgesetzt, die in [GKM07c] begonnen wurde. Die in dem vorliegenden Bericht diskutierten Metriken zu Kohäsion und Kohärenz werden auf eine Funktionsmodellierung der Fallstudie »*Condition-Based Service*« (*CBS*) angewendet.

1.3 Struktur des Berichts

Bewertungsgegenstand der Metriken sind VEIA-Architekturmodelle, mit denen Produktlinien entworfen werden können. Wesentliche Modellierungskonzepte, soweit sie zum Verständnis im Rahmen des Berichts notwendig sind, werden in Kapitel 2 erläutert. Hier wird auch der Zusammenhang von Produktlinienmodell, Konfiguration und der zugehörigen Produktmodelle skizziert.

Kapitel 3 stellt die Kohäsions- und Kohärenz-Metriken vor und bettet sie in den Kontext der VEIA-Architekturmodellierung ein. Insbesondere wird auf die VEIA-spezifische Interpretation des in den Metriken verwendeten Servicebegriffs eingegangen. Um die Metriken auf die Architekturmodelle anwendbar zu machen, müssen die Modellierungskonzepte für Komposition, Mehrfachverwendung und Varianz mit den in den Originalmetriken zugrundegelegten Annahmen abgeglichen werden. Siehe hierzu Abschnitt 3.2. Die detaillierte Diskussion der einzelnen Metriken erfolgt in den Abschnitten 3.3 bis 3.6.

In Kapitel 4 wird die zugrundeliegende Informationsbasis skizziert: die Metamodelle, die die VEIA-Modellierungskonzepte definieren, spezifizieren zugleich die Anforderungen an ein Datenmodell einer geeigneten Werkzeugunterstützung für die Informationserfassung sowie für die Bewertung der Modelle. Aufgabe einer Werkzeugunterstützung für Bewertungen ist es, die für die Metriken notwendigen Daten aus den Modellierungen zu erheben.

Die beispielhafte Anwendung der Metriken wird in Kapitel 5 für die Fallstudie »*Condition-Based Service*« (*CBS*) gezeigt. So sind die für die Bewertung herangezogenen Modelle inklusive ihrer Aufarbeitung für die Metrikanwendung angegeben. Die daraus erhobenen Daten sowie die Ergebnisse der entsprechenden Berechnungen sind in tabellarischer Form dargestellt.

Die Spezifikation der von einer Werkzeugunterstützung zu erhebenen Daten, um eine Produktlinie mit Hilfe der Metriken zu bewerten, wird in Form von DTDs angegeben: in Kapitel 6 für die Kohäsions- und Kohärenzmetriken, in Kapitel 7 für die PSSF-Metriken, die bereits in [GKM07b] ausführlich diskutiert und beispielhaft in [GKM07c] angewendet wurden. Letzere DTD ist bereits in dem VEIA-

Demonstrator prototypisch umgesetzt worden (siehe Kapitel 8). Eine Beispiel-XML gemäß der Modellierung in Kapitel 5 ist jeweils mit angegeben.

In Kapitel 8 erfolgt die Zusammenfassung der Ergebnisse. Zudem wird ein Ausblick auf eine geeignete Werkzeugunterstützung gegeben.

2 Architekturmodelle

Die im Rahmen des VEIA-Referenzprozesses betrachteten Architekturen von E/E-Systemen in Fahrzeugen sind Funktionsnetz, Architektur der Anwendungssoftware und die technische Architektur (Hardware-Topologie im Fahrzeug) [GEKM07]. Sie werden mit Hilfe einer Architekturbeschreibungssprache (vgl. die Übersicht zu typischen Konzepten von Architekturbeschreibungssprachen in [MT00]) spezifiziert: Die Modelle sind hierarchisch aufgebaute, komponentenbasierte Architekturmodelle. Um Produktlinien auch auf Architekturebene (kompakt) repräsentieren zu können, wurden die Architekturmodelle um Varianzinformationen angereichert, die es erlauben, sowohl Produktlinieninvarianten als auch architekturelle Unterschiede zwischen den Produktlinienprodukten zu erfassen sowie die Konfiguration der Produktlinienarchitekturen vorzunehmen, um einzelne Produktmodelle daraus ableiten zu können.

2.1 Architekturkonzepte

Komponenten sind die Bausteine in einem Architekturmodell. Schnittstellenbeschreibungen in Form von Ports und zugehörigen Portdeklarationen spezifizieren die Interaktionspunkte einer Komponente: welche Informationen empfangen oder gesendet bzw. welche Dienste bereitgestellt oder benötigt werden. Die Ports definieren die Fähigkeit, Komponenten miteinander zu vernetzen. Die Komposition von Komponenten ergibt hierarchisch zusammengesetzte Komponenten. Als grundlegende Eigenschaft für die VEIA-Architekturmodelle wird gefordert, dass zusammengesetzte Komponenten eine bloße Kapsel ihrer Unterkomponenten sind, die weder ein zusätzliches Verhalten der Komposition hinzufügen noch Verhalten, Schnittstellen, Funktionalität etc. verstecken. Das heißt insbesondere, dass Ports von Unterkomponenten, die nicht miteinander verbunden sind, nach außen über die zusammengesetzte Komponente delegiert werden müssen und nicht versteckt werden können.

Mehrfachverwendung von Komponenten und hierarchische Komposition von Komponenten werden in den VEIA-Architekturmodellen explizit konzeptionell unterschieden. In anderen Architekturbeschreibungssprachen, bspw. AUTOSAR-Software-Component-Description, aber auch in der UML, fallen Mehrfachverwendung und hierarchische Komposition zusammen, d. h., die Komposition einer zusammengesetzten Komponente besteht immer aus Verwendungen von »Kom-

ponententypen«[1], auch wenn diese Typen unter Umständen niemals zur mehrmaligen Wiederverwendung gedacht sind und nur einmal instanziiert werden.

Die Architekturartefakte des VEIA-Referenzprozesses (vgl. Abbildung 1), d. h. Funktionsnetz, Softwarearchitekturmodell und Hardwarearchitekturmodell, sind Spezialisierungen dieses Komponentenmodells.

2.2 Varianzkonzepte

Um Anforderungen an Produktlinien zu erfassen und um die Produktlinie abzugrenzen, hat sich die Strukturierung von Produktmerkmalen in Form von Merkmalsbäumen etabliert. Hiermit werden Produktmerkmale aus Sicht der Produktlinie in obligatorische und variable Merkmale unterschieden. Obligatorische Merkmale beschreiben die Invarianten in einer Produktlinie und repräsentieren die Gemeinsamkeiten aller Produkte, die zur der Produktlinie gehören. Variable Merkmale stellen Unterschiede zwischen den Produkten heraus. In Merkmalsmodellen wird zwischen optionalen Merkmalen, alternativen Ausprägungen eines Merkmals sowie variablen Kombinationen von Merkmalsgruppen (ODER-Varianz) unterschieden. Diese werden entlang einer hierarchischen Strukturierung erfasst, die eine Kombination von Spezialisierungs- und Dekompositionsbeziehung darstellt. Abhängigkeiten zwischen Merkmalen, die nicht entlang dieser Hierarchie erfassbar sind, werden durch Querverbindungen dargestellt.

Ein einzelnes Produkt entspricht einem »vertikal vollständigen Teilbaum«: Ausgehend vom Wurzelknoten des Merkmalsbaums gehören alle obligatorischen Untermerkmale zu diesem Teilbaum sowie alle für das Produkt ausgewählten variablen Merkmale. Durch Konfigurieren wird festgelegt, welche Ausprägung eines variablen Merkmals ausgewählt wird und so für das Produkt gültig ist.

Die VEIA-Architekturmodelle werden in Form von hierarchisch zusammengesetzten Komponenten beschrieben. Analog zu den Merkmalsmodellen werden entlang dieser Hierarchie Komponenten in Unterkomponenten dekomponiert. Die Dekomposition wird in obligatorisch und optional unterschieden. Zusätzlich sind XOR-

[1] In AUTOSAR werden die Verwendungen von Komponenten »Prototypen« genannt, in VEIA »Hooks«. Explizite Komponententypen gibt es in AUTOSAR nicht, weil jede zusammengesetzte Softwarekomponente Prototypen aggregiert. In VEIA sind mehrfachverwendbare Komponenten(typen) immer durch den Wurzelknoten der hierarchischen Komposition einschließlich aller Unterkomponenten bis zu den Blättern der Kompositionshierarchie repräsentiert.

Varianzpunkte von Komponenten eingeführt, die alternative Ausprägungen einer Komponente in der Architektur kapselt. Da Architekturbeschreibungen strukturell komplexer sind als Merkmalsbeschreibungen, spiegeln sich Varianzkonzepte auch in den Kommunikationsbeziehungen zwischen den Komponenten wider, d. h., Varianz muss auch auf Port- und Konnektorebene berücksichtigt werden.

Produktlinien- vs. Produktarchitekturmodelle

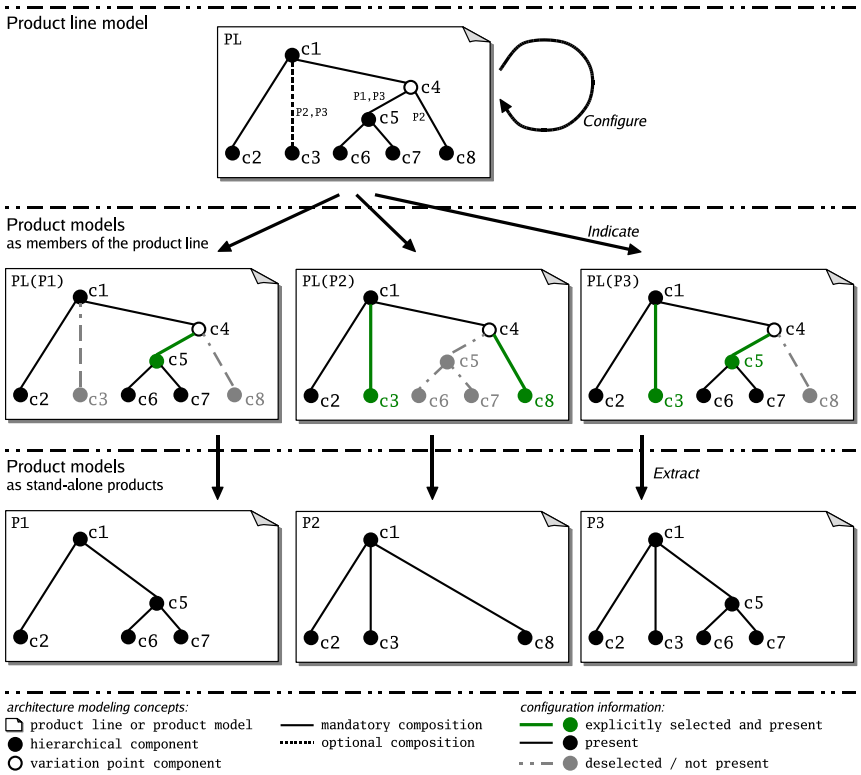
Die Integration von Varianzkonzepten in Architekturbeschreibungen führt zur Unterscheidung zwischen der Produktlinienarchitektur und einer Produktarchitektur.

In einem Produktlinienmodell, das die Gemeinsamkeiten und Unterschiede zwischen den Produkten der Produktlinie darstellt, gibt es invariante Komponenten, die in allen Produkten vorkommen, optionale Komponenten sowie alternative Komponenten, die durch eine Varianzpunktkomponente gekapselt sind. Durch Vergleich der alternativen Komponenten eines Varianzpunkts bzgl. ihrer Ports lassen sich auch für Ports bzw. Signale Gemeinsamkeiten und Unterschiede zwischen den Alternativen feststellen, so dass ebenfalls optionale und variable Ports in die Konzeption der VEIA-Architekturmodelle eingeflossen sind.

Varianz auf Komponentenebene fügt sich in die hierarchische Komposition von Komponenten ein, so dass bei einem Produktlinienentwurf optionale Komponenten und Varianzpunktkomponenten beliebig in die Hierarchie eingefügt werden können, d. h., auf welcher Hierarchieebene ein Varianzpunkt definiert wird, ist eine Entwurfsentscheidung.

In Abbildung 2 (oberste Ebene – »Product line model«) ist die hierarchische Struktur der Komponenten eines Produktlinienarchitekturmodells (`PL`) dargestellt: die Wurzelkomponente `c1` repräsentiert die gesamte modellierte Produktlinienarchitektur, die aus drei Komponenten `c2`, `c3` und `c4` besteht (Komponenten sind als Knoten im Baum dargestellt; hierarchisch zusammengesetzte Komponenten oder Blattkomponenten sind durch schwarz ausgefüllte Kreise in der Abbildung dargestellt). Komponente `c2` ist invariant, d. h., in allen Produkten der Produktlinie ist diese Komponente enthalten (die invariante Kompositionsbeziehung zwischen Komponenten ist als durchgezogene Linie dargestellt). Komponente `c3` ist optional (die optionale Kompositionsbeziehung zwischen Komponenten ist als gestrichelte Linie dargestellt), d. h., einige Produkte nutzen diese Komponente, andere nicht. Komponente `c4` ist wie `c2` ebenfalls in allen Produkten vorhanden, jedoch ist sie eine Varianzpunktkomponente mit den beiden Ausprägungen `c5` und `c8` (Varianzpunktkomponenten sind als nicht ausgefüllter Kreis dargestellt, ihre Alternativen sind die Unterkomponenten). Für die Produkte der Produktlinie bedeutet dies, dass für sie entweder Komponente `c5` oder Komponente `c8` verbaut wird.

Abbildung 2 Ableiten der Produktarchitekturmodelle aus dem Produktlinienarchitekturmodell.



Für ein Produkt muss die im Produktlinienmodell definierte Varianz entschieden werden, d. h., ob eine optionale Komponente in einem Produkt »verbaut« ist oder welche Ausprägung einer Varianzpunktkomponente für ein Produkt verwendet werden soll.

Das VEIA-Artefakt »Produktlinienbeschreibung«, das als Merkmalsmodell repräsentiert wird, dient zur Konfiguration von Architekturmodellen, d. h. zur Spezifikation, unter welchen Bedingungen welche Option oder Variante im Architekturmodell auszuwählen ist. Dadurch lassen sich die jeweiligen Architekturmodelle für die einzelnen Produkte aus dem Produktlinienarchitekturmodell ableiten. Die Gesamtheit aller Konfigurationen wird im VEIA-Artefakt »Konfigurationsmodell« festgehalten, das die Relation zwischen Merkmalen und variablen Komponenten beschreibt. Die Konfiguration des Produktlinienmodells ist im Produktlinienmodell in Abbildung 2 durch Annotation dargestellt, indem angegeben ist, für welche Produkte die optionale Komponente `c3` vorgesehen ist (nämlich für die Produkte `P2` und `P3`) und welche Alternative des Varianzpunkts `c4` genutzt wird (nämlich: `c5` für die Produkte `P1` und `P3`; `c8` für Produkt `P2`).

Der Zusammenhang zwischen dem explizit erstellten Produktlinienarchitekturmodell (PL) und den unter Zuhilfenahme der Konfigurationsinformationen daraus extrahierten Produktmodellen (P1, P2 und P3) ist ebenfalls in Abbildung 2 dargestellt. Die Modelle in der mittleren Ebene der Abbildung (PL(P1), PL(P2) und PL(P3)) repräsentieren die Produkte als Mitglieder der Produktlinie, d. h., es wird dargestellt, welche in Bezug auf die Produktlinie variablen Komponenten für das Produkt ausgewählt wurden (grün dargestellt) und welche deselektiert wurden und demzufolge nicht im Produkt vorhanden sind (grau dargestellt). Die Modelle in der unteren Ebene stellen die Produkte unabhängig von ihrem Produktlinienkontext dar. Da Produkte vollständig konfiguriert sein sollen, enthalten die von der Produktlinie losgelösten Produktmodelle keine Varianzpunkte mehr.

Die Ableitung von Modellen aus einem Produktlinienmodell kann auf Teilkonfigurationen erweitert werden. Dazu wird im Merkmalsmodell nur eine Teilmenge der Varianzpunkte entschieden. Über das Konfigurationsmodell werden dann die in Relation stehenden Varianzpunkte in der Produktlinienarchitektur aufgelöst; die nicht aufgelösten Varianzpunkte bleiben bestehen. Die entsprechenden Architekturmodelle der Teilproduktlinien können – wie die Produktmodelle – als Teile der Produktlinie (mit als selektiert oder deselektiert markierten Elementen) oder als eigenständige Produktlinien dargestellt werden.

3 Metriken zu Servicenutzungsgraden und Kohärenz in Produktlinienarchitekturen

Komponentenbasierte Architekturen können mit Hilfe von Metriken strukturell bewertet werden. In Hinblick auf das Szenario, dass Komponenten in unterschiedlichen Systemen wiederverwendet werden sollen, spielt die Frage, *wie gut* eine Komponente in ein System passt, eine wesentliche Rolle. Die Frage lässt sich durch die Betrachtung beantworten, *wieviel* der von einer Komponente zur Verfügung gestellten Funktionalität in dem betrachteten System wirklich verwendet wird und *inwieweit* die Anforderungen einer Komponente zur Erfüllung ihrer Funktionalität im betrachteten System bereitgestellt werden. Auf Basis einer Architekturbeschreibung lassen sich Indizien für die Beantwortung dieser Fragen durch Betrachtung der strukturellen Informationen ableiten, indem Maße für Kohäsion und Kohärenz (vgl. [BDW98, BWW99, Mis00, Mis01]) angewendet werden. So lässt sich beispielsweise auf Basis von Dienstbeschreibungen (*Services*) der Komponenten Kohäsion durch den *Servicenutzungsgrad* (*service utilization*) bewerten.

Auch bei der Produktlinienentwicklung sind diese Fragestellungen von Bedeutung. Hier ist es von Interesse, *inwiefern* und *wie gut* Komponenten in die einzelnen Produkte passen. Sind beispielsweise die Produkte zu unterschiedlich, so kann der Wiederverwendungsgrad von Komponenten für die Produktlinie zu gering sein, so dass der Synergieeffekt zu gering und der Aufwand für Entwicklung und Wartung der einzelnen Produkten auf Basis eines Produktlinienansatzes zu hoch ist.

Um Produktlinienarchitekturen bewerten zu können, muss die darin enthaltene Varianz berücksichtigt werden: Elemente, die in allen Produkten präsent sind, optionale Elemente sowie alternative Elemente. Metriken, die Aussagen zur Kohäsion und Kohärenz in Softwareproduktlinienarchitekturen treffen, werden in den Arbeiten [DMH01, HDM03, Mis06] vorgestellt.[2] Eine erste Diskussion dieser Metriken hinsichtlich der Anwendbarkeit auf die VEIA-Artefakte erfolgte bereits

[2] Zur Definition und Abgrenzung der Begriffe (und Maße zur) Kopplung, Kohäsion und Kohärenz verweisen wir auf die folgenden Arbeiten: In [Mis00, Mis01, Mis06] erfolgt die Definition der Kohärenz als Maß, um die externe, nutzungsorientierte Perspektive eines Artefakts im Gegensatz zur Betrachtung der internen Struktur des Artefakts zu bewerten. Dies geschieht in Abgrenzung des in der Literatur nicht einheitlich verwendeten Begriffs der Kohäsion. Lt. [Mis06] sind die Maße für Nutzungs- und Bereitstellungsgrad in [DMH01] sehr ähnlich zum Kohärenzkonzept. Einen detaillierten Überblick für Maße bzgl. Kopplung (*coupling*) und Kohäsion in der Literatur wird in [BWW99] bzw. [BDW98] gegeben.

in [GKM07b]. Im Folgenden wird die Anwendung der Metriken auf die VEIA-Artefakte genauer diskutiert und definiert.

3.1 Servicenutzungs- und Servicebereitstellungsgrad (*Service Utilization*)

Die Metriken zur Bestimmung eines Servicenutzungs- bzw. Servicebereitstellungsgrads[3] [DMH01, HDM03] dienen der Bewertung der strukturellen Qualität von Softwareproduktlinienarchitekturen. Sie geben verschiedene Maße an, um zu beurteilen, wie gut eine bestimmte Komponente in die Produkte einer Produktlinie hineinpasst oder wie gut sämtliche Komponenten in einem Produkt der Produktlinie, in allen Produkten einer Produktlinie bzw. in der Produktlinienarchitektur aufeinander abgestimmt sind. Unter dem Begriff *Service* (*Dienst*) werden dabei öffentliche Methoden, Funktionen oder direkt zugreifbare Datenstrukturen von Komponenten verstanden. Der Servicenutzungs- bzw. -bereitstellungsgrad ist dabei ein Maß, das ausdrückt, wie viele der angebotenen bzw. geforderten Dienste einer Komponente auch wirklich in der bewerteten Architektur verwendet bzw. zur Verfügung gestellt werden. Eine Komponente ist dabei eine in verschiedenen Architekturen (Produkten) wiederverwendbare Einheit. Der Begriff *Service* ist in den Arbeiten generisch gehalten, er soll auf verschiedene Architekturbeschreibungssprachen (ADLs) anwendbar sein.

Die SU-Metriken bestehen aus einer Reihe von Maßen, die unterschiedliche Aspekte beleuchten. Dabei bauen die Metriken aufeinander auf:

- Bewertung einer einzelnen Komponente im Kontext einer bestimmten Architektur,
- Bewertung einer komponentenbasierten Architektur,
- Bewertung einer Komponente bzgl. ihrer Verwendung in allen Produktarchitekturen einer Produktlinie,
- Bewertung einer Produktlinienarchitektur durch Vergleich der Produktarchitekturen.

[3] Die Metriken zur Bestimmung eines Servicenutzungs- bzw. Servicebereitstellungsgrads werden im Folgenden kurz mit *SU-Metriken (service utilization metrics, SUM)* bezeichnet.

3.2 Metrikanwendung auf VEIA-Artefakte (Überblick)

Um die SU-Metriken auf VEIA-Architekturmodelle (siehe Kapitel 2) anwenden zu können, müssen sie dahingehend untersucht werden, welche der in den VEIA-Modellen vorliegenden Informationen den von den Metriken geforderten Daten entsprechen und inwieweit die Metriken auf den gleichen bzw. vergleichbaren Modellierungskonzepten basieren. Insbesondere ist zu klären,

- inwieweit der Komponentenbegriff in der Metrik und in den VEIA-Modellen übereinstimmt,
- wie der Servicebegriff in den Metriken hinsichtlich der VEIA-Konzepte interpretiert werden kann bzw. muss,
- ob die hierarchische Komposition von Komponenten in den Metriken berücksichtigt wird oder wie mit der Hierarchie umgegangen werden muss, um die Metriken anwenden zu können,
- inwieweit die Varianzkonzepte übereinstimmen und
- wie mit den Mehrfach- und Wiederverwendungskonzepten in VEIA-Modellen bzgl. der Metrikanwendung verfahren werden muss.

Die Metriken werden im Folgenden hinsichtlich ihrer Anwendbarkeit auf die VEIA-Artefakte *Funktionsnetz* und *Softwarearchitektur* untersucht und entsprechend der Modellierungskonzepte angepasst. Die Diskussion erfolgt nach dem Schema:

- Angabe der Metrik aus der zitierten Literatur,
- danach Diskussion der Anpassung und Anwendung.

Im Folgenden wird der prinzipielle Ansatz skizziert, der anschließend für die einzelnen Metriken detailliert durchgeführt wird.

3.2.1 Interpretation des Komponenten- und des Servicebegriffs

Die VEIA-Architekturmodelle *Funktionsnetz* und *Softwarearchitektur* sind strukturell sehr ähnlich: Komponenten, die hierarchisch aufgebaut sind. Die Schnittstellenbeschreibung der Komponenten erfolgt über Ports (Eingangs- und Ausgangsports). Sie sind typisiert, dabei wird spezifiziert, welche Signaltypen ausgetauscht werden. Bei Softwarekomponenten gibt es die zusätzliche Unterscheidung zwischen

signalbasierter und operationsaufrufbasierter Kommunikation, d. h., Softwarekomponenten können auch Ports haben, die angeben, welche Operationen aufgerufen werden können bzw. benötigt werden. Siehe hierzu auch die in Abschnitt 4.2 bzw. Abschnitt 4.3 angegebenen Metamodelle für Funktionsnetze und Softwarearchitekturmodelle.

Funktionsnetz

Ein Funktionsnetz beschreibt das E/E-System aus logischer Sicht: Es werden Funktionen sowie die prinzipiellen Informationsflüsse in Form von Ports, Signalen und Konnektoren spezifiziert, die weitgehend implementierungsunabhängig die Funktionalität des E/E-Systems erfassen. Um die SU-Metriken auf VEIA-Funktionsnetze anzuwenden, wird folgende Interpretation angewendet:

Interpretation des Komponentenbegriffs
Die in den SU-Metriken bewerteten Komponenten werden mit Funktionen gleichgesetzt.

Interpretation des Servicebegriffs
Zwei Interpretationsmöglichkeiten:

1. Der Servicebegriff wird auf Ports angewendet: angebotene Dienste werden mit Ausgangsports von Funktionen gleichgesetzt, geforderte bzw. verwendete Dienste mit Eingangsports von Funktionen.
2. Der Servicebegriff wird auf Signale angewendet: angebotene Dienste werden mit Ausgangssignalen von Funktionen gleichgesetzt, geforderte bzw. verwendete Dienste mit Eingangssignalen.

Softwarearchitekturmodell

VEIA- bzw. AUTOSAR-Softwarearchitekturmodelle beschreiben die softwaretechnische Umsetzung der im Funktionsnetz beschriebenen Anwendungslogik. In diesem Sinne ist das Artefakt der Softwarearchitektur eine Verfeinerung des Funktionsnetzes. Vor allem die Festlegung auf Datenstrukturen, Kommunikationsprotokolle und die Vorbereitung der Verteilung auf die technische Infrastruktur sind wichtige Aspekte beim Softwareentwurf. Bzgl. Kommunikation werden Ports in Sender-/Receiver- und Server-/Client-Ports unterschieden. Die Anwendung der SU-Metriken muss deshalb die Portunterscheidung wie folgt berücksichtigen:

Interpretation des Komponentenbegriffs
Die SU-Metriken für Komponenten werden auf Softwarekomponenten angewendet.

Interpretation des Servicebegriffs

Zwei Interpretationsmöglichkeiten:

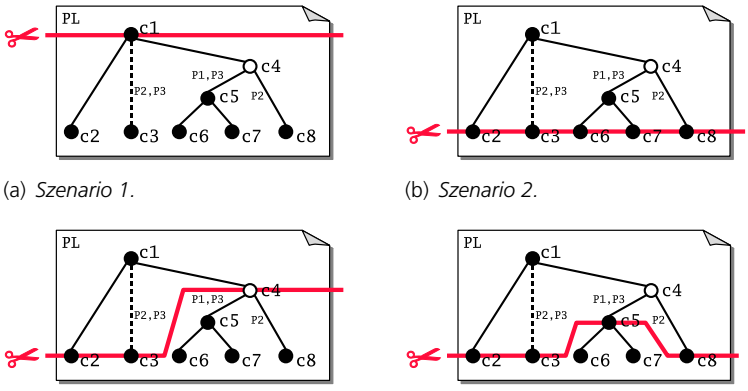
1. Der Servicebegriff wird (wie schon bei Funktionsmodellen) auf Ports angewendet, wobei hier jedoch die Differenzierung der Ports in Server-/Client-Ports und Sender-/Receiver-Ports berücksichtigt werden muss:
   - Server/Client: angebotene Dienste werden mit Eingangsports von Softwarekomponenten gleichgesetzt, geforderte bzw. verwendete Dienste mit Ausgangsports.
   - Sender/Receiver: angebotene Dienste werden mit Ausgangsports von Softwarekomponenten gleichgesetzt, geforderte bzw. verwendete Dienste mit Eingangsports (im Datenmodell: *required ports*).
2. Der Servicebegriff wird auf die Elemente von Ports, d. h. auf Operationen bzw. Signale angewendet:
   - Operationen (bei Client-/Server-Ports): angebotene Dienste werden mit angebotenen Operationen von Softwarekomponenten gleichgesetzt, geforderte bzw. verwendete Dienste mit Operationsaufrufe.
   - Signale (bei Sender-/Receiver-Ports): angebotene Dienste werden mit Ausgangssignalen von Softwarekomponenten gleichgesetzt, geforderte bzw. verwendete Dienste mit Eingangssignalen.

Bzgl. des Umgangs mit Hierarchie, Varianz und Mehrfachverwendung unterscheiden sich Funktionsmodelle und Softwarearchitekturmodelle nicht, weil die Modellierungskonzepte gleich oder ähnlich sind, und können deshalb gemeinsam diskutiert werden.

3.2.2 Umgang mit Hierarchie in den Architekturmodellen

Eine hierarchische Komposition von Komponenten wird in den SU-Metriken, so wie sie in [DMH01, HDM03] definiert werden, nicht berücksichtigt. Demzufolge muss für die Anwendung der SU-Metriken das Architekturmodell (d. h. das Funktionsnetz bzw. die Softwarearchitektur) »flachgeklopft« werden, indem die zu berücksichtigenden Komponenten ausgewählt werden und alle Ober- und Unterkomponenten ignoriert werden. Die für die ausgewählten Komponenten zu erhaltenen Kommunikationsbeziehungen (entlang der Pfade bestehend aus Delegationskonnektoren und horizontalen Konnektoren) müssen durch äquivalente horizontale Konnektoren ersetzt werden. – Ein flachgeklopftes Modell besteht nur noch aus (virtuell) atomaren Komponenten, deren Ports mit horizontalen Konnektoren verbunden sind. Würde die Hierarchie nicht herausgerechnet werden, käme es im Allgemeinen zu einer verfälschten Bewertung der Vernetzung der Komponenten.

Abbildung 3 Setzen des Schnitts durch die Hierarchie.



(a) *Szenario 1.* (b) *Szenario 2.*

(c) *Szenario 3.* (d) *Szenario 4.*

Das Flachklopfen der hierarchischen Komponentenmodelle entspricht einem horizontalen Schnitt durch die Hierarchie, so wie es in Abbildung 3 illustriert ist. An welcher Stelle der Schnitt angesetzt wird, ist im Allgemeinen frei wählbar. Hier spielen vor allem organisatorische Gesichtspunkte eine Rolle, beispielsweise unterschiedliche Zuständigkeiten für Komponenten bzw. Teilbereiche in der Architektur, aber auch der am besten geeignete Detaillierungsgrad ist für die Bewertung relevant.

In Abbildung 3 sind unterschiedliche, gültige Schnitte illustriert: In Abbildung 3(a) würde nur das Gesamtsystem, betrachtet als eine Komponente, bewertet werden. Da die SU-Metriken jedoch die Vernetzung von Komponenten bewerten, ist dieser Schnitt für die SU-Metriken nicht geeignet. Mit dem Schnitt in Abbildung 3(b) würden die SU-Metriken auf alle atomaren Komponenten (die Blätter in der Kompositionshierarchie) angewendet werden. Dagegen würden gemäß des Schnitts in Abbildung 3(c) und Abbildung 3(d) auch Komponenten bewertet werden, die hierarchisch zusammengesetzt sind. Gründe hierfür können darin liegen, dass nur Komponenten auf gleichem Abstraktionsgrad bzw. mit gleichem Komplexitätsgrad miteinander bewertet werden sollten.

3.2.3 Produktlinien- vs. Produktmodell

Die SU-Metriken bewerten unterschiedliche Aspekte in einer Produktlinie und definieren deshalb verschiedene Maße:

- Maße zur Bewertung der Einbettung einer (Produktlinien-)Komponente in einer Produktlinienarchitektur (z. B. c2 bzgl. a_{PL} von Abbildung 2)
- Maße zur Bewertung der Verwendung der Komponente aus der Produktlinienarchitektur in einem Produkt, d. h., wie gut die Komponente in die Produktarchitektur passt (z. B. c2 bzgl. a_{P2})
- zusammenfassende Maße für die Verwendung einer Komponente in sämtlichen Produktarchitekturen (Mittelwert und Spanne) (z. B. Durchschnittswert für c2, d. h. Durchschnitt von c2 bzgl. a_{P1}, c2 bzgl. a_{P2} und c2 bzgl. a_{P3})
- Maße zur Bewertung einer Produktlinienarchitektur insgesamt (z. B. von a_{PL})
- Bewertung einer einzelnen Produktarchitektur (z. B. von a_{P2})
- zusammenfassende Maße für alle Produktarchitekturen einer Produktlinie (Mittelwert und Spanne) (z. B. Durchschnitt der Werte von a_{P1}, a_{P2} und a_{P3})

Hieraus folgt, dass für die Anwendung der SU-Metriken sowohl die direkt in einem Produktlinienarchitekturmodell erfassten Informationen als auch Konfigurationsinformationen hinzugezogen und berücksichtigt werden müssen. Durch Auswertung der Konfigurationsinformationen lassen sich die einzelnen Produktarchitekturmodelle herleiten. Der zuvor diskutierte Schnitt durch das Produktlinienmodell ist auch auf die Produktmodelle anzuwenden, um die für die Metriken notwendigen Eingangsdaten ermitteln zu können. Abbildung 4 illustriert dies für das in Abbildung 2 dargestellte, abstrakte Beispiel für den Schnitt gemäß Szenario 3 in Abbildung 3(c).

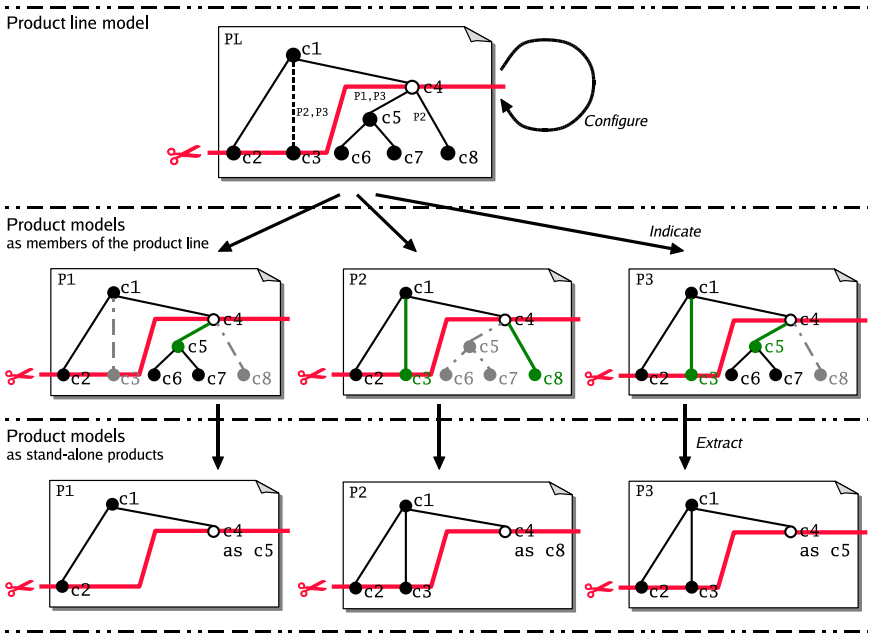
Das hierfür zugrundeliegende Metamodell, das spezifiziert, wie Produktlinienarchitekturmodell, Konfiguration, Schnitt und Produktarchitekturmodelle miteinander in Beziehung gesetzt sind, ist in Abschnitt 4.1 beschrieben.

3.2.4 Umgang mit Varianz in den Architekturmodellen

In VEIA erfassen wir Varianz sowohl auf Komponentenebene als auch auf Port- sowie Signalebene. Dies muss bei der Anwendung der SU-Metriken wie folgt berücksichtigt werden:

Für die Einzelbewertung einer Komponente in einer Produktlinienarchitektur werden alle optionalen oder variierenden Ports bzw. Signale wie obligatorische behandelt, da in der Produktlinienarchitektur prinzipiell vorgesehen sein sollte (und deshalb auch bewertet werden sollte), dass (bzw. ob) variable Elemente in irgendeiner Konfiguration auch Verwendung finden.

Abbildung 4 Flachklopfen des Produktlinienarchitekturmodells gemäß des Schnitts von Szenario 3.



Für die Einzelbewertung einer Komponente in einem Produktmodell, also in dem entsprechend einer Konfiguration aus dem Produktlinienmodell abgeleiteten Architekturmodell, werden optionale oder variierende Ports oder Signale nur berücksichtigt, wenn sie in der Konfiguration auch ausgewählt sind.

Für zusammenfassende Bewertungen der Produktlinie müssen gemäß der Konfigurationsinformationen die Einzelarchitekturen gemessen werden. Optionale Komponenten bzw. optionale Ports oder Signale, die für ein Produkt abgewählt sind, sind aus der Metrikberechnung herauszunehmen.

Der angesetzte Schnitt durch das Produktlinienmodell beeinflusst die Menge der zu bewertenden Komponenten. Erfolgt der Schnitt durch eine Varianzpunktkomponente (oder oberhalb, vgl. Schnitt in Szenario 3 bzw. 1 in Abbildung 3(c)), so werden deren Alternativen in der Bewertung ignoriert, allenfalls anhand variierender Ports und Kommunikationsbeziehungen spiegelt sich die Varianz wider. Erfolgt der Schnitt unterhalb einer Varianzpunktkomponente (bspw. der Schnitt in Szenario 2 oder 4 in Abbildung 3(c)), so werden die Alternativen (oder deren Unterkomponenten) bewertet. Im so flachgeklopften Architekturmodell können sie wie unabhängige optionale Komponenten behandelt werden, d. h., in bestimmten Konfigurationen sind sie vorhanden und in anderen nicht. Wird eine Produktlinienarchitektur bewertet, so bedeutet solch ein Schnitt, dass eine ursprünglich

am Varianzpunkt angeschlossene Komponente nun mit mehreren Komponenten verbunden ist (die Anzahl entspricht einem Faktor von der Anzahl der alternativen Komponenten des ursprünglichen Varianzpunkts – in Abhängigkeit, wie tief der Schnitt unterhalb des Varianzpunkts angesetzt war). In einem Produktarchitekturmodell ist die Komponente wieder mit entsprechend weniger Komponenten, nämlich mit der ausgewählten Alternative oder ihren Unterkomponenten, verbunden.

3.2.5 Umgang mit Mehrfachverwendungskonzepten

Mehrfachverwendungen von Komponenten werden in den VEIA-Architekturmodellen explizit modelliert und gekennzeichnet.[4]

Um die SU-Metriken anzuwenden, müssen die Mehrfachverwendungen entsprechend berücksichtigt werden: Es muss unterschieden werden, ob die architekturelle Rolle einer wiederverwendeten Komponente (dargestellt als *Hook*) bewertet wird oder ob die wiederverwendete Komponente bewertet wird. Folgende Bewertungsszenarien können unterschieden werden:

1 Es wird die Verwendung einer Komponente (*Hook*) in einer (Produktlinien-) Architektur bewertet. Hier werden Hooks wie normale, hierarchisch zusammengesetzte Komponenten behandelt.

   (Vgl. bspw. die Funktion für die Verschleißerfassung der Bremsbeläge in der Fallstudie *CBS*, die in Kapitel 5 angegeben ist: Die Verschleißerfassung für die vorderen Bremsbeläge (Funktion `CbsWdBreakPadsFront` ist als Hook in Abbildung 9, S. 60 modelliert) wird getrennt von der Verschleißerfassung für die hinteren Bremsbeläge (Funktion `CbsWdBreakPadsRear` ist ebenfalls ein Funktionshook) bewertet, obwohl hier jeweils derselbe Komponententyp (Funktionstyp `CbsWdBreakPadsAdaptive`) verwendet wird.)

2 Es wird die wiederverwendete Komponente (Komponententyp) in einer Produktarchitektur bewertet, d. h., die Verwendungen dieser Komponente werden zusammenfassend betrachtet.

   (Zum Beispiel: alle Verwendungen der Verschleißerfassung für die Bremsbeläge werden in einer Architektur zusammenfassend mittels Mittelwertberechnung

[4] In anderen Sprachen, wie bspw. in der UML oder auch bei der AUTOSAR-Software-Component-Description, fallen Mehrfachverwendung von Komponenten und hierarchische Komposition von Komponenten konzeptionell zusammen.

bewertet, d. h., es wird bewertet, wie gut der Komponententyp (`CbsWdBreakPadsAdaptive`) in die Architektur passt, um für die Verschleißerfassung für sowohl die vorderen als auch für die hinteren Bremsbeläge verwendet zu werden.)

3 Es wird die wiederverwendete Komponente (Komponententyp) in der Produktlinie betrachtet, d. h., wie gut sämtliche Verwendungen sich in sämtlichen Produkten der Produktlinie einbetten.

(Bzgl. des Beispiels: Dieses Szenario entspricht einer Kombination aus dem ersten und zweiten Szenario, da das Maß die Werte für die vorderen und hinteren Bremsbeläge pro Produkt und dann für die Produktlinie zusammenfasst.)

Außerdem müssen Hooks der Vergleichbarkeit wegen beim Flachklopfen des Architekturmodells jeweils konform behandelt werden, d. h., um den gleichen Detaillierungsgrad für die Bewertung heranzuziehen, sollte der gleiche Schnitt für alle Verwendungen eines Komponententyps angewendet werden.

3.3 Metriken für die Bewertung von Einzelkomponenten und ihrer Verwendung in einer Architektur: Nutzungsgrad und Bereitstellungsgrad

Für eine Einzelkomponente lässt sich der Nutzungsgrad der von ihr angebotenen Dienste (*PSU*) und der Bereitstellungsgrad der von ihr geforderten Dienste (*RSU*) bestimmen. Die Maße sind dabei architekturspezifisch, d. h., die Komponente wird im Kontext einer bestimmten Architektur bewertet. Die Architektur kann sowohl für ein Produkt als auch für eine Produktlinie sein. Die weiteren Metriken bauen auf beiden Begriffen auf.

Sei c eine Komponente in einem Architekturmodell a, dann wird der Nutzungsgrad für die von der Komponente angebotenen Dienste (*Provided Service Utilization*, PSU) wie folgt berechnet:

Formel 3.1

$$PSU(c,a) = \frac{P_{actual}(c,a)}{P_{total}(c,a)}$$

Der Bereitstellungsgrad für die von der Komponente c benötigten Dienste (*Required Service Utilization*, RSU) berechnet sich analog wie folgt:

Formel 3.2

$$RSU(c,a) = \frac{R_{actual}(c,a)}{R_{total}(c,a)}$$

wobei:

P_{actual} (c, a) ist die Anzahl der von der Komponente c angebotenen Dienste, die auch tatsächlich von anderen Komponenten der Architektur a verwendet werden.

P_{total} (c, a) ist die Anzahl aller von der Komponente c angebotenen Dienste.

R_{actual} (c, a) ist die Anzahl der von der Komponente c benötigten Dienste, die auch tatsächlich von anderen Komponenten der Architektur a angeboten werden.

R_{total} (c, a) ist die Anzahl aller von der Komponente c benötigten Diente.

Dabei gelten folgende Eigenschaften für eine gegebene Architektur a:
$\forall\, c \bullet P_{actual}(c,a) \leq P_{total}(c,a)$ sowie $\forall\, c \bullet R_{actual}(c,a) \leq R_{total}(c,a)$
Außerdem: $\forall\, c \bullet 0 \leq PSU(c,a) \leq 1$ und $\forall\, c \bullet 0 \leq RSU(c,a) \leq 1$.

Ein ermittelter PSU- bzw. RSU-Wert 1 zeugt von einer guten Einbettung der Komponente: alle angebotenen (bzw. geforderten) Dienste werden in der Architektur verwendet (angeboten). Die Komponente passt bzgl. dieses Maßes optimal in die Architektur. Ein PSU-Wert von 0 zeigt an, dass kein von der Komponente angebotener Dienst überhaupt verwendet wird, und deutet an, dass die Komponente aus der Architektur entfernt werden könnte oder hier ein Entwurfsfehler vorliegt. Bei einem RSU-Wert von 0 steht in Frage, inwieweit die Komponente in der gemessenen Architektur überhaupt ihre Aufgabe erfüllen kann, da die Voraussetzungen für den Einsatz der Komponente gar nicht erfüllt sind.

Die Gesamtzahl der angebotenen bzw. benötigten Dienste einer Komponente ist im Allgemeinen unabhängig von der Architektur. Wird jedoch eine konfigurierbare Komponente betrachtet, also eine Komponenten mit bspw. optionalen Diensten, so ist die jeweilige architekturspezifische Konfiguration zu berücksichtigen. Die Gesamtzahl der Dienste einer variablen Produktlinienkomponente ist folglich von der Konfiguration für das jeweilige Produkt abhängig.

3.3.1 *PSU* und *RSU* auf Ports von Funktionen angewendet

Gemäß der vorgestellten Anpassung lassen sich die PSU- und RSU-Maße an Funktionskomponenten bzgl. ihrer Ports anpassen. Für die Einzelbewertung einer Funktion ist es hier sogar egal, ob das Architekturmodell hierarchisch aufgebaut ist oder gemäß eines Schnitts flachgemacht wurde, da das PSU- und das RSU-Maß das Verhältnis der verwendeten angebotenen bzw. der bereitgestellten benötigten Dienste einer Funktion zur Gesamtzahl der Dienste bestimmen und die Anzahl der

angeschlossenen Funktionen bzw. Ports dabei keine Rolle spielen (im Unterschied zur Kohärenzmetrik in Abschnitt 3.6). Die hierarchische Strukturierung hat keinen Einfluss auf den PSU- und den RSU-Wert einer Komponente.[5] Erst weiterführende Metriken benötigen den Schnitt, weil sie einzelne Werte zusammenfassen (zum Beispiel durch Mittelwertberechnung), weswegen dann die konkrete Menge der gemessenen Komponenten eine Rolle spielt.

Für zusammenhängende und umfassende Bewertungen (vgl. die Beispielrechnung in Abschnitt 5.4) adaptieren wir jedoch die beiden Maße bereits für die Anwendung auf den Schnitt eines Architekturmodells:

Sei f eine Funktionskomponente in einem Architekturmodell a (für eine Produktlinie oder ein Produkt) und s ein gewählter Schnitt durch das Modell a, wobei f zum Schnitt s gehört. Das gemäß s flachgeklopfte Architekturmodell sei mit $a^\star$ bezeichnet, $f^\star$ ist die Entsprechung der Funktion f im flachgeklopften Architekturmodell $a^\star$. Dann:

Formel 3.3
$$PSU_{ports}(f, a, s) = \frac{\#f^\star.outputPorts_{used}}{\#f^\star.outputPorts_{total}}$$

Formel 3.4
$$RSU_{ports}(f, a, s) = \frac{\#f^\star.inputPorts_{used}}{\#f^\star.inputPorts_{total}}$$

wobei:

$f^\star.outputPorts_{used}$ ist die Menge der Ausgangsports der Funktion f, die im flachgeklopften Architekturmodell $a^\star$ gemäß des Schnitts s verwendet werden, d. h., für die es angeschlossene, externe, horizontale Konnektoren in $a^\star$ gibt.

Die Zahl der verwendeten Ausgangsports einer Funktion ist unabhängig vom Schnitt durch das Architekturmodell, da nur die Situation bewertet wird, ob der Port verwendet wird.[6] Somit gilt: $f^\star.outputPorts_{used} = f.outputPorts_{used}$.

[5] Ob ein Port verwendet wird, hängt davon ab, ob es am Port extern angeschlossene, horizontale Konnektoren gibt. Bei hierarchischen Architekturmodellen kann dies auch indirekt der Fall sein, wenn der Port nach oben delegiert wird (via Delegationskonnektoren). An einem Port extern angeschlossene Delegationskonnektoren implizieren keine Verwendung des Ports, da zusammengesetzte Komponenten die Komposition von Komponenten nur kapseln und davon abstrahieren, vgl. Kapitel 2.

[6] Bei flachgeklopften Architekturmodellen lässt sich die Anzahl der verwendeten Ports direkt an extern angeschlossenen, horizontalen Konnektoren erkennen. Bei hierarchisch aufgebauten

$f^{\star}.outputPorts_{total}$ ist die Menge aller Ausgangsports der Funktion $f^{\star}$ bzw. f. Die Zahl der Ausgangsports einer Funktion ist unabhängig vom Schnitt, so dass folgendes gilt: $f^{\star}.outputPorts_{total} = f.outputPorts_{total}$.

Wird eine Produktlinienarchitektur bewertet, so umfasst die Menge aller Ausgangsports sowohl die obligatorischen als auch optionalen bzw. abhängigen Ports der Funktion. Wird dagegen eine Produktarchitektur bewertet, so umfasst die Menge aller Ausgangsports neben den obligatorischen Ports nur die entsprechend konfigurierten, also im Produkt vorhandenen, optionalen und abhängigen Ports. Dies hat den Effekt, dass eine unterschiedliche Konfiguration von Sender- und Empfängerfunktion zu nicht optimalen Werten führt.

$f^{\star}.inputPorts_{used}$ ist die Menge der Eingangsports der Funktion f, die im flachgeklopften Architekturmodell $a^{\star}$ gemäß des Schnitts s bedient werden, d. h., für die es angeschlossene, externe, horizontale Konnektoren gibt.

Die Zahl der verwendeten Eingangsports einer Funktion ist unabhängig vom Schnitt (vgl. Erläuterung bei Ausgangsports), so dass gilt: $f^{\star}.inputPorts_{used} = f.inputPorts_{used}$.

$f^{\star}.inputPorts_{total}$ ist die Menge aller Eingangsports der Funktion $f^{\star}$ bzw. f. Die Zahl der Eingangsports einer Funktion ist unabhängig vom Schnitt, so dass gilt: $f^{\star}.inputPorts_{total} = f.inputPorts_{total}$.

Wie bei Ausgangsports werden der Gesamtzahl nur die entsprechend konfigurierten optionalen und abhängigen Ports hinzugezählt.

Die Metrik kann nur angewendet werden, wenn für eine Funktion f überhaupt Eingangs- bzw. Ausgangsports spezifiziert sind, d. h. $\#f.outputPorts_{\text{total}} > 0$ bzw. $\#f.inputPorts_{\text{total}} > 0$.

Interpretation der Maße

Das Maß ist gleich 1, wenn alle definierten Ports einer Funktion f in der Architektur a auch verwendet werden. Das Maß ist gleich 0, wenn kein Port der Funktion f

Modellen muss die Delegation des Ports nach oben soweit verfolgt werden, bis extern angeschlossene, horizontale Konnektoren für einen Delegationsport gefunden sind oder bis der Delegationsport der Wurzelfunktion erreicht ist (im letzteren Fall wäre der Port dann unbenutzt). Der Unterschied liegt hier folglich nur darin, wann der Aufwand einer Berechnung zum Tragen kommt. Sollen alle in diesem Bericht vorgestellten Metriken angewendet werden, so ist es sinnvoller, den Schnitt bzw. das flachgeklopfte Modell zu verwenden, anstatt jedes Mal die Hierarchie zu traversieren.

verwendet wird, d. h., die Funktion ist in der Architektur isoliert und wird insgesamt nicht benutzt. Je mehr Ports einer Komponente verbunden sind, desto besser passt die Komponente in die Architektur, umso näher liegt der Wert bei 1.

Werden die Maße im Kontext einer Produktlinie verwendet und sollen die Werte bzgl. der Produktarchitekturen verglichen werden, so hat die Varianz in der Produktlinie einen Einfluss auf die Werte: Wird eine optionale Komponente bewertet, die in einem Produkt abgewählt ist, so würde sie in diesem Produkt einen RSU- bzw. PSU-Wert von 0 bekommen, da sie in diesem Produkt nicht verbunden ist. Diese Situation ist jedoch erwünscht und sollte keinen negativen Einfluss auf die Maße haben. Um diese Situation in einer Werkzeugunterstützung zu erkennen bzw. geeignet zu neutralisieren, fließen bei den zusammenfassenden Metriken (z. B. Abschnitt 3.4) die Konfigurationsinformationen in die Berechnung ein.

Die Berechnung von *PSU* und *RSU* ist originär nur definiert für Funktionen, die mindestens einen Ausgangs- bzw. Eingangsports besitzen (anderenfalls liegt eine Division durch Null in der Berechnung vor). Um die Situation zu umgehen und um keine Fehlerwerte zu erzeugen (beispielsweise modellieren wir Sensorfunktionen meist ohne Eingangsports, entsprechend Aktuatorfunktionen ohne Ausgangsports), kann die Berechnung wie folgt adaptiert werden. Falls eine Funktion keinen Ausgangsport hat, können auch keine Ausgangsports der Komponente verwendet werden, so dass die Berechnung für PSU wie folgt ergänzt werden kann:

Formel 3.5

$$PSU'_{ports}(f,a,s) = \begin{cases} 1 & \text{für } f^{\star}.outputPorts_{total} = \varnothing \\ \frac{\#f^{\star}.outputPorts_{used}}{\#f^{\star}.outputPorts_{total}} & \text{für } f^{\star}.outputPorts_{total} \neq \varnothing \end{cases}$$

Analog kann die Berechnung für *RSU* erfolgen:

Formel 3.6

$$RSU'_{ports}(f,a,s) = \begin{cases} 1 & \text{für } f^{\star}.inputPorts_{total} = \varnothing \\ \frac{\#f^{\star}.inputPorts_{used}}{\#f^{\star}.inputPorts_{total}} & \text{für } f^{\star}.inputPorts_{total} \neq \varnothing \end{cases}$$

Anmerkung: Ein theoretischer Spezialfall ist, wenn für die Funktion sowohl kein Ausgangs- als auch kein Eingangsport definiert wurde (und die Funktion eigentlich »sinnlos« ist, zumindest auf dieser Abstraktionsebene). Dieser Fall würde von den Metriken nicht erkannt werden, weil *PSU'* als auch *RSU'* für die Funktion den Wert 1 liefern würden.

3.3.2 *PSU* und *RSU* auf Signale von Funktionen angewendet

Analog zur Anwendung auf Ports können PSU und RSU auch auf Signale von Funktionen angewendet werden. Die Anpassung der Formeln erfolgt genauso wie für Ports.

Sei f eine Funktionskomponente in einem Architekturmodell a (für eine Produktlinie oder ein Produkt) und s ein gewählter Schnitt durch das Modell a, wobei f zum Schnitt s gehört. Das entsprechend flachgeklopfte Architekturmodell sei mit $a^\star$ bezeichnet, $f^\star$ ist die Entsprechung der Funktion f im flachgeklopften Architekturmodell $a^\star$. Dann:

Formel 3.7
$$PSU_{signals}(f,a,s) = \frac{\#f^\star.outputSignals_{used}}{\#f^\star.outputSignals_{total}}$$

Formel 3.8
$$RSU_{signals}(f,a,s) = \frac{\#f^\star.inputSignals_{used}}{\#f^\star.inputSignals_{total}}$$

wobei:

$\boldsymbol{f^\star.outputSignals_{used}}$ ist die Menge aller Ausgangssignale der Funktion f, die im flachgeklopften Architekturmodell $a^\star$ gemäß des Schnitts s verwendet werden, d. h., für die Funktion f gibt es in $a^\star$ angeschlossene externe Funktionen, die diese Signale auch verarbeiten (empfangen). – Die bloße Existenz von extern angeschlossenen horizontalen Konnektoren (vgl. die Anpassung an Ports) reicht für die Signalebene nicht aus, die Empfängerfunktionen müssen die Signale auch in ihrer Portdeklaration enthalten.

$\boldsymbol{f^\star.outputSignals_{total}}$ ist die Menge aller Ausgangssignale der Funktion $f^\star$ bzw. f. Auch bzgl. der Signale ist die Gesamtzahl der Ausgangsports einer Funktion unabhängig vom Schnitt, so dass folgendes gilt: $f^\star.outputSignals_{total} = f.outputSignals_{total}$.

Analog zu Ports wird auch hier die Konfiguration bei Produktarchitekturen berücksichtigt, d. h., nur für das Produkt selektierte optionale Signale oder Signale von selektierten optionalen oder abhängigen Ports werden berücksichtigt.

$\boldsymbol{f^\star.inputSignals_{used}}$ ist die Menge aller Eingangssignale der Funktion f, die im flachgeklopften Architekturmodell $a^\star$ gemäß des Schnitts s auch geschickt werden können, d. h., für die Funktion f gibt es in $a^\star$ angeschlossene externe Funktionen, die diese Signale auch verschicken (analog zu Ausgangssignalen).

$f^{\star}.inputSignals_{total}$ ist die Menge aller Eingangssignale der Funktion $f^{\star}$ bzw. f, es gilt: $f^{\star}.inputSignals_{total} = f.inputSignals_{total}$.

Die Metrik kann nur angewendet werden, wenn für die Funktion überhaupt Eingangs- bzw. Ausgangssignale spezifiziert sind, d. h. $\#f.outputSignals_{\text{total}} > 0$ bzw. $\#f.inputSignals_{\text{total}} > 0$.

Interpretation der Maße

Das Maß ist gleich 1, wenn alle definierten Signale einer Funktion f in der Architektur auch verwendet (empfangen bzw. geschickt) werden. Das Maß ist gleich 0, wenn kein Signal der Funktion f verwendet wird, d. h., die Funktion kann strukturell zwar in der Architektur vernetzt sein (vgl. $PSU_{ports}\ (f)$ bzw. $RSU_{ports}\ (f)$), jedoch sind die Portdeklarationen inkompatibel zu den an f angeschlossenen Funktionen, so dass keine Kommunikation stattfinden kann.

Analog zu Ports hat die Varianz, die in einer Produktlinie definiert ist, ggf. einen Einfluss auf die Bewertung einer Funktion in einem Produktmodell. Wie Konfigurationsinformationen hier genutzt werden können, wird bei den zusammenfassenden Metriken diskutiert (Abschnitt 3.4).

Die Metriken könnten bzgl. leerer Signalmengen analog zu den Berechnungen bei Ports (*PSU'* bzw. *RSU'*, vgl. Formel 3.5 bzw. Formel 3.6) ergänzt werden.

3.3.3 *PSU* und *RSU* auf Ports von Softwarekomponenten angewendet

Die Anwendung der Metriken auf Softwarekomponenten mit Ports mit einem Sender / Receiver-Interface ist identisch zur Messung von Funktionen mit Ports (Abschnitt 3.3.1).[7] Für Softwarekomponenten mit Ports in Form von Client- / Server-Interfaces ist die *PSU* definiert über *Provide*-Ports, die *RSU* für *Require*-Ports.[8]

Die Interpretation der Maße ist analog zu der von Funktionen.

[7] Eingangsports, über die Signale fließen, haben im Datenmodell für VEIA-Softwarearchitekturen für das Richtungsattribut *direction* den Wert *require*. Ausgangsports, über die Signale fließen, haben den Wert *provide* für das Richtungsattribut *direction*. Siehe Abschnitt 4.3.

[8] Im VEIA-Datenmodell haben *Provide*-Ports von Softwarekomponenten die Richtungsangabe *provide*, dementsprechend *Require*-Ports die Richtungsangabe *require*. Siehe Abschnitt 4.3.

3.3.4 *PSU* und *RSU* auf Signale von Softwarekomponenten angewendet

Die Anwendung der Metriken auf Softwarekomponenten mit Signalen an Ports mit einem Sender / Receiver-Interface ist analog zu Funktionen, Abschnitt 3.3.2. Eingangsports, über die »benötigte« Signale fließen, haben für das Richtungsattribut *direction* den Wert *require*. Ausgangsports, über die »angebotene« Signale fließen, haben für das Richtungsattribut *direction* den Wert *provide*.

Für Softwarekomponenten mit Ports in Form von Client / Server-Interfaces ist die Anwendung gemäß der Intention der Metriken im Original: Die Elemente der Ports sind Operationen: Aufrufbare Operationen sind Angebote (*provide*), hierfür wird der *PSU*-Wert ermittelt. Benötigte Operationen (*require*) sind Operationen, die von der Komponente aufgerufen werden; hierfür wird der *RSU*-Wert ermittelt.

Die Interpretation der Maße ist wie gehabt.

3.4 Metriken für die Bewertung von Architekturen: Zusammengesetzter Nutzungs- und Bereitstellungsgrad

Mit den Metriken *CPSU* (*compound provided service utilization*) und *CRSU* (*compound required service utilization*) wird die interne Kohäsion einer Architektur ermittelt, d. h. der Grad der Servicenutzung bzw. -bereitstellung in einer Architektur. Die Architektur kann sowohl ein Produkt als auch eine Produktlinie repräsentieren.

Sei a ein Architekturmodell, dann ist dessen zusammengesetzter Servicenutzungsgrad (CPSU) das Verhältnis der verwendeten Dienste zur Anzahl aller angebotenen Dienste im Architekturmodell:

Formel 3.9

$$CPSU(a) = \frac{\sum_{c \in C(a)} P_{actual}(c, a)}{\sum_{c \in C(a)} P_{total}(c, a)}$$

Der zusammengesetzte Bereitstellungsgrad (CRSU) für ein Architekturmodell a berechnet sich analog:

Formel 3.10

$$CRSU(a) = \frac{\sum_{c \in C(a)} R_{actual}(c, a)}{\sum_{c \in C(a)} R_{total}(c, a)}$$

wobei:

$\boldsymbol{P_{actual}\ (c,\ a)}$ sowie $\boldsymbol{P_{total}\ (c,\ a)}$ wie vorher.

$\boldsymbol{R_{actual}\ (c,\ a)}$ sowie $\boldsymbol{R_{total}\ (c,\ a)}$ wie vorher.

$\boldsymbol{C\ (a)}$ ist die Menge der (für die Bewertung zu berücksichtigenden) Komponenten aus dem Architekturmodell a.

Aus der Eigenschaft $\forall\, c : C(a) \bullet P_{actual}(c, a) \leq P_{total}(c, a)$ folgt, dass der CPSU-Wert für ein Architekturmodell a zwischen 0 und 1 liegt: $0 \leq \boldsymbol{CPSU}(a) \leq 1$. Auch der CRSU-Wert liegt zwischen 0 und 1, weil $\forall\, c : C(a) \bullet R_{actual}(c, a) \leq R_{total}(c, a)$.

3.4.1 CPSU und CRSU auf Ports von Funktionen einer Architektur angewendet

Die Metriken für CPSU und CRSU in [DMH01, HDM03] beziehen sich auf flache Architekturmodelle, d. h., sie berücksichtigen nicht die hierarchische Komposition von Komponenten. Dies zeigt sich daran, dass beide Maße Aussagen über eine Menge von Komponenten treffen. In hierarchisch aufgebauten Architekturmodellen muss man die Menge der zu messenden Komponenten jedoch erst noch bestimmen. Der Grund hierfür liegt darin, dass Delegationsverbindungen Ports von Unterkomponenten nur durch die Hierarchie durchleiten, logisch gesehen sind solche Ports aber noch unverbunden. Das heißt insbesondere, zur Bestimmung von CPSU-Werten für die Gesamtarchitektur dürfen hierarchisch in Beziehung stehende Komponenten nicht gleichzeitig gemessen werden, weil sonst Ports bzw. Signale mehrfach in die Bewertung einfließen würden.

Um die Metriken auf VEIA-Architekturmodelle anwenden zu können, müssen demzufolge bestimmte Komponenten in der Hierarchie ausgewählt werden, die Auswahl erfolgt durch das Setzen eines Schnitts durch die Hierarchie (vgl. Abschnitt 3.2.2).

Sei a ein hierarchisch aufgebautes Funktionsmodell für eine Produktlinie oder ein Produkt und s ein gewählter Schnitt durch das Funktionsmodell a. Die Menge der Funktionen, die zum Schnitt s gehören, sei durch die Menge $\boldsymbol{functions}$ repräsentiert. Das gemäß des Schnitts s flachgeklopfte Architekturmodell sei mit $a^\star$ bezeichnet. Die Menge $\boldsymbol{functions}^\star$ entspricht der Menge $\boldsymbol{functions}$ im Kontext von $a^\star$, d. h., sie repräsentiert die Menge der Funktionen, die den Funktionen aus $\boldsymbol{functions}$ im

Kontext des flachgeklopften Architekturmodells $a^\star$ entsprechen. Dann:

Formel 3.11

$$CPSU_{ports}(a,s) = \frac{\sum_{f^\star \in functions^\star} \#f^\star.outputPorts_{used}}{\sum_{f^\star \in functions^\star} \#f^\star.outputPorts_{total}}$$

Formel 3.12

$$CRSU_{ports}(a,s) = \frac{\sum_{f^\star \in functions^\star} \#f^\star.inputPorts_{used}}{\sum_{f^\star \in functions^\star} \#f^\star.inputPorts_{total}}$$

Die CPSU als auch die CRSU kann sowohl für ein Produktlinienarchitekturmodell als auch für ein Produktmodell berechnet werden.

Interpretation der Maße

Das Maß ist gleich 1, wenn alle definierten Ports aller Funktionen der Architektur a in der Architektur auch verwendet werden. Das Maß ist gleich 0, wenn es überhaupt keinen Port in der Architektur a gibt, der verwendet wird, d. h., alle Funktionen sind strukturell isoliert.

Der Normalfall bei einer VEIA-Modellierung ist, dass der CPSU- und der CRSU-Wert gleich 1 ist, weil alle Funktionen miteinander so abgestimmt werden können.

Der Wert wird von 1 abweichen, wenn eine Funktion in eine Architektur eingesetzt wird, die nicht hundertprozentig mit den anderen Funktionen bzgl. der Ports harmoniert. Dieser Fall ist immer dann wahrscheinlich, wenn Funktionen von dritter Seite verwendet werden.

Der Wert kann auch von 1 in Produktarchitekturen abweichen, wenn diese aus Produktlinienarchitekturen abgeleitet sind und die Varianz nicht vollständig »abgefangen« bzw. durch Konfiguration aufgelöst wird, d. h., die Varianzbindung an zwei Stellen »asymmetrisch« erfolgte. Dies kann gewollt sein (bspw. bei Kombinationen von produktgenerischen und produktspezifischen Komponenten, beim so genannten 150 %-Ansatz) oder aber auch auf eine Inkonsistenz bei der Konfiguration hindeuten.

Der Wert weicht von 1 ab, falls deselektierte optionale Funktionen oder nicht ausgewählte Funktionsalternativen in die Berechnung einfließen. Soll dies nicht geschehen, so dürfen entweder solche Funktionen generell nicht berücksichtigt

werden (d. h., in die Berechnung fließen die Funktionen der Menge $function^{\star}$ abzüglich aller optionalen und variierenden Funktionen und ihrer Alternativen), oder es muss, wie im nächsten Abschnitt diskutiert ist, die Konfigurationsinformation entsprechend berücksichtigt werden.

Berücksichtigung von Varianz- bzw. Konfigurationsinformationen in den SU-Metriken

Werden vergleichend die Produktmodelle zu einem Produktlinienmodell bewertet und über die Funktionen aus dem Produktlinienmodell traversiert, so gibt es folgendes zu beachten: Für ein Produkt deselektierte optionale Funktionen bzw. deselektierte Alternativen von Funktionsvarianzpunkten müssen (eigentlich) aus der Berechnung der Nutzungs- bzw. Bereitstellungsgrade für ein Produktarchitekturmodell herausgenommen werden. Bleiben sie berücksichtigt, so würden solche Funktionen nicht angeschlossene Ports besitzen und die zugehörigen PSU- bzw. RSU-Werte wären 0. Dadurch würden die Variationsbreite und der Durchschnitt für PSU bzw. RSU für die Funktion sowie insgesamt die Architekturbewertungen mittels CPSU und CRSU negativ beeinflusst werden.

Um deselektierte Funktionen aus der Berechnung herauszunehmen, kann die Konfigurationsinformation wie folgt in den Berechnungsformeln Berücksichtigung finden:

Formel 3.13

$$CPSU'_{ports}(a,s) = \frac{\sum_{f^{\star} \in functions^{\star}} conf(a^{\star}, f^{\star}) * \#f^{\star}.outputPorts_{used}}{\sum_{f^{\star} \in functions^{\star}} conf(a^{\star}, f^{\star}) * \#f^{\star}.outputPorts_{total}}$$

Formel 3.14

$$CRSU'_{ports}(a,s) = \frac{\sum_{f^{\star} \in functions^{\star}} conf(a^{\star}, f^{\star}) * \#f^{\star}.inputPorts_{used}}{\sum_{f^{\star} \in functions^{\star}} conf(a^{\star}, f^{\star}) * \#f^{\star}.inputPorts_{total}}$$

wobei:

conf (a, f) ist die Konfigurationsinformation für der Funktion f im Produktmodell a der Produktlinie. Mit *conf (a)* wird die Konfiguration des Produktlinienmodells für das Produkt a repräsentiert.[9] Die Konfigurationsinformation *conf (a, f)* hat

[9] Die Konfiguration für sämtliche Produkte einer Produktlinie kann mit Hilfe der mathematischen

den Wert 1, wenn die Funktion f im Produkt a vorhanden ist. Sie hat den Wert 0, wenn die Funktion f für das Produkt a deselektiert wurde.

Werden die CPSU- und CRSU-Maße direkt auf ein Produktlinienarchitekturmodell angewendet (und nicht auf einem Produktarchitekturmodell), so werden optionale Funktionen und Funktionsalternativen wie obligatorische Funktionen behandelt. In diesem Fall gilt für sämtliche Funktionen: $conf(a,f) = 1$.

3.4.2 CPSU und CRSU auf Signale von Funktionen einer Architektur angewendet

Die Anwendung der Maße CPSU und CRSU auf Signale erfolgt analog zur Anwendung auf Ports von Funktionen, vgl. Abschnitt 3.4.1.

Interpretation der Maße

Das Maß ist gleich 1, wenn alle definierten Signale aller Funktionen der Architektur a in der Architektur auch verwendet, d. h. sowohl gesendet als auch empfangen, werden. Das Maß ist gleich 0, wenn es überhaupt kein Signal in der Architektur a gibt, das sowohl gesendet als auch empfangen wird. Ein positives Beispiel hierfür ist die Bewertung einer Top-Level-Funktion. Ein negatives Beispiel ist, wenn die Portdeklarationen für die Ports der miteinander verbundenen Funktionen nicht zu einander kompatibel sind.

Der Normalfall bei einer VEIA-Modellierung ist, dass der CPSU- und der CRSU-Wert gleich 1 ist, weil alle Funktionen miteinander so aufeinander abgestimmt sind, dass die Signale auch wirklich fließen.

Der Wert wird von 1 abweichen, wenn eine Funktion in eine Architektur eingesetzt wird, die nicht hundertprozentig mit den anderen Funktionen bzgl. der Portdekla-

Funktion $conf$ dargestellt werden: $conf(p)$ ergibt die Konfigurationsinformation aller Komponenten der Produktlinie *pl* bzgl. ihres Vorhandenseins im Produkt p.

$$conf : \mathbb{P}(Product \times \mathbb{P}(Component \times \mathbb{N}))$$

$$conf \in Product \rightarrow (Component \rightarrow \mathbb{N})$$
$$\forall p : Product \bullet (conf\ p) \in (Component \rightarrow \mathbb{N}) \land \mathrm{ran}(conf\ p) \subseteq \{0, 1\}$$

Der Einfachheit halber wird $conf(p, c)$ synonym zu $(conf\ p)(c)$ verwendet. Im Übrigen gilt, dass der Schnitt nicht die Konfiguration beeinflusst, d. h. $conf(a, c) = conf(a^\star, c^\star)$.

rationen harmoniert. Dieser Fall ist immer dann wahrscheinlich, wenn Funktionen von dritter Seite verwendet werden.

Der Wert kann auch von 1 in Produktarchitekturen abweichen[10], wenn diese aus Produktlinienarchitekturen abgeleitet sind und die Varianz nicht vollständig abgefangen bzw. durch Konfiguration aufgelöst wird. Dies kann gewollt sein (bspw. bei der Kombination von produktgenerischen und produktspezifischen Komponenten in einer Architektur) oder aber auch auf eine Inkonsistenz bei der Konfiguration hindeuten.

Berücksichtigung von Varianz- bzw. Konfigurationsinformationen in den SU-Metriken

Die Berücksichtigung von Varianz- bzw. Konfigurationsinformationen in den an Signale angepassten Metriken ist wie bei den Anpassungen an Ports und wird deshalb hier nicht erneut diskutiert. Siehe hierzu Abschnitt 3.4.1.

3.4.3 CPSU und CRSU auf Ports von Softwarekomponenten in einer Softwarearchitektur angewendet

Die Anpassung der Maße CPSU und CRSU an aus Softwarekomponenten bestehende Architekturmodelle und in Bezug auf Ports erfolgt analog zur Anwendung auf Funktionsmodelle, siehe Abschnitt 3.4.1.

3.4.4 CPSU und CRSU auf Ports von Softwarekomponenten in einer Softwarearchitektur angewendet

Ebenso ist die Anpassung der Maße CPSU und CRSU an Softwarearchitekturmodelle hinsichtlich der Signale analog zur entsprechenden Anwendung auf Funktionsmodelle, siehe Abschnitt 3.4.2.

[10] vgl. Diskussion bzgl. Konfigurationsberücksichtigung.

3.5 Metriken zur Bewertung der Varianz in Produktlinien

Im Gegensatz zu den bisher beschriebenen Metriken, die sowohl auf ein Produktarchitekturmodell als auch auf ein Produktlinienarchitekturmodell angewendet werden können und nur das jeweilige Modell allein bewerten, wird mit den folgenden Metriken eine Produktlinie inklusive ihrer Produkte bewertet.

Es werden Maße erhoben, die Aussagen für die Komponenten einer Produktlinienarchitektur treffen, indem angeschaut wird, wie sich diese Komponenten in die einzelnen Produktarchitekturen einbetten. Die Bewertungen für die Komponenten im Kontext der Produktarchitekturmodelle werden durch Verteilungsmaße, die die Spanne an gemessenen Werten angibt, sowie durch Maße für Mittelwerte zusammengefasst. Diese Maße dienen dazu, die Varianz (bspw. XOR-Varianz und Optionalität von Komponenten) in Produktlinienarchitekturen zu bewerten.

Variationsbreite von PSU bzw. RSU einer Komponente bzgl. ihrer Verwendungen in den Produktarchitekturen einer Produktlinie

Die Variationsbreite (Spanne) von *PSU* bzw. *RSU* einer Komponente c_{pl} einer Produktlinienarchitektur a_{pl} wird über die einzelnen Produktarchitekturen a_p berechnet:[11]

Formel 3.15

$$Span_{PSU}(c_{pl}, a_{pl}) = \biguplus_{p \in Products(pl)} \{PSU(c_p, a_p)\} = \biguplus_{p \in Products(pl)} \left\{ \frac{P_{actual}(c_p, a_p)}{P_{total}(c_p, a_p)} \right\}$$

Formel 3.16

$$Span_{RSU}(c_{pl}, a_{pl}) = \biguplus_{p \in Products(pl)} \{RSU(c_p, a_p)\} = \biguplus_{p \in Products(pl)} \left\{ \frac{R_{actual}(c_p, a_p)}{R_{total}(c_p, a_p)} \right\}$$

wobei:

Products (pl) ist die Menge aller Produkte der Produktlinie pl.

p ist ein Produkt aus der Produktmenge.

[11] Das Symbol $\uplus$ steht für die Vereinigung in Multimengen, d. h. Mengen, in denen Elemente mehrfach vorkommen können und die Anzahl ihres Vorkommens in der Menge von Interesse ist.

a_{pl} ist das Produktlinienarchitekturmodell.

a_p ist das Architekturmodell des Produkts p der Produktlinie pl, abgeleitet aus dem Produktlinienarchitekturmodell a_{pl}.

c_p repräsentiert die Verwendung der Produktlinienkomponente c_{pl} in dem Produktarchitekturmodell a_p.

$PSU\ (c_p, a_p)$ repräsentiert die Anwendung des Maßes *PSU* auf die Komponente c im Kontext des Produktarchitekturmodells a_p des Produkts p.

$RSU\ (c_p, a_p)$ repräsentiert die Anwendung des Maßes *RSU* auf die Komponente c im Kontext des Produktarchitekturmodells a_p des Produkts p.

$P_{actual}\ (c_p, a_p)$ bzw. $P_{total}\ (c_p, a_p)$ wie oben.

$R_{actual}\ (c_p, a_p)$ bzw. $R_{total}\ (c_p, a_p)$ wie oben.

Anhand dieser Maße lässt sich erkennen, inwieweit die bewertete Komponente in die einzelnen Produktarchitekturen hineinpasst. Liegen die meisten Werte nahe 1, so lässt dies auf eine gute strukturelle Qualität in allen Produkten bzgl. dieser Komponente schließen. Mit Hilfe der Spanne lassen sich auch Ausreißer identifizieren: Gilt für eine Komponente ein mehrheitlich guter Wert, aber nur in einem Produkt ein schlechterer, so könnte dies daran liegen, dass entweder das Produkt nicht »richtig« konfiguriert wurde oder aber zu stark von der Produktlinie abweicht. Auch kann es sein, dass die Produktlinienarchitektur entsprechend umgestaltet werden müsste. In [HDM03] werden hierzu als Beispiele das Hinzufügen von optionalen Komponenten oder das Aufteilen einer Komponente in unterschiedliche (produktspezifische) Varianten aufgeführt. Liegen die meisten Werte bei 0, so deutet dies daraufhin, dass die Komponente generell nicht optimal auf die einzelnen Produkte abgestimmt ist.

Durchschnitt von PSU bzw. RSU einer Komponente bzgl. ihrer Verwendungen in den Produktarchitekturen einer Produktlinie

Der Durchschnitt der *PSU*s bzw. *RSU*s einer Komponente c_{pl} in einer Produktlinienarchitektur a_{pl} berechnet sich wie folgt:

Formel 3.17

$$Average_{PSU}(c_{pl}, a_{pl}) = \frac{\sum_{p \in Products(pl)} PSU(c_p, a_p)}{\#Products(pl)} = \frac{\sum_{p \in Products(pl)} \frac{P_{actual}(c_p, a_p)}{P_{total}(c_p, a_p)}}{\#Products(pl)}$$

Formel 3.18

$$Average_{RSU}(c_{pl}, a_{pl}) = \frac{\sum_{p \in Products(pl)} RSU(c_p, a_p)}{\#Products(pl)} = \frac{\sum_{p \in Products(pl)} \frac{R_{actual}(c_p, a_p)}{R_{total}(c_p, a_p)}}{\#Products(pl)}$$

wobei:

Products (pl) ist die Menge aller Produkte der Produktlinie pl.

p ist ein Produkt aus der Produktmenge *Products (pl)*.

c_p repräsentiert die Verwendung der Produktlinienkomponente c_{pl} in dem Produktarchitekturmodell a_p des Produkts p.

PSU (c_p, a_p) repräsentiert die Anwendung des Maßes *PSU* auf die Komponente c_{pl} im Kontext des Produktarchitekturmodells a_p des Produkts p.

P_{actual} (c_p, a_p) bzw. ***P_{total} (c_p, a_p)*** wie oben.

RSU (c_p, a_p) die Anwendung des Maßes *RSU* auf die Komponente c_{pl} im Kontext des Produktarchitekturmodells a_p des Produkts p repräsentiert.

R_{actual} (c_p, a_p) bzw. ***R_{total} (c_p, a_p)*** wie oben.

Anhand dieser Maße lässt sich erkennen, inwieweit die bewertete Komponente im Schnitt in die einzelnen Produktarchitekturen hineinpasst. Liegt der Durchschnitt bei 1, so lässt dies auf eine durchschnittlich gute strukturelle Qualität in allen Produkten bzgl. dieser Komponente schließen. Die Komponente erfüllt alle Anforderungen aus Produktsicht, wird aber auch nicht »unterfordert«, d. h. der Anteil an in einem Produkt nicht verwendeter Funktionalität ist gering. Liegt der Wert bei 0, so sollte man die Verwendung dieser Komponente überdenken bzw. die Architektur umgestalten, da die Komponente offensichtlich nicht auf die Produkte optimal abgestimmt ist.

Variationsbreite von CPSU bzw. CRSU einer Produktlinienarchitektur

Die Variationsbreite (Spanne) von *CPSU* bzw. *CRSU* einer Produktlinienarchitektur a_{pl} wird über die einzelnen Produktarchitekturen a_p ermittelt und wird wie folgt berechnet:

Formel 3.19

$$Span_{CPSU}(a_{pl}) = \biguplus_{p \in Products(pl)} \{CPSU(a_p)\} = \biguplus_{p \in Products(pl)} \left\{ \frac{\sum_{c_p \in C(a_p)} P_{actual}(c_p, a_p)}{\sum_{c_p \in C(a_p)} P_{total}(c_p, a_p)} \right\}$$

Formel 3.20

$$Span_{CRSU}(a_p l) = \biguplus_{p \in Products(pl)} \{CRSU(a_p)\} = \biguplus_{p \in Products(pl)} \left\{ \frac{\sum_{c_p \in C(a_p)} R_{actual}(c_p, a_p)}{\sum_{c_p \in C(a_p)} R_{total}(c_p, a_p)} \right\}$$

wobei:

Products (pl) ist die Menge aller Produkte der Produktlinie pl.

p ist ein Produkt aus der Produktmenge *Products (pl)*.

C (a_p) ist die Menge der (für die Bewertung zu berücksichtigenden) Komponenten in der Architektur a_p des Produkts p.

CPSU (a_p) repräsentiert die Anwendung des Maßes *CPSU* auf das Architekturmodell a_p des Produkts p der Produktlinie pl.

CRSU (a_p) repräsentiert die Anwendung des Maßes *CRSU* auf das Architekturmodell a_p des Produkts p der Produktlinie pl.

c_{a_p} repräsentiert die Verwendung der Produktlinienkomponente c_{pl} in dem Produktarchitekturmodell a_p des Produkts p.

P_{actual} (c_p, a_p) bzw. ***P_{total} (c_p, a_p)*** wie oben.

R_{actual} (c_p, a_p) bzw. ***R_{total} (c_p, a_p)*** wie oben.

Die Spanne für die zusammengesetzten Nutzungs- und Bereitstellungsgrade lässt Rückschlüsse auf die interne Kohäsion, auf die interne strukturelle Integrität, zu. Sind die meisten Werte bei 1, so sind die Komponenten in allen Produkten gut aufeinander abgestimmt. Einzelne Ausreißer lassen sich erkennen, dies kann daraufhin zurückzuführen sein, dass für manche Produktlinienkomponenten produktspezifische Varianten erforderlich sein könnten.

Durchschnitt von CPSU bzw. CRSU einer Produktlinienarchitektur

Der Durchschnitt von *CPSU* bzw. *CRSU* einer Produktlinienarchitektur a_{pl} berechnet sich wie folgt:

Formel 3.21

$$Average_{CPSU}(a_{pl}) = \frac{\sum_{p \in Products(pl)} CPSU(a_p)}{\#Products(pl)} = \frac{\sum_{p \in Products(pl)} \frac{\sum_{c_p \in C(a_p)} P_{actual}(c_p, a_p)}{\sum_{c_p \in C(a_p)} P_{total}(c_p, a_p)}}{\#Products(pl)}$$

Formel 3.22

$$Average_{CRSU}(a_{pl}) = \frac{\sum_{p \in Products(pl)} CRSU(a_p)}{\#Products(pl)} = \frac{\sum_{p \in Products(pl)} \frac{\sum_{c_p \in C(a_p)} R_{actual}(c_p, a_p)}{\sum_{c_p \in C(a_p)} R_{total}(c_p, a_p)}}{\#Products(pl)}$$

wobei:

Products (pl) ist die Menge aller Produkte der Produktlinie pl.

p ist ein Produkt aus der Produktmenge *Products (pl)*.

CPSU (a_p) repräsentiert die Anwendung des Maßes *CPSU* auf das Architekturmodell a_p des Produkts p der Produktlinie pl.

CRSU (a_p) repräsentiert die Anwendung des Maßes *CRSU* auf das Architekturmodell a_p des Produkts p der Produktlinie pl.

c_p repräsentiert die Verwendung der Produktlinien-Komponente c_{pl} in dem Produktarchitekturmodell a_p des Produkts p.

P_{actual} (c_p, a_p) bzw. ***P_{total} (c_p, a_p)*** wie oben.

R_{actual} (c_p, a_p) bzw. ***R_{total} (c_p, a_p)*** wie oben.

Analog zur Spanne lässt auch der Durchschnitt für die zusammengesetzten Nutzungs- und Bereitstellungsgrade der Produktarchitekturen Rückschlüsse auf die interne Kohäsion in den einzelnen Produktarchitekturen zu.

3.5.1 Variationsbreite und Durchschnitt auf Funktionen bzw. Softwarekomponenten von Produktlinienarchitekturen angewendet

Die Metriken zur Bewertung der Varianz (Spanne und Durchschnitt bei Komponenten) lassen sich nach dem gleichen Schema, das bereits auf den Einzelmetriken PSU bzw. RSU angewendet wurde (vgl. Abschnitt 3.3), anpassen, um die spezifizierte Varianz für Funktionen bzw. Softwarekomponenten zu bewerten.

Die im Folgenden aufgeführten Maße können für die Komponenten (gilt gleichermaßen für Funktionen als auch Softwarekomponenten) einer Produktlinienarchitektur a_{pl} berechnet werden, c_{pl} steht dabei für die Komponente der Produktlinienarchitektur. Mit s wird der angesetzte Schnitt zur Auswahl der zu bewertenden Komponenten repräsentiert, wobei die Komponente c_{pl} zum Schnitt s gehören muss.

$SpanPSU_{ports}$ (c_{pl}, a_{pl}, s)
Variationsbreite der *PSU*s bzgl. der Ports einer Komponente

$SpanRSU_{ports}$ (c_{pl}, a_{pl}, s)
Variationsbreite der *RSU*s bzgl. der Ports einer Komponente

$AveragePSU_{ports}$ (c_{pl}, a_{pl}, s)
Durchschnitt der *PSU*s bzgl. Ports einer Komponente

AverageRSU$_{ports}$ (c_{pl}, a_{pl}, s)
Durchschnitt der *RSU*s bzgl. Ports einer Komponente

SpanPSU$_{signals}$ (c_{pl}, a_{pl}, s)
Variationsbreite der *PSU*s bzgl. der Signale einer Komponente

SpanRSU$_{signals}$ (c_{pl}, a_{pl}, s)
Variationsbreite der *RSU*s bzgl. der Signale einer Komponente

AveragePSU$_{signals}$ (c_{pl}, a_{pl}, s)
Durchschnitt der *PSU*s bzgl. der Signale einer Komponente

AverageRSU$_{signals}$ (c_{pl}, a_{pl}, s)
Durchschnitt der *RSU*s bzgl. der Signale einer Komponente

Die Maße ergeben sich aus der Kombination der Originalmaße mit den Interpretationsmöglichkeiten für den Servicebegriff von Komponenten als Ports bzw. als Signale / Operationen.

Da die Einzelmetriken bereits ausführlich bzgl. ihrer Anpassung an die VEIA-Artefakte diskutiert wurden, wird hier auf die erneute Ausformulierung verzichtet.

Berücksichtigung von Varianz- bzw. Konfigurationsinformationen

In [DMH01, HDM03] wird die Spanne und der Durchschnitt nur für die invarianten Komponenten einer Produktlinienarchitektur ermittelt. Die Maße können aber auch für optionale oder alternative Komponenten von Interesse sein, beispielsweise in dem Sinne, inwieweit sich eine optionale Komponente integriert, wenn sie in einem Produkt verbaut wird.

Um diese Situationen auch erfassen zu können, müssen die Metriken, wie schon beim zusammengesetzten Nutzungs- bzw. Bereitstellungsgrad für Architekturmodelle (Abschnitt 3.4), so angepasst werden, dass die Konfigurationsinformation geeignet berücksichtigt wird:

Würden optionale und alternative Komponenten wie normale obligatorische Komponenten behandelt werden, so ist im Allgemeinen der Wert 0 immer in der Spanne enthalten, weil es Produkte geben dürfte, in denen die optionale oder alternative Komponente nicht verwendet wird. Entsprechend würde auch der Durchschnittswert für die Komponente negativ beeinflusst werden. Rechnet man jedoch die Situationen heraus, in denen die Komponenten für ein Produkt deselektiert ist, so erhält man die gewünschten Maße auch für optionale und alternative Komponenten.

Um in einem Produkt deselektierte Komponenten aus der Berechnung herauszunehmen, wird die Konfigurationsinformation wie folgt in den Berechnungsformeln berücksichtigt: Ist eine Komponente c_{pl} für ein Produkt deselektiert, so wird das Maß der Komponente für das Produktarchitekturmodell in der Zusammenfassung nicht berücksichtigt. Die Konfigurationsinformation für eine Komponente c_{pl} der Produktlinie *pl* für ein Produkt p wird durch die mathematische Funktion $conf(p, c_{pl})$ beschrieben (siehe die Definition in Abschnitt 3.4.1), der Rückgabewert ist 0 bei Deselektion und 1, wenn die Komponente im Produkt vorhanden ist. Obligatorische Produktlinienkomponenten haben für alle Produkte der Produktlinie den Konfigurationswert 1.

Folgende Formel ist die entsprechende Anpassung von Formel 3.15:

Formel 3.23

$$Span'_{PSU}(c_{pl}, a_{pl}) = \biguplus_{p \in Products(pl) \wedge conf(p,c_{pl})=1} \{PSU(c_p, a_p)\}$$

$$= \biguplus_{p \in Products(pl) \wedge conf(p,c_{pl})=1} \left\{ \frac{P_{actual}(c_p, a_p)}{R_{total}(c_p, a_p)} \right\}$$

Folgende Formel ist die entsprechende Anpassung von Formel 3.17:

Formel 3.24

$$Average'_{PSU}(c_{pl}, a_{pl}) = \frac{\sum_{p \in Products(pl)} conf(p, c_{pl}) * PSU(c_p, a_p)}{\sum_{p \in Products(pl)} conf(p, c_{pl})}$$

$$= \frac{\sum_{p \in Products(pl)} \frac{conf(p,c_{pl}) * P_{actual}(c_p,a_p)}{P_{total}(c_p,a_p)}}{\sum_{p \in Products(pl)} conf(p, c_{pl})}$$

wobei:

conf (p, c_{pl}) die Konfigurationsinformation für die Komponente c_{pl} im Produkt p ist. Mit $conf\,p$ wird die Konfiguration des Produktlinienmodells für das Produkt p repräsentiert. Die Konfigurationsinformation $conf(p, c_{pl})$ hat den Wert 1, wenn die Komponente c_{pl} im Produkt p vorhanden ist. Sie hat den Wert 0, wenn die Komponente c_{pl} für das Produkt p deselektiert wurde.

Die Anpassung der anderen Metriken zur Spanne und zum Durchschnitt bzgl. RSU, CPSU und CRSU erfolgt genauso und wird hier nicht mehr explizit dargestellt.

SU-Metriken für mehrfach verwendbare bzw. wiederverwendbare Komponenten

Die Metriken zur Variationsbreite sowie zum Durchschnitt des Servicenutzungsgrades lassen sich auch auf mehrfach verwendbare bzw. wiederverwendbare Komponenten anwenden, indem die Verwendungen des Komponententyps (d. h. die Hooks) bzgl. ihrer Einbettung in den jeweiligen Architekturmodellen angeschaut werden. Damit wird weniger eine Produktlinie, sondern vielmehr die Wiederverwendbarkeit von Komponenten in gegebenen Architekturen bewertet.

3.5.2 Variationsbreite und Durchschnitt auf Funktions- bzw. Softwarearchitekturmodelle von Produktlinien angewendet

Nach dem gleichen Schema, das bereits auf die Einzelmetriken CPSU bzw. CRSU angewendet wurde (vgl. Abschnitt 3.4), lassen sich auch die Metriken zur Bewertung der Varianz in Produktlinienarchitekturen an Funktionsmodelle und Softwarearchitekturmodelle hinsichtlich Ports bzw. Signale anpassen.

Folgende Maße können für Produktlinienarchitekturmodelle a_{pl} (sowohl für Funktions- als auch für Softwarearchitekturmodelle) berechnet werden, wobei mit s der angesetzte Schnitt zur Auswahl der zu bewertenden Komponenten angegeben wird:

SpanCPSU$_{ports}$ (a_{pl}, s)
Variationsbreite der *CPSU*s bzgl. der Ports der Komponenten

SpanCRSU$_{ports}$ (a_{pl}, s)
Variationsbreite der *CRSU*s bzgl. der Ports der Komponenten

AverageCPSU$_{ports}$ (a_{pl}, s)
Durchschnitt der *CPSU*s bzgl. der Ports der Komponenten

AverageCRSU$_{ports}$ (a_{pl}, s)
Durchschnitt der *CRSU*s bzgl. der Ports der Komponenten

SpanCPSU$_{signals}$ (a_{pl}, s)
Variationsbreite der *CPSU*s bzgl. der Signale der Komponenten

SpanCRSU$_{signals}$ (a_{pl}, s)
Variationsbreite der *CPSU*s bzgl. der Signale der Komponenten

AverageCPSU$_{signals}$ (a_{pl}, s)
Durchschnitt der *CPSU*s bzgl. der Signale der Komponenten

AverageCRSU$_{signals}$ (a_{pl}, s)
Durchschnitt der *CPSU*s bzgl. der Signale der Komponenten

Berücksichtigung von Varianz- bzw. Konfigurationsinformationen

Die Berücksichtigung der Konfiguration der optionalen Komponenten und Varianzpunktkomponenten kann analog zur Bewertung der Einzelkomponenten, wie es in Formel 3.13, S. 34 und Formel 3.14, S. 34 angegeben ist, erfolgen.

3.6 Metriken zur Kohärenz

Den Begriff der Kohärenz hat der Autor der folgenden Metrik geprägt, um die externe, nutzungsorientierte Perspektive eines Artefakts im Gegensatz zur Betrachtung der internen Struktur des Artefakts hervorzuheben [Mis00, Mis01]. Letztere Sichtweise wird häufig auch als Kohäsion bezeichnet.

Während mit den SU-Metriken (Abschnitt 3.3) gemessen wird, wieviel der Funktionalität einer Komponente überhaupt in einer Architektur genutzt wird, angegeben durch den prozentualen Anteil, wird mit Hilfe der Kohärenzmetrik gemessen, wieviel von einer Komponente jeweils von den Nutzern der Komponente verwendet wird, angegeben in Relation zur Nutzerzahl der Komponente.

Die Kohärenz $\psi(c)$ einer Komponente c wird in [Mis06] wie folgt definiert: *Anzahl der angebotenen Dienste, die von Clients verwendet werden, gemittelt über alle angeschlossenen Clients der Komponente*.[12]

Formel 3.25

$$\psi(c,a) = \frac{\sum_{w \in W(c,a)} (\#S(c,w,a) - 1)}{\#W(c,a) * (\#P(c,a) - 1)}$$

wobei:

[12] Die Kohärenzmetrik für Komponenten ist eine Anpassung der allgemeiner formulierten Metrik in [Mis00, Mis01], die »beliebige« Objektmengen bzgl. Kohärenz bewerten. Für die Anwendung der Kohärenzmetrik auf Komponenten werden in [Mis06] diese Objektmengen als die angebotenen bzw. benötigten Dienste der Komponenten interpretiert.

C(a) ist die Menge aller Komponenten in einem Produktlinien- oder Produktarchitekturmodell a.

c ist eine Komponente aus der Menge $C(a)$ des Architekturmodells a.

P(c,a) = {s1, s2, ...} ist die Menge der von der Komponente c angebotenen Dienste im Architekturmodell a.

W(c, a) = {w1, w2, ...} ist die Menge der Komponenten $w_i \in C(a)$, die Dienste der Komponente c nutzen bzw. erfordern, d. h., $W(c, a)$ ist die Menge der Clients von c im Architekturmodell a.

S(c,w,a) ist die Menge der Dienste, die von c angeboten und von w in a tatsächlich genutzt werden.

3.6.1 Die Anpassung der Metriken zur Kohärenz bei Funktionen und Softwarekomponenten

Die Kohärenzmetrik lässt sich auf Funktions- als auch auf Softwarekomponenten wie folgt anwenden:

- Die Bewertung der Kohärenz einer Komponente in einer Architektur kann bzgl. der angebotenen und bzgl. der geforderten Dienste erfolgen.
- Die Kohärenzbewertung kann auf Basis der Betrachtung von Ports und / oder von Signalen / Operationen erfolgen.
- Die Metrik kann sowohl auf ein Produktlinienarchitekturmodell als auch auf ein Produktarchitekturmodell angewendet werden.
- Analog zu den Nutzungs- und Bereitstellungsgraden könnten aufbauend auf dem Kohärenzmaß einer Komponente zusammenfassende Metriken definiert werden: bspw. die Spanne und der Durchschnitt der Kohärenzwerte einer Komponente über alle Produktmodelle sowie der Durchschnittswert der Kohärenzwerte aller Komponenten einer Architektur.
- Ebenso ist die Konfigurationsinformation für variierende Komponenten zu berücksichtigen, sollten nicht nur obligatorische Komponenten bewertet werden.
- Ebenfalls ist die Metrik nur auf flachgeklopfte Architekturmodelle, die entsprechend eines anzugebenden Schnitts abgeleitet sind, anzuwenden, weil die Metrik nicht für hierarchisch aufgebaute Komponenten gemäß der Definition der VEIA-Architekturmodelle ausgerichtet ist.

Im Folgenden die beispielhafte Anwendung der Kohärenzmetrik auf Ausgangsports von Funktionen – analog zur Anpassung der SU-Metriken:

Formel 3.26

$$\psi_{pout}(f,a,s) = \frac{\sum_{w \in f^{\star}.connectedFunctions_{pout}} (\#(f^{\star}.outputPorts_{UsagePerClient}(w)) - 1)}{\#f^{\star}.connectedFunctions_{pout} * (\#f^{\star}.outputPorts_{total} - 1)}$$

wobei:

$f^{\star}.connectedFunctions_{pout}$ ist die Menge der Clientfunktionen, die an den Ausgangsports der Funktion f gemäß des Schnitts s angeschlossen sind.

$f^{\star}.outputPorts_{UsagePerClient}(w)$ ist die Menge der Ausgangsports der Funktion $f^{\star}$, an denen die Clientfunktion w angeschlossen ist. Im Allgemeinen kann eine Funktion über mehrere Ports mit einer anderen Funktion kommunizieren.

Siehe hierzu auch die in Kapitel 4 skizzierten Datenmodelle für Funktions- und Softwarearchitekturmodelle.

Anmerkungen

Die Definition der Kohärenzberechnung ist nicht anwendbar bei Komponenten mit nur einem angebotenen bzw. geforderten Dienst, weil die Anzahl der Dienste um eins minimiert in die Formel eingeht, so dass in diesem Fall eine Division durch Null vorliegt.

Die Begründung liegt in der Definition der Kohärenz in [Mis06]: Die Kohärenz ist 0 (kleinster möglicher Wert), wenn jeder Client nur einen Dienst der betrachteten Komponente nutzt bzw. anbietet. Die Kohärenz ist 1 (größter möglicher Wert), wenn jeder Client alle angebotenen (bzw. geforderten) Dienste der betrachteten Komponente nutzt (bzw. anbietet). Die Kohärenz ist undefiniert, wenn die betrachtete Komponente nur einen Dienst anbietet bzw. erfordert: *"In fact, in such cases it would not make sense to calculate coherence at all."*

Im Zuge der Anpassung der Metrik an die VEIA-Artefakte und die Interpretation des Servicebegriffs als Ports bzw. Signale erscheint es jedoch als sinnvoll, die Metrik so anzupassen, dass sie Aussagen für Komponenten treffen kann, die nur einen einzelnen Port haben bzw. nur ein einzelnes Signal empfangen oder senden, siehe Formel 3.27. Zumindest gibt es in der CBS-Beispielmodellierung in Kapitel 5 etliche Funktionen, die sonst aus der Bewertung herausfallen würden (vgl. bspw. die

Werte für die Spalte psi_{pout} bzw. psi_{pin} in Tabelle 1).

Formel 3.27

$$\psi'(c,a) = \frac{\sum_{w \in W(c,a)} \#S(c,w,a)}{\#W(c,a) * \#P(c,a)}$$

Die in Kapitel 5 angegebenen Werte für die ψ'-Maße für Ports und Signale basieren auf der in Formel 3.27 angegebenen Anpassung von Formel 3.25.

4 Metamodell für VEIA-Produktlinienarchitekturmodelle als Berechnungsgrundlage für die Metriken

Die Metriken, die in Kapitel 3 diskutiert wurden, basieren auf mehreren Modellen:

- Ausgangspunkt im VEIA-Kontext ist ein Produktlinienarchitekturmodell.
- Es ist der Schnitt auf dem Produktlinienarchitekturmodell zu bilden.
- Es sind die Produktmodelle zur Produktlinienarchitektur (gemäß angegebener Konfiguration) zu bilden.
- Der Schnitt auf dem Produktlinienarchitekturmodell ist auch auf die Produktarchitekturmodelle anzuwenden.
- Die Konfigurationsinformation für jede Komponente ist vorzuhalten.

Sowohl die Spanne als auch der Durchschnitt fassen die Werte, die bei der Betrachtung einer Komponente in den verschiedenen Produktmodellen ermittelt werden, zusammen.

Das Metamodell für VEIA-Architekturmodelle zur Darstellung von Produktlinien auf Architekturebene reflektiert diese verschiedenen Sichten.

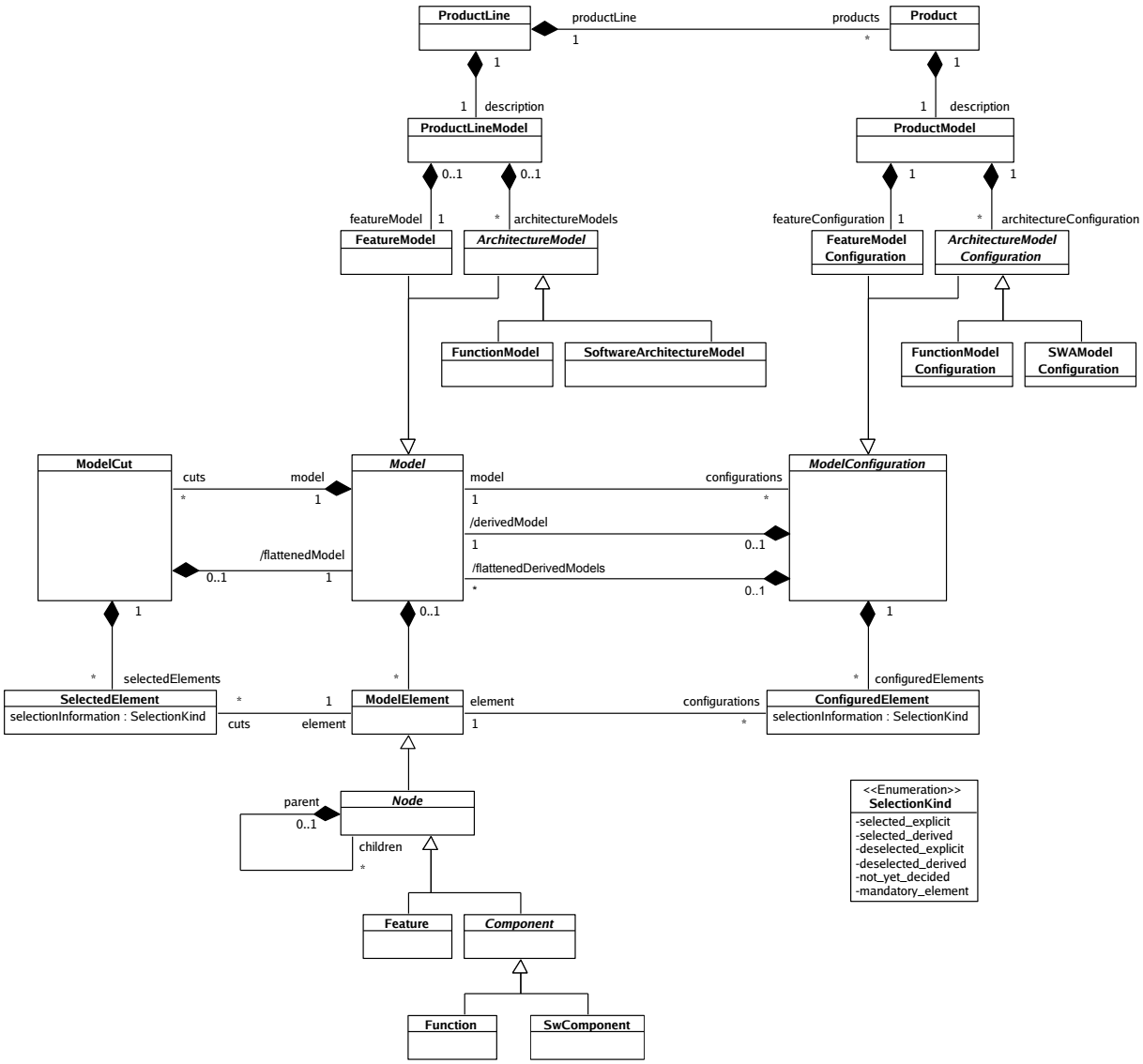
4.1 Metamodell zur Repräsentation von Produktlinien und ihren Produkten

Die relevanten Aspekte für die Metrikanbindung an die VEIA-Metamodelle sind in Abbildung 5 angegeben. Das hier abgebildete Klassendiagramm stellt nicht exakt das VEIA-Datenmodell dar, sondern ist eine Abstraktion davon. Die Teile, die spezifisch für Funktions- und Softwarearchitekturmodelle sind, sind in Abbildung 7 bzw. Abbildung 8 dargestellt.

Produktlinien- vs. Produktarchitekturmodell

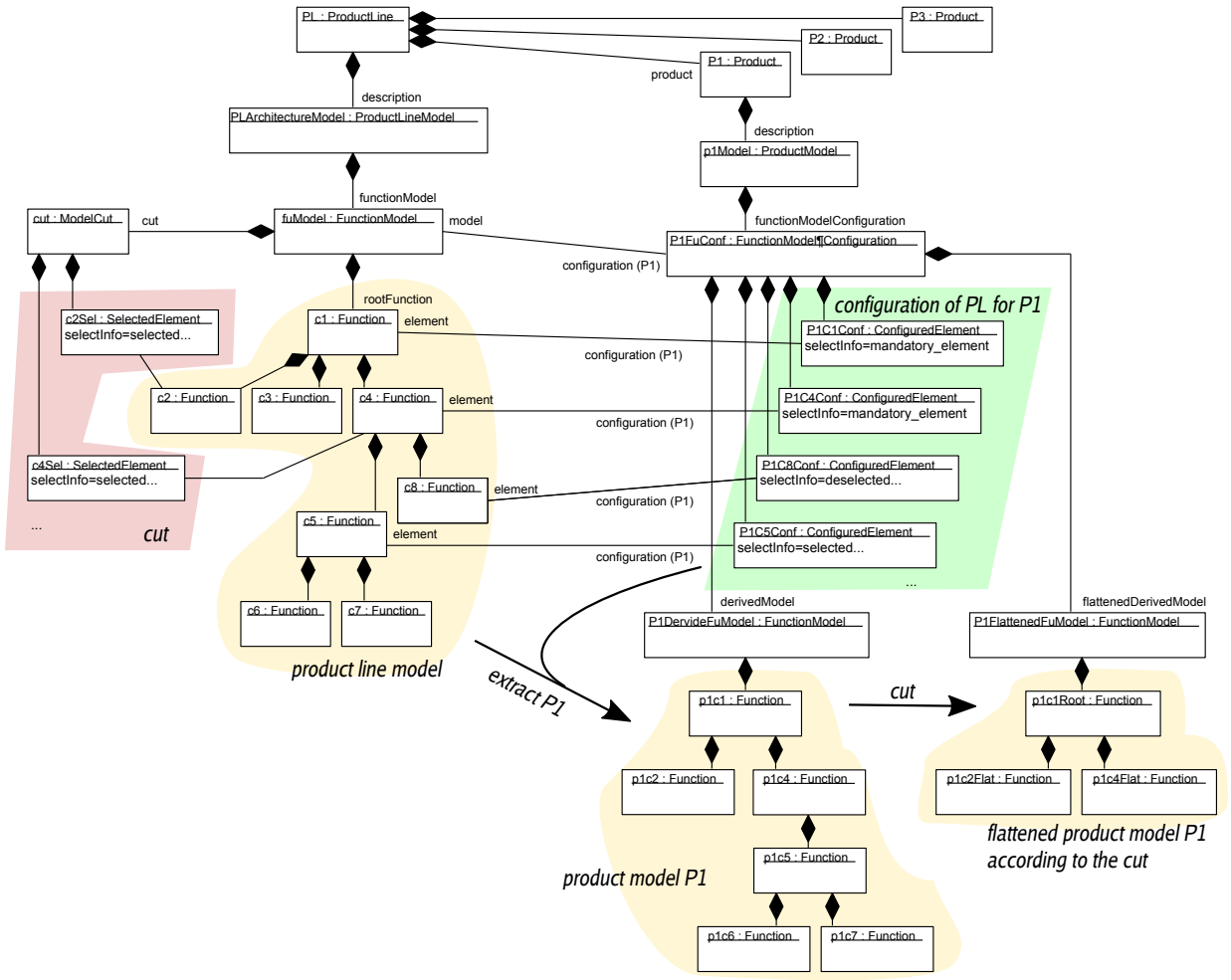
Um die Problematik der unterschiedlichen Modelle und abgeleiteten Modelle – Produktlinienmodell, Konfiguration des Produktlinienmodells, abgeleitetes Produktmodell gemäß Konfiguration, Schnitt durch und Flachklopfen eines Modells – zu veranschaulichen, ist in Abbildung 6 ein Objektdiagramm konform zum Metamodell in Abbildung 5 dargestellt, das die abstrakten Beispiele in Abbildung 2,

Abbildung 5 Metamodell zur Repräsentation von Produktlinien und ihren Produkten – für die konzeptionelle Metrikanbindung (vereinfacht).



S. 12 (Produktlinien- vs. Produktmodelle) sowie Abbildung 4, S. 22 (Schnitt) verdeutlichen soll.

Abbildung 6 Objektdiagramm zum Metamodell (Ausschnitt).

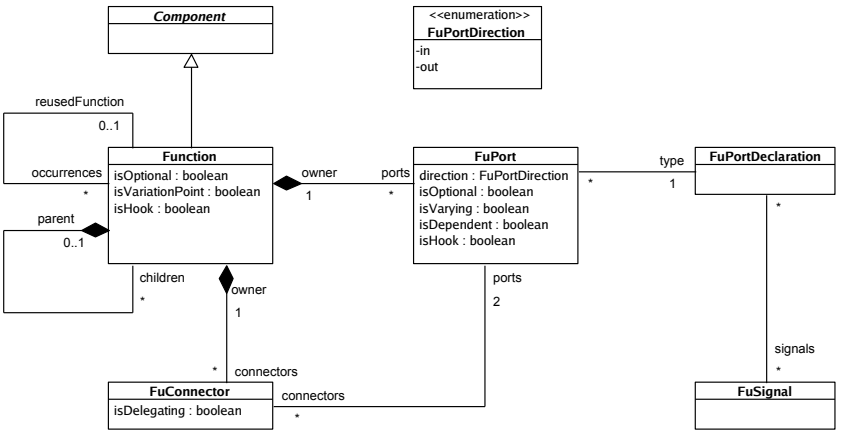


Ein aus dem Produktlinienarchitekturmodell extrahiertes Produktarchitekturmodell (bspw. das Funktionsnetz für das Produkt *P1*) ist selbst wieder konform zum entsprechenden Metamodell für Architekturmodelle (Funktionsnetz, Abschnitt 4.2 bzw. Softwarearchitektur, Abschnitt 4.3). Ebenso verhält es sich mit einem gemäß eines Schnitts flachgeklopften Architekturmodell.

4.2 Metamodell für Funktionsnetze (vereinfacht)

Das (vereinfachte) Metamodell für Funktionsnetze in VEIA ist in Abbildung 7 angegeben. Die zum Klassendiagramm konformen, nachfolgend angegebenen *Object-Z*-Spezifikationen[13] der Metaklassen verdeutlichen diejenigen Anteile an Informationen, die für die Anwendung der in diesem Bericht vorgestellten Metriken notwendig sind und sich aus den primären Strukturinformationen herleiten lassen.[14] Dabei beziehen sich sämtliche abgeleiteten Attribute auf ein spezifisches (Produktlinien- bzw. Produkt-) Architekturmodell (Attribut $model$ der Metaklasse $Function$).

Abbildung 7 Metamodell für VEIA-Funktionsnetze (vereinfacht).



FunctionModel

Model

$root : Function_{\copyright}$;

[13] Siehe bspw. [DR00].

[14] Attribute nach dem Δ-Zeichen im Deklarationsteil des Zustandsschemas einer *Object-Z*-Klasse sind berechenbare, abgeleitete Attribute. Ihre Berechnung ist hier nicht angegeben. Ebenso wurde auf die Angabe aller Constraints, die in den VEIA-Datenmodellen gelten, in diesem Bericht verzichtet.

Function

Component

$model : FunctionModel;$
$parent : \mathbb{F}_{0..1}\ Function;$
$children : \mathbb{F}\ Function_{\copyright};$
$ports : \mathbb{F}\ FuPort_{\copyright};$
$connectors : \mathbb{F}\ FuConnector;$
$isOptional, isVariationPoint : \mathbb{B};$
$isHook : \mathbb{B};$
$reusedFunction : \mathbb{F}_{0..1}\ Function;$
$occurrences : \mathbb{F}\ Function$
Δ
$inputPorts_{total}, inputPorts_{used} : \mathbb{F}\ FuPort;$
$outputPorts_{total}, outputPorts_{used} : \mathbb{F}\ FuPort;$
$inputSignals_{total}, inputSignals_{used} : \mathbb{F}\ FuSignals;$
$outputSignals_{total}, outputSignals_{used} : \mathbb{F}\ FuSignals;$
$connectedFunctions_{pin}, connectedFunctions_{pout} : \mathbb{F}\ Function;$
$inputPorts_{UsagePerClient} : \mathbb{F}\ Function \nrightarrow \mathbb{F}\ FuPort;$
$outputPorts_{UsagePerClient} : \mathbb{F}\ Function \nrightarrow \mathbb{F}\ FuPort;$
$connectedFunctions_{sigin}, connectedFunctions_{sigout} : \mathbb{F}\ Function;$
$inputSignals_{ProvisionPerServer} : \mathbb{F}\ Function \nrightarrow \mathbb{F}\ FuSignal;$
$outputSignals_{UsagePerClient} : \mathbb{F}\ Function \nrightarrow \mathbb{F}\ FuSignal$

$\mathrm{dom}\ inputPorts_{ProvisionPerServer} = connectedFunctions_{pin}$
$\mathrm{dom}\ outputPorts_{UsagePerClient} = connectedFunctions_{pout}$
$\mathrm{dom}\ inputSignals_{ProvisionPerServer} = connectedFunctions_{sigin}$
$\mathrm{dom}\ outputSignals_{UsagePerClient} = connectedFunctions_{sigout}$

$FuPortDirection \mathrel{\widehat{=}} in \mid out$

FuPort

ModelElement

$owner : Function;$
$direction : FuPortDirection;$
$type : FuPortDeclaration;$
$functionConnectors : \mathbb{F}\ FuConnector;$
$isOptional, isDependent, isVarying : \mathbb{B};$
$isHook : \mathbb{B}$

FuPortDeclaration

ModelElement

$signals : \mathbb{F}\ FuSignal;$

FuSignal

ModelElement

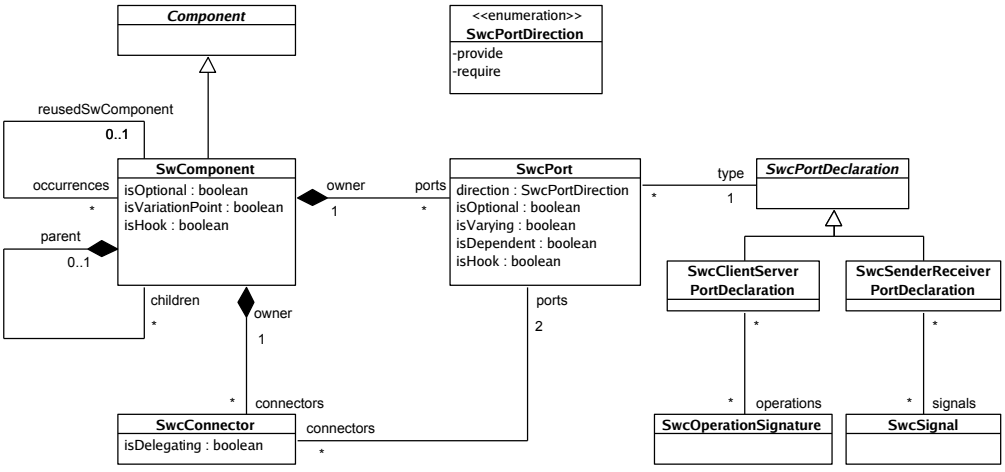
FuConnector
ModelElement

$owner : Function;$
$ports : \mathbb{F}_{2..2}\ FuPort;$
$isDelegating : \mathbb{B}$

4.3 Metamodell für Softwarearchitekturmodelle (vereinfacht)

Das (vereinfachte) Metamodell für Softwarearchitekturmodelle in VEIA ist in Abbildung 8 angegeben. Analog zum Metamodell für Funktionsnetze sind nachfolgend die zum Klassendiagramm konformen *Object-Z*-Spezifikationen der Metaklassen angegeben.

Abbildung 8 Metamodell für VEIA-Softwarearchitekturmodelle (vereinfacht).



SoftwareArchitecureModel
Model

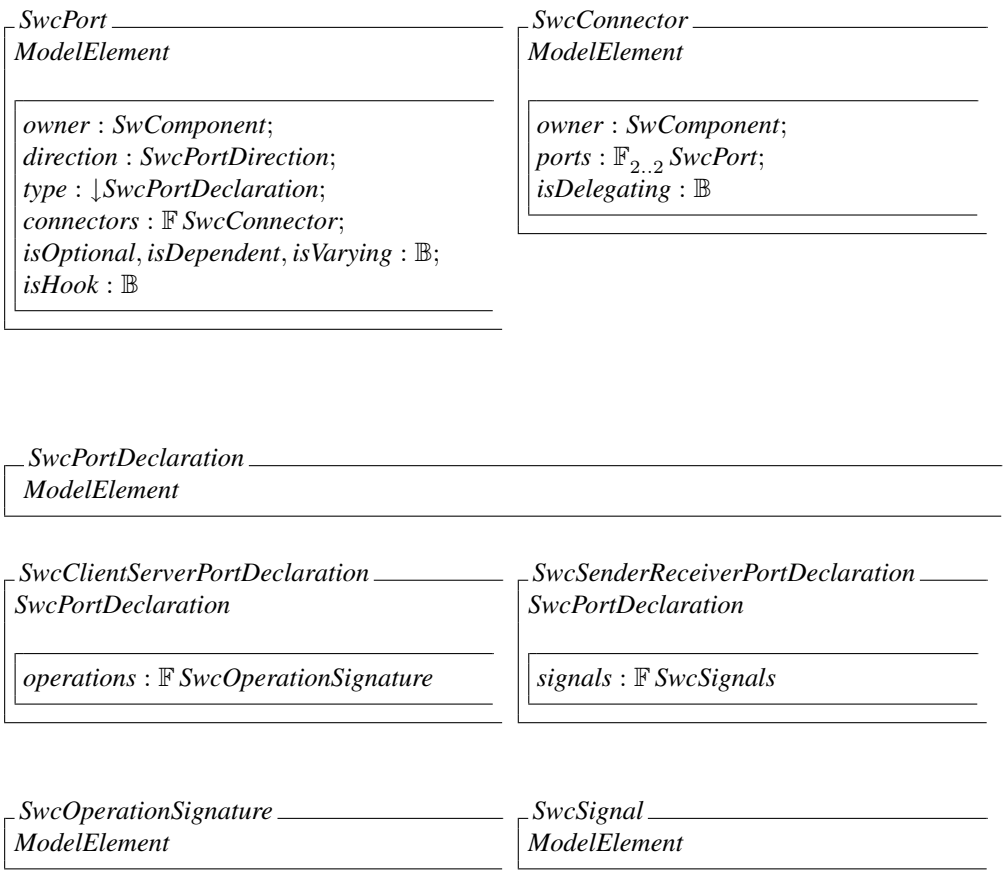
$root : SwComponent_{©}$

SwComponent

Component

$$
\begin{array}{l}
model : SoftwareArchitecureModel; \\
parent : \mathbb{F}_{0..1}\, SwComponent; \\
children : \mathbb{F}\, SwComponent_{\copyright}; \\
ports : \mathbb{F}\, SwcPort_{\copyright}; \\
connectors : \mathbb{F}\, SwcConnector; \\
isOptional, isVariationPoint : \mathbb{B}; \\
isHook : \mathbb{B}; \\
reusedSwComponent : \mathbb{F}_{0..1}\, SwComponent; \\
occurrences : \mathbb{F}\, SwComponent \\
\Delta \\
requiredPorts_{total}, requiredPorts_{used} : \mathbb{F}\, SwcPort; \\
providedPorts_{total}, providedPorts_{used} : \mathbb{F}\, SwcPort; \\
inputSignals_{total}, inputSignals_{used} : \mathbb{F}\, SwcSignal; \\
outpoutSignals_{total}, outputSignals_{used} : \mathbb{F}\, SwcSignal; \\
providedOperations_{total}, providedOperations_{used} : \mathbb{F}\, SwcOperationSignature; \\
requiredOperations_{total}, requiredOperations_{used} : \mathbb{F}\, SwcOperationSignature; \\
connectedComponents_{pprov}, connectedComponents_{preq} : \mathbb{F}\, SwComponent; \\
requiredPorts_{ProvisionPerServer} : \mathbb{F}\, SwComponent \nrightarrow \mathbb{F}\, SwcPort; \\
providedPorts_{UsagePerClient} : \mathbb{F}\, SwComponent \nrightarrow \mathbb{F}\, SwcPort; \\
connectedComponents_{sigin}, connectedComponents_{sigout} : \mathbb{F}\, SwComponent; \\
inputSignals_{ProvisionPerServer} : \mathbb{F}\, SwComponent \nrightarrow \mathbb{F}\, SwcSignal; \\
outputSignals_{UsagePerClient} : \mathbb{F}\, SwComponent \nrightarrow \mathbb{F}\, SwcSignal; \\
connectedComponents_{opreq}, connectedComponents_{opprov} : \mathbb{F}\, SwComponent; \\
requiredOperations_{ProvisionPerServer} : \mathbb{F}\, SwComponent \nrightarrow \mathbb{F}\, SwcOperationSignature; \\
providedOperations_{UsagePerClient} : \mathbb{F}\, SwComponent \nrightarrow \mathbb{F}\, SwcOperationSignature \\
\hline
\mathrm{dom}\, requiredPorts_{ProvisionPerServer} = connectedComponents_{preq} \\
\mathrm{dom}\, providedPorts_{UsagePerClient} = connectedComponents_{pprov} \\
\mathrm{dom}\, inputSignals_{ProvisionPerServer} = connectedComponents_{sigin} \\
\mathrm{dom}\, outputSignals_{UsagePerClient} = connectedComponents_{sigout} \\
\mathrm{dom}\, requiredOperations_{ProvisionPerServer} = connectedComponents_{opreq} \\
\mathrm{dom}\, providedOperations_{UsagePerClient} = connectedComponents_{opprov}
\end{array}
$$

$$SwcPortDirection \mathrel{\widehat{=}} provide \mid require$$



4.4 Der Schnitt durch ein Architekturmodell

Die Menge der Komponenten eines Schnitts durch ein Architekturmodell (vgl. Abschnitt 3.2.2) lässt sich wie folgt bestimmen:

1 Die Menge der auswählbaren Komponenten ist initial die Menge aller Komponenten eines Architekturmodells.

2 Wähle eine Komponente aus der Menge an auswählbaren Komponenten aus, sie gehört zum Schnitt.

3 Alle in der Hierarchie höher liegenden Komponenten (alle direkten und indirekten Elternkomponenten der ausgewählten Komponente) sind abgewählt und gehören nicht zum Schnitt. Sie werden aus der Menge der auswählbaren Komponenten entfernt.

4 Alle in der Hierarchie tiefer liegenden Komponenten (alle direkten und indirekten Kinderkomponenten der ausgewählten Komponente) sind abgewählt und gehören nicht zum Schnitt. Sie werden aus der Menge der auswählbaren Komponenten entfernt.

5 Weiter mit Schritt 2, bis die Menge der auswählbaren Komponenten leer ist.

Die Varianz kann beim Setzen des Schnitts ignoriert werden, entweder es wird der Varianzpunkt berücksichtigt, dann werden alle Alternativen ignoriert, oder es werden die Alternativen bzw. ihre Kinderkomponenten gezählt. Im letzteren Fall erhöht sich im bewerteten Produktlinienmodell der Nutzungsgrad für eine Komponente, die mit diesen Alternativen in Beziehung steht, da nun mehrere Komponenten (potenziell) angeschlossen sind.

Bei wiederverwendbaren Funktionen und Hooks sollte der Schnitt konsistent bzgl. der Vergleichbarkeit erfolgen: Entweder kann der Schnitt bei den Hooks angesetzt werden, dann werden der wiederverwendete Funktionstyp und seine Kinderkomponenten ignoriert. Oder der Schnitt kann tiefer angesetzt werden, dann sollte für jeden Hook derselbe Schnitt gelten (d. h., die Wiederverwendung von Funktionen erwirkt eine »Wiederverwendung« des Schnitts innerhalb der wiederverwendeten Funktion), um die Vergleichbarkeit der Werte sicherzustellen.

Wird gemäß eines so durchgeführten Schnitts ein flaches Architekturmodell aus dem ursprünglichen Architekturmodell erzeugt, so ist es wieder ein zum Metamodell für Funktions- bzw. Softwarearchitekturmodelle konformes Architekturmodell, nur dass keine hierarchische Komposition (außer die Kapsel für die Wurzelkomponente) angewendet wurde, d. h., alle Komponenten sind atomar.

5 Anwendung der Metriken auf das Fallbeispiel »*Condition-Based Service*«

Im Folgenden werden die zuvor vorgestellten Kohäsions- und Kohärenzmetriken auf die VEIA-Fallstudie »Condition-Based Service« (CBS) angewendet. Die Fallstudie ist bereits in [GKM07c] vorgestellt worden. Aus Gründen der Übersichtlichkeit wird nur ein Auszug der Umfänge des Fallbeispiels betrachtet.

5.1 Funktionsnetz

Das zur Bewertung verwendete CBS-Funktionsmodell ist in Abbildung 9 angegeben.

Die zu den Ports zugehörigen Portdeklarationen sind in Abbildung 10 und Abbildung 11 dargestellt. Die Zuordnung der Portdeklaration zu den Ports ergibt sich in der hier angegebenen Modellierung meist aus der Namensähnlichkeit, d. h., die Typisierung der Ports ist der Übersichtlichkeit wegen in die Portbenennung eingeflossen – mit den folgenden Ausnahmen:

- Die Ports `poutAvailability` der Funktionen `CbsWdBreakPadsAdaptive`, `CbsWdMotorOilAdaptive`, `CbsWdSparkPlugs`, `CbsWdParticleFilter` sowie die entsprechenden Ports der Funktion `CbsComputeOriginalServiceDate` sind in der hier angegebenen Beispielmodellierung vom Typ `ICbsAvailabilityDistanceTime`.
- Der Port `poutAvailability` von der Funktion `CbsDmVehicleCheck` und der Port `pinAvailabilityVc` von der Funktion `CbsComputeOriginalServiceDate` sind vom Typ `ICbsAvailabilityTime`.
- Die Ports `pinKm` sind vom Typ `IMileage`.
- Die Ports `pinDoService` der Funktionen `CbsWd...` und `CbsDmVehicleCheck` sowie die Ports `poutServ...` der Funktion `CbsReset` sind vom Typ `ICbsDoServiceInternal`.

Abbildung 9 Funktionsnetz für CBS-Produktlinie.

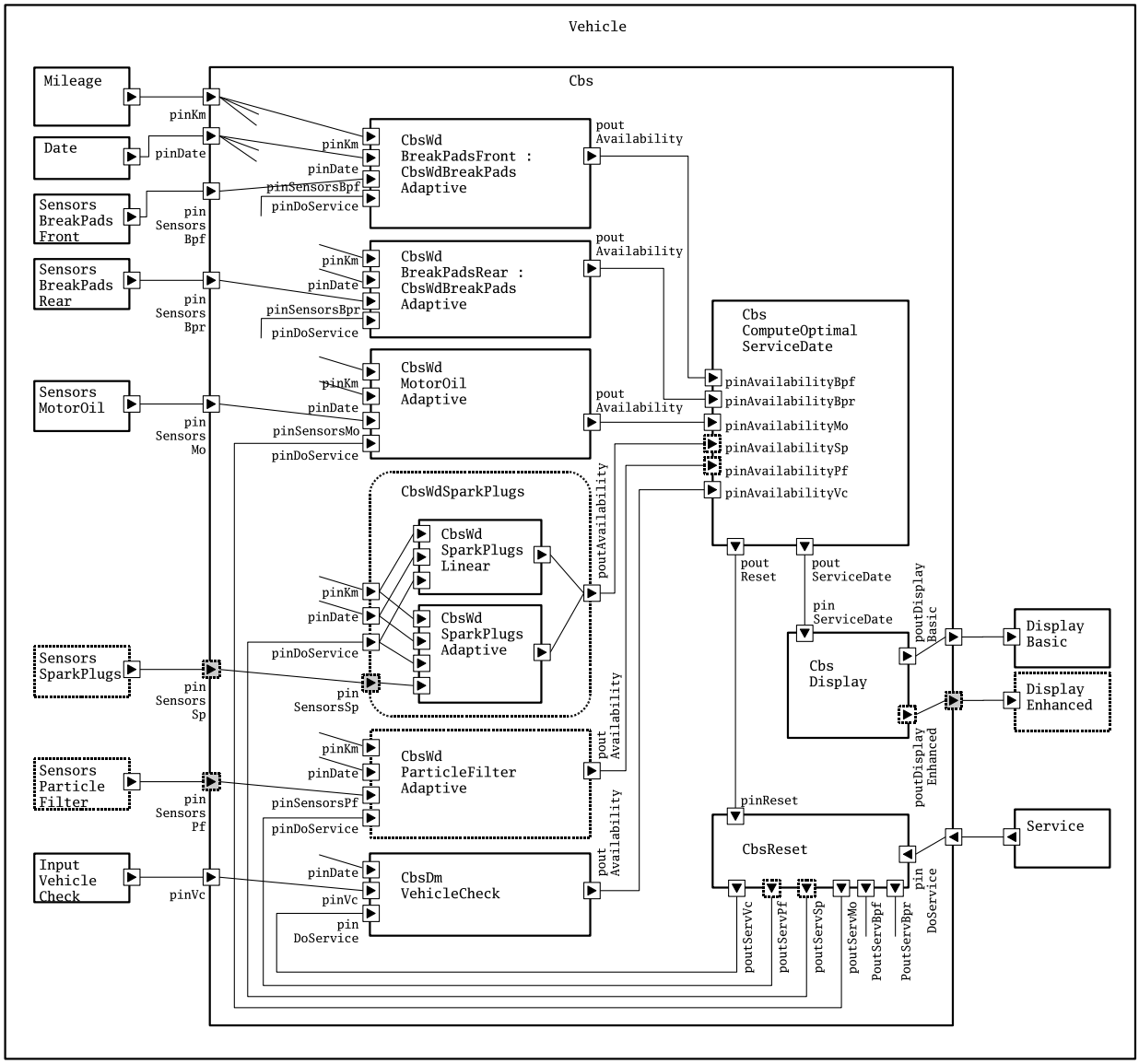


Abbildung 10 Portdeklaration für die Funktion CBS mit ihrer Umgebung.

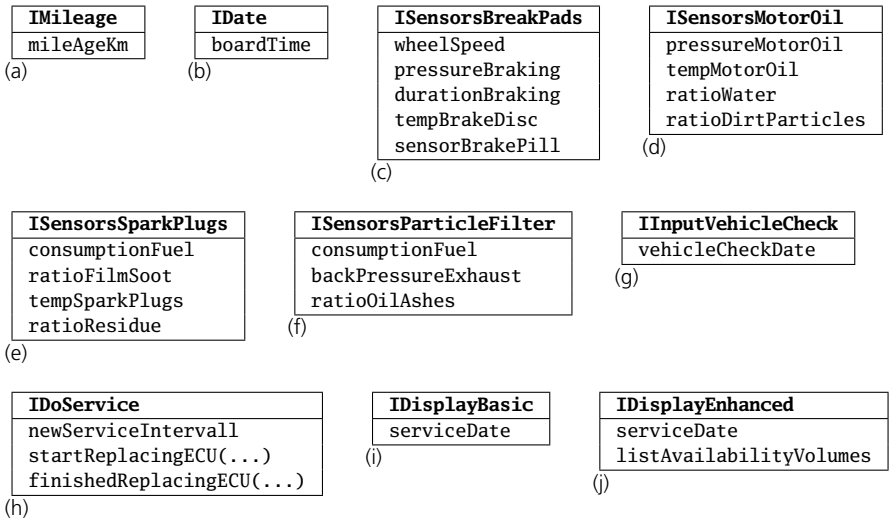


Abbildung 11 Portdeklarationen für die internen Funktion von CBS.

5.2 Schnitt

Um die vorgestellten Metriken anwenden zu können, muss die hierarchische Komposition im Funktionsnetz ignoriert werden. Hierzu wird der in Abbildung 12 dargestellte Schnitt angesetzt. Das entsprechend flachgeklopfte Funktionsmodell ist in Abbildung 13 angegeben.

Abbildung 12 Funktionshierarchie für CBS-Produktlinie und ein Schnitt durch diese Hierarchie.

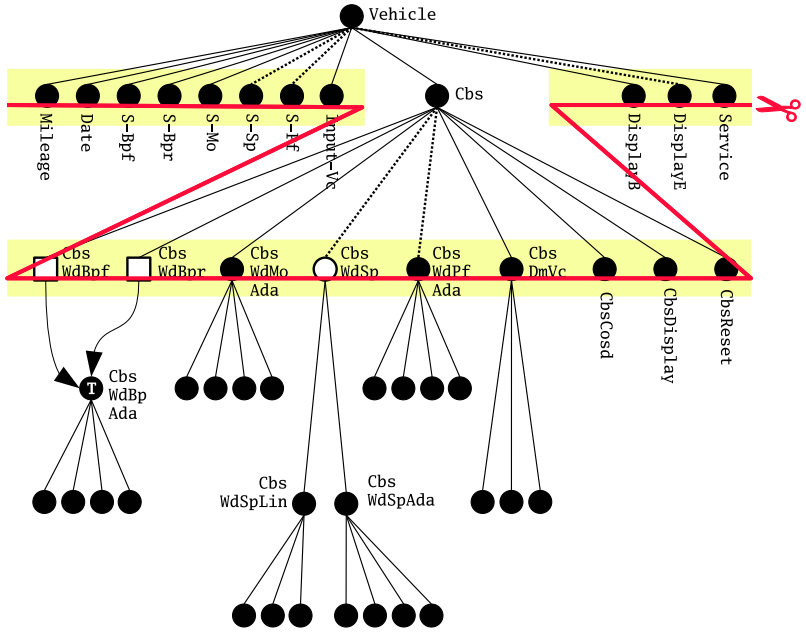


Abbildung 14 ist inhaltlich identisch mit Abbildung 13, jedoch ist zusätzlich die Anzahl der über einen Port fließenden Signale angegeben, um die Zählung der Strukturinformationen zu vereinfachen bzw. nachvollziehbarer zu machen. Sie ergibt sich aus der Zählung der Signale der entsprechenden Portdeklaration.

Abbildung 13 Flachgeklopftes Funktionsnetz für CBS-Produktlinie bzgl. des Schnitts.

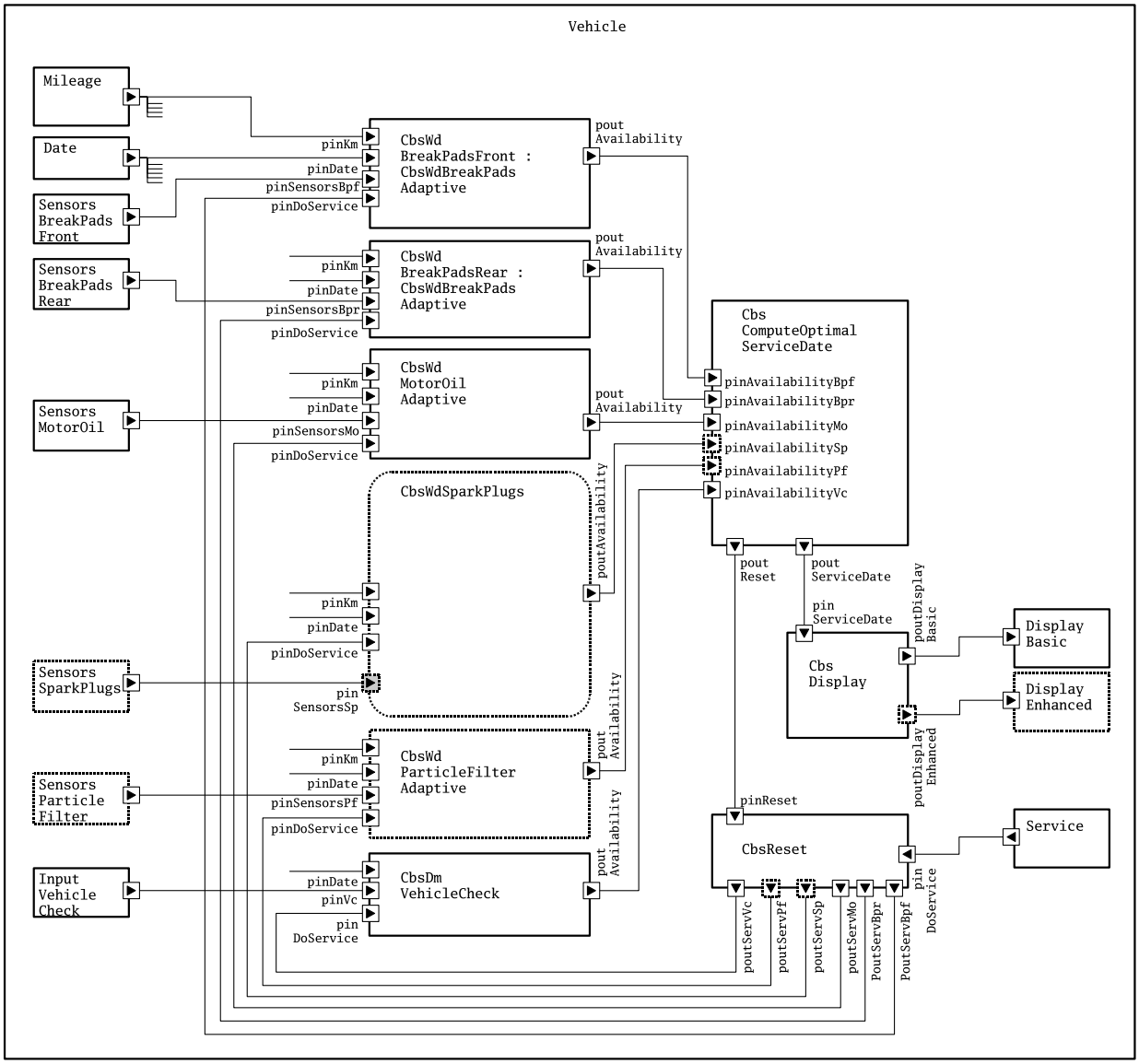
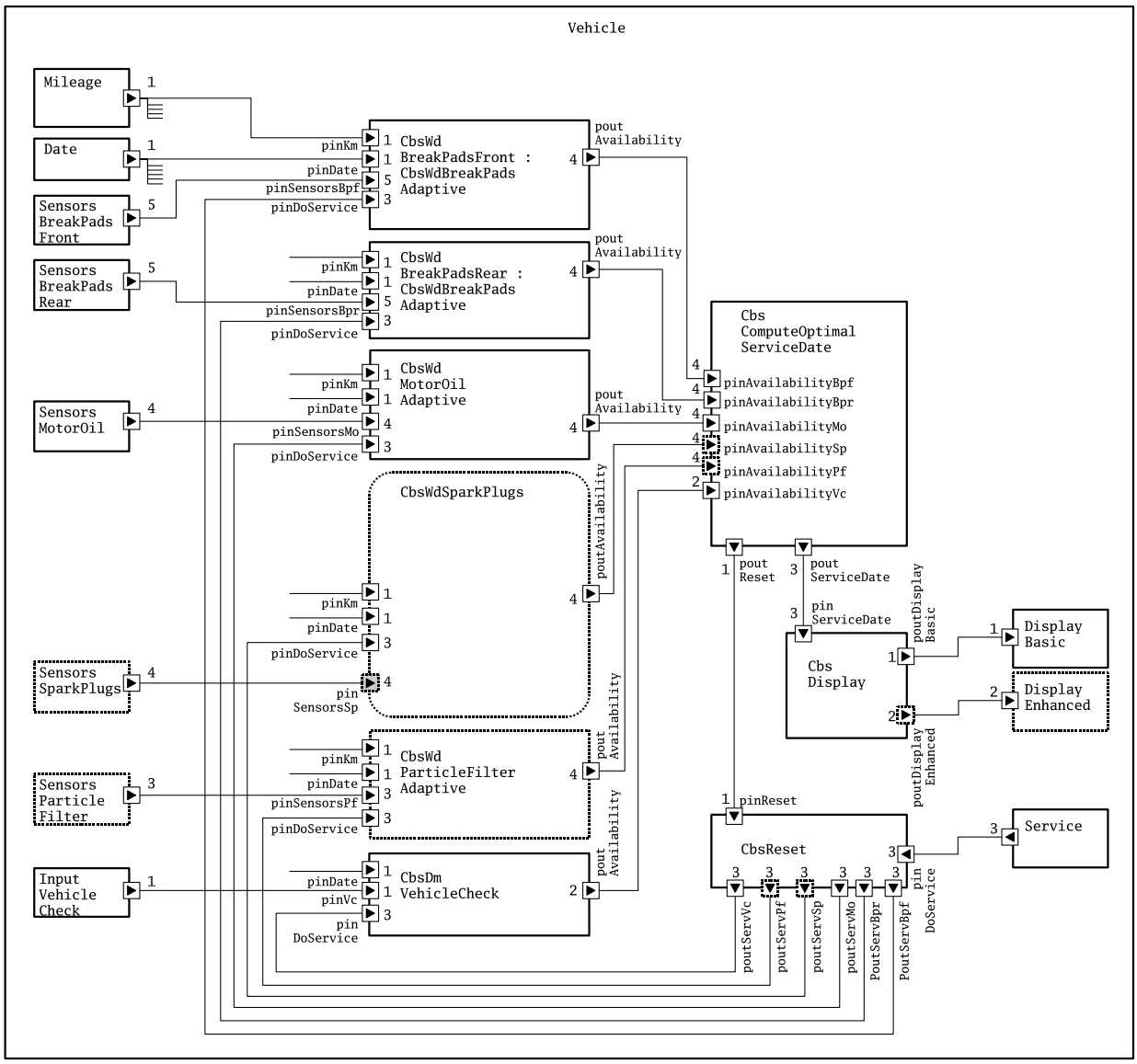


Abbildung 14 Flachgeklopftes Funktionsnetz für CBS-Produktlinie bzgl. des Schnitts plus Angabe der Anzahl der Signale, die über den Port fließen.



5.3 Produkte

Wir gehen beispielhaft von drei Produkten aus. Sie sind durch folgende Merkmale ausgezeichnet:

P1: Ottomotor, adaptive Zündkerzenverschleißerfassung, keine erweiterte Anzeige

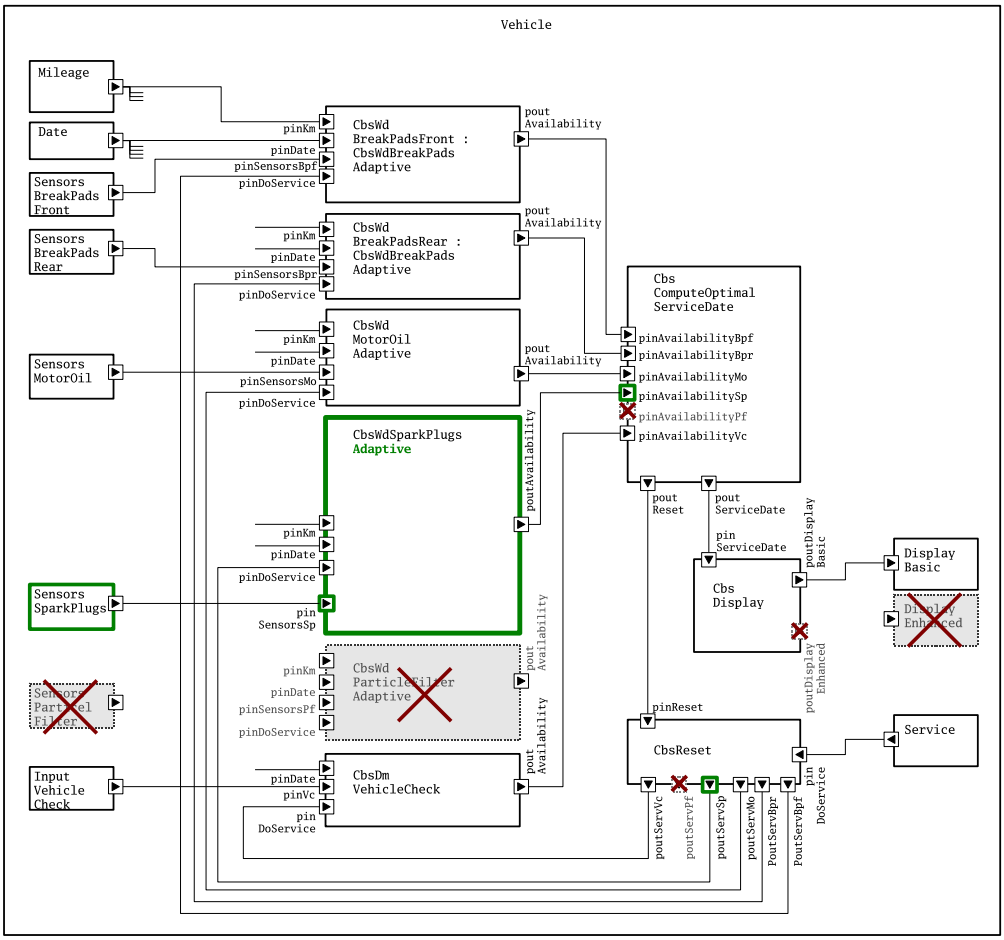
P2: Ottomotor, lineare Zündkerzenverschleißerfassung, keine erweiterte Anzeige

P3: Dieselmotor, adaptive Partikelfilterverschleißerfassung, keine erweiterte Anzeige

5.3.1 Produkt P1 der CBS-Produktlinie bzgl. des Schnitts

Das flachgeklopfte Funktionsnetz für das Produkt *P1* – gemäß des in Abschnitt 5.2 angegebenen Schnittes – ist in Abbildung 15 angegeben. Die Funktionsnetze für die anderen Produkte sehen analog dazu aus.

Abbildung 15 Flach geklopftes Funktionsnetz für Produkt P1 der CBS-Produktlinie.



5.4 Anwendung der SU-Metriken auf die Beispielmodellierung unter Berücksichtigung der Ports als Dienstbegriff

Im Folgenden werden die SU-Metriken sowie die Kohärenzmetrik auf die Funktionsmodellierung der CBS-Produktlinie angewendet. Grundlage der Berechnung ist der in Abbildung 12 dargestellte Schnitt sowie die Interpretation des Servicebegriffs als Ports von Funktionen.

5.4.1 Anwendung der Metriken auf das Produktlinienmodell (Funktionsnetz / Ports)

In Tabelle 1 ist die Anwendung der SU-Metriken sowie der Kohärenzmetrik auf das Produktlinienfunktionsmodell gemäß des gewählten Schnitts (vgl. flachgeklopftes Funktionsmodell Abbildung 13) tabellarisch dargestellt. Die in Tabelle 1 durchgeführten Berechnungen verwenden auch Werte aus Tabelle 2. Die notwendigen Eingangswerte für die Kohärenzmetrik sind in Tabelle 3 aufgelistet. Die Metriken wurden hier beispielhaft auf Ports angewendet.

Tabelle 1 Anwendung auf CBS-PL-Funktionsnetz bzgl. Ports.

Funktion	$\#pout_{total}$	$\#pout_{used}$	$\#pin_{total}$	$\#pin_{used}$	$\#C_{pout}$	$\#C_{pin}$	PSU_{ports}	RSU_{ports}	$SpanPSU_{ports}$ ($SpanPSU'_{ports}$)	$SpanRSU_{ports}$ ($SpanRSU'_{ports}$)	$AvPSU_{ports}$ ($AvPSU'_{ports}$)	$AvRSU_{ports}$ ($AvRSU'_{ports}$)	ψ_{pout} (ψ'_{pout})	ψ_{pin} (ψ'_{pin})
Mileage	1	1	0	0	5	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
Date	1	1	0	0	6	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
S-BreakPadsFront	1	1	0	0	1	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
S-BreakPadsRear	1	1	0	0	1	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
S-MotorOil	1	1	0	0	1	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
S-SparkPlugs	1	1	0	0	1	0	1	1*	2x0, 1x1 (1x1)	3x1* (1x1*)	0.33 (1)	1*	– (1)	– (–)
S-ParticleFilter	1	1	0	0	1	0	1	1*	2x0, 1x1 (1x1)	3x1* (1x1*)	0.33 (1)	1*	– (1)	– (–)
InputVehicle-Check	1	1	0	0	1	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
CbsWdBreakPads-Front	1	1	4	4	1	4	1	1	3x1	3x1	1	1	– (1)	0 (0,25)
CbsWdBreakPads-Rear	1	1	4	4	1	4	1	1	3x1	3x1	1	1	– (1)	0 (0,25)
CbsWdMotorOil	1	1	4	4	1	4	1	1	3x1	3x1	1	1	– (1)	0 (0,25)
CbsWdSparkPlugs	1	1	4	4	1	4	1	1	1x0, 2x1 (2x1)	1x0, 2x1 (2x1)	0.67 (1)	0.67 (1)	– (1)	0 (0,25)

(Weiter auf der nächsten Seite)

Tabelle 1 Anwendung auf CBS-PL-Funktionsnetz bzgl. Ports (Fortsetzung).

Funktion	$\#pout_{total}$	$\#pout_{used}$	$\#pin_{total}$	$\#pin_{used}$	$\#C_{pout}$	$\#C_{pin}$	PSU_{ports}	RSU_{ports}	$SpanPSU_{ports}$ ($SpanPSU'_{ports}$)	$SpanRSU_{ports}$ ($SpanRSU'_{ports}$)	$AvPSU_{ports}$ ($AvPSU'_{ports}$)	$AvRSU_{ports}$ ($AvRSU'_{ports}$)	ψ_{pout} (ψ'_{pout})	ψ_{pin} (ψ'_{pin})
`CbsWdParticle-Filter`	1	1	4	4	1	4	1	1	2x0, 1x1 (1x1)	2x0, 1x1 (1x1)	0.33 (1)	0.33 (1)	– (1)	0 (0,25)
`CbsDmVehicle-Check`	1	1	3	3	1	3	1	1	3x1	3x1	1	1	– (1)	0 (0,33)
`CbsComputeOpti-malServiceDate`	2	2	6	6	2	6	1	1	3x1	3x1	1	1	0 (0,5)	0 (0,17)
`CbsDisplay`	2	2	1	1	2	1	1	1	3x1	3x1	1	1	0 (0,5)	– (1)
`CbsReset`	6	6	2	2	6	2	1	1	3x1	3x1	1	1	0 (0,17)	0 (0,5)
`DisplayBasic`	0	0	1	1	0	1	1*	1	3x1*	3x1	1*	1	– (–)	– (1)
`DisplayEnhanced`	0	0	1	1	0	1	1*	1	3x1*	3x0	1* (–)	0 (–)	– (–)	– (1)
`Service`	1	1	0	0	1	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
$\sum$	25	25	34	34										
$CPSU_{ports}(pl)$							1							
$SpanCPSU_{ports}(pl)$ [$SpanCPSU'_{ports}(pl)$]									3x0,96 [3x1]					
$AvCPSU_{ports}(pl)$ [$AvCPSU'_{ports}(pl)$]											0,96 [1]			
$CRSU_{ports}(pl)$								1						
$SpanCRSU_{ports}(pl)$ [$SpanCRSU'_{ports}(pl)$]										1x0,84, 2x0,85 [3x1]				
$AvCRSU_{ports}(pl)$ [$AvCRSU'_{ports}(pl)$]												0,85 [1]		

Legende

Tabelle	**Attribut einer Funktion f gemäß Metamodell bzw. Kommentar**
pin_{total}	$f.inputPorts_{total}$
pin_{used}	$f.inputPorts_{used}$
$pout_{total}$	$f.outputPorts_{total}$
$pout_{used}$	$f.outputPorts_{used}$
C_{pout}	$f.connectedFunctions_{pout}$
C_{pin}	$f.connectedFunctions_{pin}$
*	Die Werte, die mit dem Zeichen »*« versehen sind, kennzeichnen, dass für die PSU- und RSU-Maße die angepassten Formeln 3.5 und 3.6 angewendet worden sind.

5.4.2 Anwendung der Metriken auf die Produktmodelle (Funktionsnetze / Ports)

In Tabelle 2 ist die Anwendung der SU-Metriken sowie der Kohärenzmetrik auf die einzelnen Funktionsmodelle der Produkte *P1*, *P2* und *P3* der CBS-Produktlinie dargestellt. Die einzelnen Funktionsmodelle sind aus dem flachgeklopften Produktlinienmodell (Abbildung 13) extrahiert, wie es exemplarisch für das Produkt *P1* in Abbildung 15 angegeben ist. Die notwendigen Eingangswerte für die Kohärenzmetrik sind in Tabelle 3 aufgelistet. Die Metriken wurden hier beispielhaft auf Ports angewendet.

Tabelle 2 Anwendung auf CBS-Produkt-Funktionsnetze bzgl. Ports.

Funktion[Conf: P1|P2|P3]	$\#pout_{total}$ P1|P2|P3	$\#pout_{used}$ P1|P2|P3	$\#pin_{total}$ P1|P2|P3	$\#pin_{used}$ P1|P2|P3	$\#C_{pout}$ P1|P2|P3	$\#C_{pin}$ P1|P2|P3	PSU_{ports} P1|P2|P3	RSU_{ports} P1|P2|P3	ψ_{pout} (ψ'_{pout}) P1|P2|P3	ψ_{pin} (ψ'_{pin}) P1|P2|P3
Mileage	1	1	0	0	4	0	1	1*	– (1)	– (–)
Date	1	1	0	0	5	0	1	1*	– (1)	– (–)
S-BreakPadsFront	1	1	0	0	1	0	1	1*	– (1)	– (–)
S-BreakPadsRear	1	1	0	0	1	0	1	1*	– (1)	– (–)
S-MotorOil	1	1	0	0	1	0	1	1*	– (1)	– (–)
S-SparkPlugs$^{1\|0\|0}$	1	1|0|0	0	0	1|0|0	0	1|0|0	1*	– (1|–|–)	– (–)
S-ParticleFilter$^{0\|0\|1}$	1	0|0|1	0	0	0|0|1	0	0|0|1	1*	– (–|–|1)	– (–)
InputVehicleCheck	1	1	0	0	1	0	1	1*	– (1)	– (–)
CbsWdBreakPadsFront	1	1	4	4	1	4	1	1	– (1)	0 (0,25)
CbsWdBreakPadsRear	1	1	4	4	1	4	1	1	– (1)	0 (0,25)
CbsWdMotorOil	1	1	4	4	1	4	1	1	– (1)	0 (0,25)
CbsWdSparkPlugs$^{1\|1\|0}$	1	1|1|0	4|3|4^{+}	4|3|0	1|1|0	4|3|0	1|1|0	1|1|0	– (1|1|–)	0|0|– (0,25|0,33|–)
CbsWdParticleFilter$^{0\|0\|1}$	1	0|0|1	4	0|0|4	0|0|1	0|0|4	0|0|1	0|0|1	– (–|–|1)	–|–|0 (–|–|0,25)
CbsDmVehicleCheck	1	1	3	3	1	3	1	1	– (1)	0 (0,33)
CbsComputeOptimalServiceDate	2	2	5	5	2	5	1	1	0 (0,5)	0 (0,2)
CbsDisplay	1	1	1	1	1	1	1	1	– (1)	– (1)
CbsReset	5	5	2	2	5	2	1	1	0 (0,2)	0 (0,5)
DisplayBasic	0	0	1	1	0	1	1*	1	– (–)	– (1)
DisplayEnhanced$^{0\|0\|0}$	0	0	1	0	0	0	1*	0	– (–)	– (–)
Service	1	1	0	0	1	0	1	1*	– (1)	– (–)

(Weiter auf der nächsten Seite)

Tabelle 2 Anwendung auf CBS-Produkt-Funktionsnetze bzgl. Ports (Fortsetzung).

Funktion[Conf: P1|P2|P3]	$\#pout_{total}$ P1|P2|P3	$\#pout_{used}$ P1|P2|P3	$\#pin_{total}$ P1|P2|P3	$\#pin_{used}$ P1|P2|P3	$\#C_{pout}$ P1|P2|P3	$\#C_{pin}$ P1|P2|P3	PSU_{ports} P1|P2|P3	RSU_{ports} P1|P2|P3	$\psi_{pout}\ (\psi'_{pout})$ P1|P2|P3	$\psi_{pin}\ (\psi'_{pin})$ P1|P2|P3
$\sum$	23|23|23	21|20|21	33|32|33	28|27|28						
$CPSU_{ports}(p_i)$							0,91|0,87|0,91			
$CRSU_{ports}(p_i)$								0,85|0,84|0,85		
$\sum'_{conf}$	21|20|21	21|20|21	28|27|28	28|27|28						
$CPSU'_{ports}(p_i)$							1|1|1			
$CRSU'_{ports}(p_i)$								1|1|1		

Legende

In der ersten Spalte der Tabelle ist die Konfigurationsinformation für die Funktionen explizit nach dem Schema »$conf(P1,f)$ | $conf(P2,f)$ | $conf(P3,f)$« angegeben. Ist keine Konfiguration angegeben, so gilt für die Funktion: »1|1|1«, d. h., sie ist obligatorisch und in allen Produkten immer vorhanden.

Für die Zellen der Tabelle gilt: Haben Funktionen in den verschiedenen Produkten den gleichen Wert, so ist der Wert nur einmal angegeben. Bei unterschiedlichen Werten sind sie explizit für jedes Produkt in der Reihenfolge »*P1, P2, P3*« angegeben, beispielsweise bedeutet »1|0|1«, dass die Funktion im Produkt *P1* den Wert 1, im Produkt *P2* den Wert 0 und im Produkt *P3* den Wert 1 hat.

Tabelle	**Attribut einer Funktion f gemäß Metamodell bzw. Kommentar**
pin_{total}	$f.inputPorts_{total}$
pin_{used}	$f.inputPorts_{used}$
$pout_{total}$	$f.outputPorts_{total}$
$pout_{used}$	$f.outputPorts_{used}$
C_{pout}	$f.connectedFunctions_{pout}$
C_{pin}	$f.connectedFunctions_{pin}$
*	Die Werte, die mit dem Zeichen »*« versehen sind, kennzeichnen, dass für die PSU- und RSU-Maße die angepassten Formeln 3.5 und 3.6 angewendet worden sind.
+	Die Funktion ist in der Konfiguration für das Produkt *P3* deselektiert, deswegen ist der optionale Port ebenfalls deselektiert. Gesamtanzahl der Eingangsports ist 4, wenn konfiguriert: 3.

5.4.3 Strukturinformationen als Eingangsdaten für die Kohärenzmetrik (Funktionsnetze / Ports)

Die Zählung, wie viele der Ports die an einer Komponente angeschlossenen Komponenten verwenden, ist in Tabelle 3 sowohl für das Produktlinienarchitekturmodell als auch für die einzelnen Produktarchitekturmodelle aufgelistet (in der Reihenfolge »*Produktlinie, Produkt P1, Produkt P2, Produkt P3*«).

Tabelle 3 Eingangsdaten für die Kohärenzmetrik bzgl. Ports.

Funktion	$\#pout_{used}Client_1$ PL|P1|P2|P3	$\#pout_{used}Client_2$ PL|P1|P2|P3	$\#pout_{used}Client_3$ PL|P1|P2|P3	$\#pout_{used}Client_4$ PL|P1|P2|P3	$\#pout_{used}Client_5$ PL|P1|P2|P3	$\#pout_{used}Client_6$ PL|P1|P2|P3	$\sum \#pout_{used}Client_i$ PL|P1|P2|P3	$\#pin_{used}Client_1$ PL|P1|P2|P3	$\#pin_{used}Client_2$ PL|P1|P2|P3	$\#pin_{used}Client_3$ PL|P1|P2|P3	$\#pin_{used}Client_4$ PL|P1|P2|P3	$\#pin_{used}Client_5$ PL|P1|P2|P3	$\#pin_{used}Client_6$ PL|P1|P2|P3	$\sum \#pin_{used}Client_i$ PL|P1|P2|P3
Mileage	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|-|-|-		5|4|4|4							0|0|0|0
Date	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|-|-|-	6|5|5|5							0|0|0|0
S-BreakPadsFront	1|1|1|1						1|1|1|1							0|0|0|0
S-BreakPadsRear	1|1|1|1						1|1|1|1							0|0|0|0
S-MotorOil	1|1|1|1						1|1|1|1							0|0|0|0
S-SparkPlugs	1|1|-|-						1|1|0|0							0|0|0|0
S-ParticleFilter	1|-|-|1						1|0|0|1							0|0|0|0
InputVehicleCheck	1|1|1|1						1|1|1|1							0|0|0|0
CbsWdBreakPads-Front	1|1|1|1						1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1			4|4|4|4
CbsWdBreakPads-Rear	1|1|1|1						1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1			4|4|4|4
CbsWdMotorOil	1|1|1|1						1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1			4|4|4|4
CbsWdSparkPlugs	1|1|1|-						1|1|1|0	1|1|1|-	1|1|1|-	1|1|1|-	1|1|-|-			4|4|3|0
CbsWdParticleFilter	1|-|-|1						1|0|0|1	1|-|-|1	1|-|-|1	1|-|-|1	1|-|-|1			4|0|0|4
CbsDmVehicle-Check	1|1|1|1						1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1				3|3|3|3
CbsComputeOptimalServiceDate	1|1|1|1	1|1|1|1					2|2|2|2	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|-|-|-	6|5|5|5
CbsDisplay	1|1|1|1	1|-|-|-					2|1|1|1	1|1|1|1						1|1|1|1
CbsReset	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|-|-|-	6|5|5|5	1|1|1|1	1|1|1|1						2|2|2|2
DisplayBasic							0|0|0|0	1|1|1|1						1|1|1|1
DisplayEnhanced							0|0|0|0	1|-|-|-						1|0|0|0
Service	1|1|1|1						1|1|1|1							0|0|0|0

5.4.4 Interpretation der Ergebnisse

Wie in der Tabelle 1 ersichtlich ist, liegen die meisten *PSU*- und *RSU*-Werte für die Komponenten der Produktlinie bei 1. Dies ist nicht überraschend, weil bei der hier erfolgten Modellierung von CBS alle Komponenten sowohl auf Produktebene als auch auf Produktlinienebene aufeinander abgestimmt wurden. Immer dann, wenn Komponenten von dritter Seite verwendet werden müssen (zum Beispiel bei der Integration von Modellen verschiedener Fachabteilungen), also auf die Gestaltung der Schnittstelle mancher Komponenten kein Einfluss genommen werden kann und dabei Kompromisse gemacht werden müssen, werden die Werte kleiner als 1 ausfallen.

Die *PSU*- bzw. *RSU*-Spanne von Komponenten enthalten immer dann eine 0, wenn es sich um optionale oder alternative Komponenten handelt, die nicht in sämtlichen Produkten angewendet werden (vgl. zum Beispiel die Funktionskomponenten `S-SparkPlugs` und `CbsWdSparkPlugs`). Das lässt sich auch daran erkennen, dass die Spanne für diese Komponenten bzgl. der angepassten Formeln, *SpanRSU'* und *SpanPSU'*, nur die 1 enthalten. Dies ist darauf zurückzuführen, dass die Komponenten optimal für das jeweilige Produkt konfiguriert sind und in die jeweilige Produktarchitektur passen, wenn sie für das Produkt ausgewählt sind. Gleiches gilt für den Durchschnitt bei *PSU* und *RSU*.

Dass die Kohärenzwerte für viele Funktionen nicht ermittelbar sind, liegt daran, dass sie in der Modellierung nur mit einem Ausgangs- oder Eingangsport versehen worden sind. Bei der angepassten Kohärenzberechnung (ψ') lassen sich immer nur dann Werte bei 1 erreichen, wenn sämtliche Clientfunktionen an sämtliche Ports der betrachteten Funktion angeschlossen sind (vgl. die Sensorfunktionen sowie die Verschleißermittlungfunktionen bzgl. ψ'_{pout}). Immer dann, wenn eine Funktion Werte von verschiedenen Funktionen aufsammelt oder an verschiedene Funktionen verteilt, wird der Kohärenzwert deutlich kleiner als 1 sein (siehe Funktionen `CbsComputeOptimalServiceDate` und `CbsReset`). Dies ist für die hier betrachteten Architekturmodelle jedoch normal und stellt hier im Allgemeinen kein negatives Ergebnis dar.

Die Interpretation der Messwerte für die Einzelkomponenten schlägt sich auch auf die zusammengefassten Werte auf Architekturebene nieder. So sind die *CPSU*- und *CRSU*-Werte ebenfalls bei 1. Wird die Konfigurationsinformation der Komponenten berücksichtigt, so sind die Spanne und der Durchschnitt für die Produktlinie auch bei 1.

Die Interpretation der Messergebnisse für die einzelnen Produktarchitekurmodelle (vgl. Tabelle 2) ist analog. Die *CPSU*- und *CRSU*-Werte weichen von der 1 ab, weil die herauskonfigurierten Funktionen in die Berechnung miteingeflossen sind. Die

CPSU'- und *CRSU'*-Werte, die die Konfigurationsinformationen berücksichtigen, verdeutlichen dies, da hier für jedes Produkt der Wert 1 ist.

5.5 Anwendung der SU-Metriken auf die Beispielmodellierung unter Berücksichtigung der Signale als Dienstbegriff

Im Folgenden werden die SU-Metriken sowie die Kohärenzmetrik auf die Funktionsmodellierung der CBS-Produktlinie angewendet. Grundlage der Berechnung ist der in Abbildung 12 dargestellte Schnitt sowie die Interpretation des Servicebegriffs als Signale von Funktionen.

5.5.1 Anwendung der Metriken auf das Produktlinienmodell (Funktionsnetz / Signale)

In Tabelle 4 ist die Anwendung der SU-Metriken sowie der Kohärenzmetrik auf das Produktlinienfunktionsmodell gemäß des gewählten Schnitts (vgl. flachgeklopftes Funktionsmodell Abbildung 14) tabellarisch dargestellt. Die in Tabelle 4 durchgeführten Berechnungen verwenden auch Werte aus Tabelle 5. Die notwendigen Eingangswerte für die Kohärenzmetrik sind in Tabelle 6 aufgelistet. Die Metriken wurden hier beispielhaft auf Signale angewendet.

Tabelle 4 Anwendung auf CBS-PL-Funktionsnetz bzgl. Signale.

Funktion	$\#sigout_{total}$	$\#sigout_{used}$	$\#sigin_{total}$	$\#sigin_{used}$	$\#C_{sigout}$	$\#C_{sigin}$	$PSU_{signals}$	$RSU_{signals}$	$SpanPSU_{signals}$ ($SpanPSU'_{signals}$)	$SpanRSU_{signals}$ ($SpanRSU'_{signals}$)	$AvPSU_{signals}$ ($AvPSU'_{signals}$)	$AvRSU_{signals}$ ($AvRSU'_{signals}$)	ψ_{sigout} (ψ'_{sigout})	ψ_{sigin} (ψ'_{sigin})
`Mileage`	1	1	0	0	5	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
`Date`	1	1	0	0	6	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)
`S-BreakPadsFront`	5	5	0	0	1	0	1	1*	3x1	3x1*	1	1*	1 (1)	– (–)
`S-BreakPadsRear`	5	5	0	0	1	0	1	1*	3x1	3x1*	1	1*	1 (1)	– (–)
`S-MotorOil`	4	4	0	0	1	0	1	1*	3x1	3x1*	1	1*	1 (1)	– (–)
`S-SparkPlugs`	4	4	0	0	1	0	1	1*	2x0, 1x1 (1x1)	3x1* (1x1*)	0,33 (1)	1*	1 (1)	– (–)
`S-ParticleFilter`	3	3	0	0	1	0	1	1*	2x0, 1x1 (1x1)	3x1* (1x1*)	0,33 (1)	1*	1 (1)	– (–)
`InputVehicle-Check`	1	1	0	0	1	0	1	1*	3x1	3x1*	1	1*	– (1)	– (–)

(Weiter auf der nächsten Seite)

Tabelle 4 Anwendung auf CBS-PL-Funktionsnetz bzgl. Signale (Fortsetzung).

Funktion	$\#sigout_{total}$	$\#sigout_{used}$	$\#sigin_{total}$	$\#sigin_{used}$	$\#C_{sigout}$	$\#C_{sigin}$	$PSU_{signals}$	$RSU_{signals}$	$SpanPSU_{signals}$ ($SpanPSU'_{signals}$)	$SpanRSU_{signals}$ ($SpanRSU'_{signals}$)	$AvPSU_{signals}$ ($AvPSU'_{signals}$)	$AvRSU_{signals}$ ($AvRSU'_{signals}$)	ψ_{sigout} (ψ'_{sigout})	ψ_{sigin} (ψ'_{sigin})
`CbsWdBreakPads-Front`	4	4	10	10	1	4	1	1	3x1	3x1	1	1	1 (1)	0,17 (0,25)
`CbsWdBreakPads-Rear`	4	4	10	10	1	4	1	1	3x1	3x1	1	1	1 (1)	0,17 (0,25)
`CbsWdMotorOil`	4	4	9	9	1	4	1	1	3x1	3x1	1	1	1 (1)	0,16 (0,25)
`CbsWdSparkPlugs`	4	4	9	9	1	4	1	1	1x0, 2x1 (2x1)	1x0, 2x1	0,67 (1)	0,67 (1)	1 (1)	0,16 (0,25)
`CbsWdParticle-Filter`	4	4	8	8	1	4	1	1	2x0, 1x1 (1x1)	2x0, 1x1 (1x1)	0,33 (1)	0,33 (1)	1 (1)	0,14 (0,25)
`CbsDmVehicle-Check`	2	2	5	5	1	3	1	1	3x1	3x1	1	1	1 (1)	0,17 (0,33)
`CbsComputeOptimalServiceDate`	4	4	22	22	2	6	1	1	3x1	3x1	1	1	0,33 (0,5)	0,13 (0,17)
`CbsDisplay`	3	3	3	3	2	1	1	1	3x1	3x1	1	1	0,25 (0,5)	1 (1)
`CbsReset`	18	18	4	4	6	2	1	1	3x1	3x1	1	1	0,12 (0,17)	0,33 (0,5)
`DisplayBasic`	0	0	1	1	0	1	1*	1	3x1*	3x1	1*	1	– (–)	– (1)
`DisplayEnhanced`	0	0	2	2	0	1	1*	1	3x1*	3x0	1*	0	– (–)	1 (1)
`Service`	3	3	0	0	1	0	1	1*	3x1	3x1*	1*	1	1 (1)	– (–)
$\sum$	74	74	83	83										
$CPSU_{signals}(pl)$							1							
$SpanCPSU_{signals}(pl)$ [$SpanCPSU'_{signals}(pl)$]									1x0,90, 1x0,84, 1x0,88 [3x1]					
$AvCPSU_{signals}(pl)$ [$AvCPSU'_{signals}(pl)$]											0,87 [1]			
$CRSU_{signals}(pl)$								1						
$SpanCRSU_{signals}(pl)$ [$SpanCRSU'_{signals}(pl)$]										1x0,86, 2x0,87 [3x1]				
$AvCRSU_{signals}(pl)$ [$AvCRSU'_{signals}(pl)$]												0,87 [1]		

Legende

Tabelle	Attribut einer Funktion f gemäß Metamodell bzw. Kommentar
$sigout_{total}$	$f.outputSignals_{total}$
$sigout_{used}$	$f.outputSignals_{used}$
$sigin_{total}$	$f.inputSignals_{total}$
$sigin_{used}$	$f.inputSignals_{used}$
C_{sigout}	$f.connectedFunctions_{sigout}$
C_{sigin}	$f.connectedFunctions_{sigin}$
*	Die Werte, die mit dem Zeichen »*« versehen sind, kennzeichnen, dass für die PSU- und RSU-Maße die angepassten Formeln 3.5 und 3.6 angewendet worden sind.

5.5.2 Anwendung der Metriken auf die Produktmodelle (Funktionsnetze / Signale)

In Tabelle 5 ist die Anwendung der SU-Metriken sowie der Kohärenzmetrik auf die einzelnen Funktionsmodelle der Produkte der CBS-Produktlinie dargestellt. Die einzelnen Funktionsmodelle sind aus dem flachgeklopften Produktlinienmodell (Abbildung 14) extrahiert, wie es exemplarisch für das Produkt *P1* in Abbildung 15 angegeben ist. Die notwendigen Eingangswerte für die Kohärenzmetrik sind in Tabelle 6 aufgelistet. Die Metriken wurden hier beispielhaft auf Signale angewendet.

Tabelle 5 Anwendung auf CBS-Produkt-Funktionsnetze bzgl. Signale.

Funktion[Conf: P1|P2|P3]	$\#sigout_{total}$ P1|P2|P3	$\#sigout_{used}$ P1|P2|P3	$\#sigin_{total}$ P1|P2|P3	$\#sigin_{used}$ P1|P2|P3	$\#C_{sigout}$ P1|P2|P3	$\#C_{sigin}$ P1|P2|P3	$PSU_{signals}$ P1|P2|P3	$RSU_{signals}$ P1|P2|P3	ψ_{sigout} (ψ'_{sigout}) P1|P2|P3	ψ_{sigin} (ψ'_{sigin}) P1|P2|P3
Mileage	1	1	0	0	4	0	1	1*	– (1)	– (–)
Date	1	1	0	0	5	0	1	1*	– (1)	– (–)
S-BreakPadsFront	5	5	0	0	1	0	1	1*	1 (1)	– (–)
S-BreakPadsRear	5	5	0	0	1	0	1	1*	1 (1)	– (–)
S-MotorOil	4	4	0	0	1	0	1	1*	1 (1)	– (–)
S-SparkPlugs$^{1\|0\|0}$	4	4|0|0	0	0	1|0|0	0	1|0|0	1*	1|–|– (1|–|–)	– (–)
S-ParticleFilter$^{0\|0\|1}$	3	0|0|3	0	0	0|0|1	0	0|0|1	1*	–|–|1 (–|–|1)	– (–)
InputVehicleCheck	1	1	0	0	1	0	1	1*	– (1)	– (–)
CbsWdBreakPadsFront	4	4	10	10	1	4	1	1	1 (1)	0,17 (0,25)

(Weiter auf der nächsten Seite)

Tabelle 5 Anwendung auf CBS-Produkt-Funktionsnetze bzgl. Signale (Fortsetzung).

Funktion[Conf: P1|P2|P3]	$\#sigout_{total}$ P1|P2|P3	$\#sigout_{used}$ P1|P2|P3	$\#sigin_{total}$ P1|P2|P3	$\#sigin_{used}$ P1|P2|P3	$\#C_{sigout}$ P1|P2|P3	$\#C_{sigin}$ P1|P2|P3	$PSU_{signals}$ P1|P2|P3	$RSU_{signals}$ P1|P2|P3	ψ_{sigout} (ψ'_{sigout}) P1|P2|P3	ψ_{sigin} (ψ'_{sigin}) P1|P2|P3
`CbsWdBreakPads-Rear`	4	4	10	10	1	4	1	1	1 (1)	0,17 (0,25)
`CbsWdMotorOil`	4	4	9	9	1	4	1	1	1 (1)	0,16 (0,25)
`CbsWdSpark-Plugs`$^{1\|1\|0}$	4	4|4|0	9|5|9^{+}	9|5|0	1|1|0	4|3|0	1|1|0	1|1|0	1|1|– (1|1|–)	0,16|0,17|– (0,25|0,33|–)
`CbsWdParticle-Filter`$^{0\|0\|1}$	4	0|0|4	8	0|0|8	0|0|1	0|0|4	0|0|1	0|0|1	–|–|1 (–|–|1)	–|–|0,14 (–|–|0,25)
`CbsDmVehicle-Check`	2	2	5	5	1	3	1	1	1 (1)	0,17 (0,33)
`CbsComputeOpti-malServiceDate`	4	4	18	18	2	5	1	1	0,33 (0,5)	0,15 (0,2)
`CbsDisplay`	1	1	3	3	1	1	1	1	– (1)	1 (1)
`CbsReset`	15	15	4	4	5	2	1	1	0,14 (0,2)	0,33 (0,5)
`DisplayBasic`	0	0	1	1	0	1	1*	1	– (–)	– (1)
`DisplayEnhan-ced`$^{0\|0\|0}$	0	0	2	0	0	0	1*	0	– (–)	– (–)
`Service`	3	3	0	0	1	0	1	1*	1 (1)	– (–)
$\sum$	69|69|69	62|58|61	79|75|79	69|65|68						
$CPSU_{signals}(p_i)$							0,90|0,84|0,88			
$CRSU_{signals}(p_i)$								0,87|0,87|0,86		
$\sum'_{conf}$	62|58|61	62|58|61	69|65|68	69|65|68						
$CPSU'_{signals}(p_i)$							1|1|1			
$CRSU'_{signals}(p_i)$								1|1|1		

Legende

In der ersten Spalte der Tabelle ist die Konfigurationsinformation für die Funktionen explizit nach dem Schema »$conf(P1,f)$ | $conf(P2,f)$ | $conf(P3,f)$« angegeben. Ist keine Konfiguration angegeben, so gilt für die Funktion: »1|1|1«.

Für die Zellen der Tabelle gilt: Haben Funktionen in den verschiedenen Produkten den gleichen Wert, so ist der Wert nur einmal angegeben. Unterschiedliche Werte

sind explizit für jedes Produkt in der Reihenfolge »*P1, P2, P3*« angegeben, zum Beipiel: »9|5|0«.

Tabelle	Attribut einer Funktion f gemäß Metamodell bzw. Kommentar
$sigout_{total}$	$f.outputSignals_{total}$
$sigout_{used}$	$f.outputSignals_{used}$
$sigin_{total}$	$f.inputSignals_{total}$
$sigin_{used}$	$f.inputSignals_{used}$
C_{sigout}	$f.connectedFunctions_{sigout}$
C_{sigin}	$f.connectedFunctions_{sigin}$
*	Die Werte, die mit dem Zeichen »*« versehen sind, kennzeichnen, dass für die PSU- und RSU-Maße die angepassten Formeln 3.5 und 3.6 angewendet worden sind.
+	Die Funktion ist in der Konfiguration für das Produkt *P3* deselektiert, deswegen ist der optionale Port ebenfalls deselektiert. Gesamtanzahl der Eingangssignale ist 9, wenn konfiguriert: 5.

5.5.3 Strukturinformationen als Eingangsdaten für die Kohärenzmetrik (Funktionsnetze / Signale)

Die Zählung, wie viele der Ports die an einer Komponente angeschlossenen Komponenten verwenden, ist in Tabelle 6 sowohl für das Produktlinienarchitekturmodell als auch die einzelnen Produktarchitekturmodelle aufgelistet (in der Reihenfolge »*Produktlinie, Produkt P1, Produkt P2, Produkt P3*«).

Tabelle 6 Eingangsdaten für die Kohärenzmetrik bzgl. Signale.

Funktion	$\#sigout_{used}Client_1$ PL|P1|P2|P3	$\#sigout_{used}Client_2$ PL|P1|P2|P3	$\#sigout_{used}Client_3$ PL|P1|P2|P3	$\#sigout_{used}Client_4$ PL|P1|P2|P3	$\#sigout_{used}Client_5$ PL|P1|P2|P3	$\#sigout_{used}Client_6$ PL|P1|P2|P3	$\sum \#sigout_{used}Client_i$ PL|P1|P2|P3	$\#sigin_{used}Client_1$ PL|P1|P2|P3	$\#sigin_{used}Client_2$ PL|P1|P2|P3	$\#sigin_{used}Client_3$ PL|P1|P2|P3	$\#sigin_{used}Client_4$ PL|P1|P2|P3	$\#sigin_{used}Client_5$ PL|P1|P2|P3	$\#sigin_{used}Client_6$ PL|P1|P2|P3	$\sum \#sigin_{used}Client_i$ PL|P1|P2|P3
Mileage	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|-|-|-		5|4|4|4							0|0|0|0
Date	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|1|1|1	1|-|-|-	6|5|5|5							0|0|0|0
S-BreakPadsFront	5|5|5|5						5|5|5|5							0|0|0|0
S-BreakPadsRear	5|5|5|5						5|5|5|5							0|0|0|0
S-MotorOil	4|4|4|4						4|4|4|4							0|0|0|0
S-SparkPlugs	4|4|-|-						4|4|0|0							0|0|0|0
S-ParticleFilter	3|-|-|3						3|0|0|3							0|0|0|0
InputVehicleCheck	1|1|1|1						1|1|1|1							0|0|0|0
CbsWdBreakPads-Front	4|4|4|4						4|4|4|4	1|1|1|1	1|1|1|1	5|5|5|5	3|3|3|3			10|10|10|10

(Weiter auf der nächsten Seite)

Tabelle 6 Eingangsdaten für die Kohärenzmetrik bzgl. Signale (Fortsetzung).

Funktion	$\#pout_{used}Client_1$ PL|P1|P2|P3	$\#pout_{used}Client_2$ PL|P1|P2|P3	$\#pout_{used}Client_3$ PL|P1|P2|P3	$\#pout_{used}Client_4$ PL|P1|P2|P3	$\#pout_{used}Client_5$ PL|P1|P2|P3	$\#pout_{used}Client_6$ PL|P1|P2|P3	$\sum \#pout_{used}Client_i$ PL|P1|P2|P3	$\#pin_{used}Client_1$ PL|P1|P2|P3	$\#pin_{used}Client_2$ PL|P1|P2|P3	$\#pin_{used}Client_3$ PL|P1|P2|P3	$\#pin_{used}Client_4$ PL|P1|P2|P3	$\#pin_{used}Client_5$ PL|P1|P2|P3	$\#pin_{used}Client_6$ PL|P1|P2|P3	$\sum \#pin_{used}Client_i$ PL|P1|P2|P3
CbsWdBreakPads-Rear	4|4|4|4						4|4|4|4	1|1|1|1	1|1|1|1	5|5|5|5	3|3|3|3			10|10| 10|10
CbsWdMotorOil	4|4|4|4						4|4|4|4	1|1|1|1	1|1|1|1	4|4|4|4	3|3|3|3			9|9|9|9
CbsWdSparkPlugs	4|4|4|-						4|4|4|0	1|1|1|-	1|1|1|-	4|4|-|-	3|3|3|-			9|9|5|0
CbsWdParticleFilter	4|-|-|4						4|0|0|4	1|-|-|1	1|-|-|1	3|-|-|3	3|-|-|3			8|0|0|8
CbsDmVehicle-Check	2|2|2|2						2|2|2|2	1|1|1|1	1|1|1|1	3|3|3|3				5|5|5|5
CbsComputeOptimalServiceDate	1|1|1|1	3|3|3|3					4|4|4|4	4|4|4|4	4|4|4|4	4|4|4|4	4|4|4|-	4|-|-|4	2|2|2|2	22|18| 18|18
CbsDisplay	1|1|1|1	2|-|-|-					3|1|1|1	3|3|3|3						3|3|3|3
CbsReset	3|3|3|3	3|3|3|3	3|3|3|3	3|3|3|3	3|3|3|-	3|-|-|3	18|15| 15|15	1|1|1|1	3|3|3|3					4|4|4|4
DisplayBasic							0|0|0|0	1|1|1|1						1|1|1|1
DisplayEnhanced							0|0|0|0	2|-|-|-						2|0|0|0
Service	3|3|3|3						3|3|3|3							0|0|0|0

5.5.4 Interpretation der Ergebnisse

Die Anwendung der Metriken auf Signale anstatt auf Ports von Funktionen ergibt kein prinzipiell anderes Ergebnis für die vorliegende CBS-Funktionsmodellierung. Dies resultiert daraus, dass Ports, die miteinander verbunden sind, mit der gleichen Portdeklaration typisiert wurden, so dass sämtliche Signale vom Ausgangsport auch vom gegenüberliegenden Eingangsport empfangen werden. Auch hier gilt, immer dann, wenn Komponenten von dritter Hand verwendet werden müssen oder für manche Komponenten standardisierte Portdeklarationen (standardisierte Schnittstellen) genutzt werden müssen, können die Werte von 1 abweichen.

Wird Varianz auf Signalebene zugelassen, beispielsweise im Fall von konfigurierbaren, variierenden Signalmengen bei Portdeklarationen, so kann aufgrund einer nicht optimalen Konfiguration ein anderes Ergebnis vorliegen als bei der Betrachtung auf Portebene. Durch Vergleich der Ergebnisse der Messungen bzgl. Ports und Signale lassen sich solche Situationen jedoch identifizieren.

6 Format für die Eingangsdaten für die Anwendung der Metriken der Servicenutzungsgrade und Kohärenz (Ausgabeformat des VEIA-Demonstrators)

Die folgende DTD[15] spezifiziert die notwendigen Eingangsdaten zur Berechnung der SU-Metriken für eine Funktions- bzw. Softwarearchitekturmodellierung einer Produktlinie. Die Daten können auf Grundlage des in Kapitel 4 vorgestellten Datenmodells erhoben bzw. berechnet werden, vgl. auch die Beispielberechnung in Kapitel 5.

Die DTD passt sowohl für Funktions- als auch für Softwarearchitekturmodelle. Funktionen bzw. Softwarekomponenten werden nachfolgend als `component` referenziert.

6.1 DTD für die Eingangsdaten zur Anwendung der Kohäsions- und Kohärenzmetriken

In Listing 1 ist die DTD angegeben, die die erforderlichen Eingangsdaten spezifiziert, um die Kohäsions- und Kohärenzmetriken für VEIA-Architekturmodelle anwenden zu können.

Listing 1 DTD für die Eingabedaten zur Anwendung der Metriken für Servicenutzungsgraden und Kohärenz in komponentenbasierten Architekturmodellen (`metrics_sum.dtd`).

```
<!-- Hauptelement: metrics_sum_input> -->
<!ELEMENT metrics_sum_input (plarchitecture, parchitecture+)>

<!-- Produktlinienmodell / Produktlinienarchitektur -->
<!ELEMENT plarchitecture (id, name, number_products, number_components,
  components)>

<!-- Einzelmodell / Architektur -->
<!ELEMENT parchitecture (id, name, number_components_pl,
  number_components_selected, components)>

<!-- Informationen zu Komponenten
  (Funktionen bzw. Softwarekomponenten) -->
```

[15] »DTD« steht für »*document type defintion*«.

```
<!ELEMENT components (component*)>

<!ELEMENT component
  (id, name,
  configurationinformation?,
  number_inports_total, number_inports_used,
  number_outports_total, number_outports_used,
  number_insignals_total, number_insignals_used,
  number_outsignals_total, number_outsignals_used,
  number_inconnectors, number_outconnectors,
  number_incomponents, number_outcomponents,
  incomponents, outcomponents)>

<!ELEMENT incomponents (component_connected*)>

<!ELEMENT outcomponents (component_connected*)>

<!ELEMENT component_connected
  (id, name, number_ports_used, number_signals_used)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT configurationinformation (#PCDATA)>
<!ELEMENT number_products (#PCDATA)>
<!ELEMENT number_components (#PCDATA)>
<!ELEMENT number_components_pl (#PCDATA)>
<!ELEMENT number_components_selected (#PCDATA)>
<!ELEMENT number_ports_used (#PCDATA)>
<!ELEMENT number_inports_total (#PCDATA)>
<!ELEMENT number_inports_used (#PCDATA)>
<!ELEMENT number_outports_total (#PCDATA)>
<!ELEMENT number_outports_used (#PCDATA)>

<!ELEMENT number_signals_used (#PCDATA)>
<!ELEMENT number_insignals_total (#PCDATA)>
<!ELEMENT number_insignals_used (#PCDATA)>
<!ELEMENT number_outsignals_total (#PCDATA)>
<!ELEMENT number_outsignals_used (#PCDATA)>

<!ELEMENT number_inconnectors (#PCDATA)>
<!ELEMENT number_outconnectors (#PCDATA)>

<!ELEMENT number_incomponents (#PCDATA)>
<!ELEMENT number_outcomponents (#PCDATA)>
```

6.2 Beispiel-XML-Daten

XML-Beispieldaten, konform zur DTD in Listing 1, sind in Listing 2 angegeben. Die folgenden Daten sind konform zur Modellierung in Kapitel 5 und beziehen sich auf den in Abschnitt 5.2 definierten Schnitt (vgl. Abbildung 14).

Listing 2 XML-Beispieldaten (Datei `metrics_sum_inputdata_example.xml`)

```
<metrics_sum_input>

  <!-- Data of a product line architecture -->
  <plarchitecture>
    <id>cbs_fn_pl_c1</id>
    <name>CBS Product Line Function Net (Cut 1)</name>
    <number_products>3</number_products>
    <number_components>20</number_components>
    <components>

      <!-- Data of a component in the product line architecture -->
      <component>
        <id>cbswdsparkplugs</id>
        <name>CbsWdSparkPlugs</name>

        <!-- Ports of a component -->
        <number_inports_total>4</number_inports_total>
        <number_inports_used>4</number_inports_used>
        <number_outports_total>1</number_outports_total>
        <number_outports_used>1</number_outports_used>

        <!-- Signals of a component -->
        <number_insignals_total>9</number_insignals_total>
        <number_insignals_used>9</number_insignals_used>
        <number_outsignals_total>4</number_outsignals_total>
        <number_outsignals_used>4</number_outsignals_used>

        <!-- Externally horizontal connectors of a component -->
        <number_inconnectors>4</number_inconnectors>
        <number_outconnectors>1</number_outconnectors>

        <!-- Externally connected components of a component -->
        <number_incomponents>4</number_incomponents>
        <number_outcomponents>1</number_outcomponents>

        <!-- On input ports connected components of a component -->
        <incomponents>
          <component_connected>
            <id>mileage</id>
            <name>Mileage</name>
            <number_ports_used>1</number_ports_used>
```

```
        <number_signals_used>1</number_signals_used>
      </component_connected>
      <component_connected>
        <id>date</id>
        <name>Date</name>
        <number_ports_used>1</number_ports_used>
        <number_signals_used>1</number_signals_used>
      </component_connected>
      <component_connected>
        <id>cbsreset</id>
        <name>CbsReset</name>
        <number_ports_used>1</number_ports_used>
        <number_signals_used>3</number_signals_used>
      </component_connected>
      <component_connected>
        <id>sensorssparkplugs</id>
        <name>SensorsSparkPlugs</name>
        <number_ports_used>1</number_ports_used>
        <number_signals_used>4</number_signals_used>
      </component_connected>
    </incomponents>

    <!-- On output ports connected components of a component -->
    <outcomponents>
      <component_connected>
        <id>cbscomputeoptimalservicedate</id>
        <name>CbsComputeOptimalServiceDate</name>
        <number_ports_used>1</number_ports_used>
        <number_signals_used>4</number_signals_used>
      </component_connected>
    </outcomponents>

  </component>

  <!-- Data of the other 19 components -->
  <!-- ... omitted here ... -->

 </components>
</plarchitecture>

<!-- Data of a product architecture,
     generated by configuration of the
     product line architecture above -->
<parchitecture>
  <id>cbs_fn_p1_c1</id>
  <name>CBS Function Net, Product P1 (Cut 1)</name>
  <number_components_pl>20</number_components_pl>
  <number_components_selected>17</number_components_selected>
  <components>
```

```
<!-- Data of a component in the product architecture -->
<component>
  <id>cbssparkplugs</id>
  <name>CbsSparkPlugs</name>
  <configurationinformation>1</configuratoninformation>

  <!-- Ports of a component -->
  <number_inports_total>4</number_inports_total>
  <number_inports_used>4</number_inports_used>
  <number_outports_total>1</number_outports_total>
  <number_outports_used>1</number_outports_used>

  <!-- Signals of a component-->
  <number_insignals_total>9</number_insignals_total>
  <number_insignals_used>9</number_insignals_used>
  <number_outsignals_total>4</number_outsignals_total>
  <number_outsignals_used>4</number_outsignals_used>

  <!-- Externally horizontal connectors of a component -->
  <number_inconnectors>4</number_inconnectors>
  <number_outconnectors>1</number_outconnectors>

  <!-- Externally connected components of a component -->
  <number_incomponents>4</number_incomponents>
  <number_outcomponents>1</number_outcomponents>

  <!-- On input ports connected components of a component -->
  <incomponents>
    <component_connected>
      <id>mileage</id>
      <name>Mileage</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>1</number_signals_used>
    </component_connected>
    <component_connected>
      <id>date</id>
      <name>Date</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>1</number_signals_used>
    </component_connected>
    <component_connected>
      <id>cbsreset</id>
      <name>CbsReset</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>3</number_signals_used>
    </component_connected>
    <component_connected>
      <id>sensorssparkplugs</id>
```

```
      <name>SensorsSparkPlugs</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>4</number_signals_used>
    </component_connected>
  </incomponents>

  <!-- On output ports connected components of a component -->
  <outcomponents>
    <component_connected>
      <id>cbscomputeoptimalservicedate</id>
      <name>CbsComputeOptimalServiceDate</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>4</number_signals_used>
    </component_connected>
  </outcomponents>
</component>

<!-- Data of a component in the product architecture -->
<component>
  <id>cbsparticlefilteradaptive</id>
  <name>CbsParticleFilterAdaptive</name>
  <configurationinformation>0</configuratoninformation>

  <!-- Ports of a component -->
  <number_inports_total>4</number_inports_total>
  <number_inports_used>0</number_inports_used>
  <number_outports_total>1</number_outports_total>
  <number_outports_used>0</number_outports_used>

  <!-- Signals of a component-->
  <number_insignals_total>9</number_insignals_total>
  <number_insignals_used>0</number_insignals_used>
  <number_outsignals_total>4</number_outsignals_total>
  <number_outsignals_used>0</number_outsignals_used>

  <!-- Externally horizontal connectors of a component -->
  <number_inconnectors>0</number_inconnectors>
  <number_outconnectors>0</number_outconnectors>

  <!-- Externally connected components of a component -->
  <number_incomponents>0</number_incomponents>
  <number_outcomponents>0</number_outcomponents>

  <!-- On input ports connected components of a component -->
  <incomponents></incomponents>

  <!-- On output ports connected components of a component -->
  <outcomponents></outcomponents>
</component>
```

```
    <!-- Data of the other 18 components -->
    <!-- ... omitted here ... -->

  </components>
</parchitecture>

<!-- Data of a product architecture,
     generated by configuration of the
     product line architecture above -->
<parchitecture>
  <id>cbs_fn_p2</id>
  <name>CBS Function Net, Product P2 (Cut 1)</name>
  <number_components_pl>20</number_components_pl>
  <number_components_selected>16</number_components_selected>
  <components>
    <!-- Data of the component in the product architecture -->
    <component>
      <id>cbssparkplugs</id>
      <name>CbsSparkPlugs</name>
      <configurationinformation>1</configurationinformation>

      <!-- Ports of a component -->
      <number_inports_total>3</number_inports_total>
      <number_inports_used>3</number_inports_used>
      <number_outports_total>1</number_outports_total>
      <number_outports_used>1</number_outports_used>

      <!-- Signals of a component -->
      <number_insignals_total>5</number_insignals_total>
      <number_insignals_used>5</number_insignals_used>
      <number_outsignals_total>4</number_outsignals_total>
      <number_outsignals_used>4</number_outsignals_used>

      <!-- Externally horizontal connectors of a component -->
      <number_inconnectors>3</number_inconnectors>
      <number_outconnectors>1</number_outconnectors>

      <!-- Externally connected components of a component -->
      <number_incomponents>3</number_incomponents>
      <number_outcomponents>1</number_outcomponents>

      <!-- On input ports connected components of a component -->
      <incomponents>
        <component_connected>
          <id>mileage</id>
          <name>Mileage</name>
          <number_ports_used>1</number_ports_used>
          <number_signals_used>1</number_signals_used>
```

```
    </component_connected>
    <component_connected>
      <id>date</id>
      <name>Date</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>1</number_signals_used>
    </component_connected>
    <component_connected>
      <id>cbsreset</id>
      <name>CbsReset</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>3</number_signals_used>
    </component_connected>
  </incomponents>

  <!-- On output ports connected components of a component -->
  <outcomponents>
    <component_connected>
      <id>cbscomputeoptimalservicedate</id>
      <name>CbsComputeOptimalServiceDate</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>4</number_signals_used>
    </component_connected>
  </outcomponents>
</component>

<!-- Data of a component in the product architecture -->
<component>
  <id>cbsparticlefilteradaptive</id>
  <name>CbsParticleFilterAdaptive</name>
  <configurationinformation>0</configuratoninformation>

  <!-- Ports of a component -->
  <number_inports_total>4</number_inports_total>
  <number_inports_used>0</number_inports_used>
  <number_outports_total>1</number_outports_total>
  <number_outports_used>0</number_outports_used>

  <!-- Signals of a component-->
  <number_insignals_total>9</number_insignals_total>
  <number_insignals_used>0</number_insignals_used>
  <number_outsignals_total>4</number_outsignals_total>
  <number_outsignals_used>0</number_outsignals_used>

  <!-- Externally horizontal connectors of a component -->
  <number_inconnectors>0</number_inconnectors>
  <number_outconnectors>0</number_outconnectors>

  <!-- Externally connected components of a component -->
```

```
      <number_incomponents>0</number_incomponents>
      <number_outcomponents>0</number_outcomponents>

      <!-- On input ports connected components of a component -->
      <incomponents></incomponents>

      <!-- On output ports connected components of a component -->
      <outcomponents></outcomponents>
    </component>

    <!-- Data of the other 18 components -->
    <!-- ... omitted here ... -->

  </components>
</parchitecture>


<!-- Data of a product architecture,
     generated by configuration of the
     product line architecture above -->
<parchitecture>
  <id>cbs_fn_p3</id>
  <name>CBS Function Net, Product P3 (Cut 1)</name>
  <number_components_pl>20</number_components_pl>
  <number_components_selected>17</number_components_selected>

  <components>
    <!-- Data of a component in the product architecture -->
    <component>
      <id>cbssparkplugs</id>
      <name>CbsSparkPlugs</name>
      <configurationinformation>0</configurationinformation>

      <!-- Ports of a component -->
      <number_inports_total>4</number_inports_total>
      <number_inports_used>4</number_inports_used>
      <number_outports_total>0</number_outports_total>
      <number_outports_used>0</number_outports_used>

      <!-- Signals of a component -->
      <number_insignals_total>9</number_insignals_total>
      <number_insignals_used>9</number_insignals_used>
      <number_outsignals_total>0</number_outsignals_total>
      <number_outsignals_used>0</number_outsignals_used>

      <!-- Externally horizontal connectors of a component -->
      <number_inconnectors>0</number_inconnectors>
      <number_outconnectors>0</number_outconnectors>
```

```
  <!-- Externally connected components of a component -->
  <number_incomponents>0</number_incomponents>
  <number_outcomponents>0</number_outcomponents>

  <!-- On input ports connected components of a component -->
  <incomponents></incomponents>

  <!-- On output ports connected components of a component -->
  <outcomponents></outcomponents>
</component>

<!-- Data of a component in the product architecture -->
<component>
  <id>cbsparticlefilteradaptive</id>
  <name>CbsParticleFilterAdaptive</name>
  <configurationinformation>1</configuratoninformation>

  <!-- Ports of a component -->
  <number_inports_total>4</number_inports_total>
  <number_inports_used>4</number_inports_used>
  <number_outports_total>1</number_outports_total>
  <number_outports_used>1</number_outports_used>

  <!-- Signals of a component-->
  <number_insignals_total>8</number_insignals_total>
  <number_insignals_used>8</number_insignals_used>
  <number_outsignals_total>4</number_outsignals_total>
  <number_outsignals_used>4</number_outsignals_used>

  <!-- Externally horizontal connectors of a component -->
  <number_inconnectors>4</number_inconnectors>
  <number_outconnectors>1</number_outconnectors>

  <!-- Externally connected components of a component -->
  <number_incomponents>4</number_incomponents>
  <number_outcomponents>1</number_outcomponents>

  <!-- On input ports connected components of a component -->
  <incomponents>
    <component_connected>
      <id>mileage</id>
      <name>Mileage</name>
      <number_ports_used>1</number_ports_used>
      <number_signals_used>1</number_signals_used>
    </component_connected>
    <component_connected>
      <id>date</id>
      <name>Date</name>
      <number_ports_used>1</number_ports_used>
```

```
            <number_signals_used>1</number_signals_used>
          </component_connected>
          <component_connected>
            <id>cbsreset</id>
            <name>CbsReset</name>
            <number_ports_used>1</number_ports_used>
            <number_signals_used>3</number_signals_used>
          </component_connected>
          <component_connected>
            <id>sensorsparticlefilter</id>
            <name>SensorsParticleFilter</name>
            <number_ports_used>1</number_ports_used>
            <number_signals_used>3</number_signals_used>
          </component_connected>
        </incomponents>

        <!-- On output ports connected components of a component -->
        <outcomponents>
          <component_connected>
            <id>cbscomputeoptimalservicedate</id>
            <name>CbsComputeOptimalServiceDate</name>
            <number_ports_used>1</number_ports_used>
            <number_signals_used>4</number_signals_used>
          </component_connected>
        </outcomponents>
      </component>

      <!-- Data of the other 18 components -->
      <!-- ... omitted here ... -->

    </components>
  </parchitecture>


</metrics_sum_input>
```

7 Format für die Eingangsdaten für die Anwendung der PSSF-Metriken (Ausgabeformat des VEIA-Demonstrators)

Die folgende Spezifikation der notwendigen Eingangsdaten zur Berechnung der PSSF-Metriken (siehe [GKM07b]) sollten sowohl für Funktions- als auch Softwarearchitekturmodelle verwendet werden können. Funktionen und Softwarekomponenten werden im Folgenden mit `component` referenziert.

7.1 DTD für die Eingangsdaten zur Anwendung der Metriken

In Listing 3 ist die DTD angegeben, die die erforderlichen Eingangsdaten spezifiziert, um die PSSF-Metriken für VEIA-Architekturmodelle anwenden zu können.

Listing 3 DTD für die Eingangsdaten zur Anwendung der PSSF-Metriken (`metrics_pssf.dtd`)

```
<!-- Hauptelement: metrics_pssf -->
<!ELEMENT metrics_pssf (architecture, functions)>

<!-- Allgemeines -->
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>

<!-- Element: Architektur -->
<!ELEMENT architecture (id, name, products)>
<!ELEMENT products (#PCDATA)>

<!-- Element: Funktionen -->
<!ELEMENT functions (function+)>
<!ELEMENT function (id, name, in, out, var, variability,
    realtimecategory, implementations, hooks)>
<!ELEMENT variability (#PCDATA)>
<!ELEMENT realtimecategory (#PCDATA)>
<!ELEMENT implementations (#PCDATA)>
<!ELEMENT hooks (#PCDATA)>
<!ELEMENT in (#PCDATA)>
<!ELEMENT out (#PCDATA)>
<!ELEMENT var (#PCDATA)>
```

7.2 Beispiel-XML-Daten

XML-Beispieldaten, konform zur DTD in Listing 3, sind in Listing 4 angegeben. Die folgenden Daten sind konform zur Modellierung in Kapitel 5 und beziehen sich auf den in Abschnitt 5.2 definierten Schnitt (vgl. Abbildung 13). Es wurden im Folgenden nur die Eingangs- und Ausgangsports einer Komponente, nicht deren Eingangs- bzw. Ausgangssignale, gezählt.

Listing 4 XML-Beispieldaten (Datei `metrics_pssf_inputdata_example.xml`)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE metrics_pssf SYSTEM "metrics_pssf.dtd" >

<metrics_pssf>

  <architecture>
    <id>cbs_fn_pl_c1</id>
    <name>CBS Product Line Function Net (Cut 1)</name>

    <products>3</products>
  </architecture>

  <functions>
    <function>
      <id>mileage</id>
      <name>Mileage</name>

      <in>0</in>
      <out>1</out>
      <var>0</var>

      <variability>EG</variability>
      <realtimecategory>weich</realtimecategory>
      <implementations>1</implementations>
      <hooks>1</hooks>
    </function>

    <function>
      <id>date</id>
      <name>Date</name>

      <in>0</in>
      <out>1</out>
      <var>0</var>

      <variability>EG</variability>
      <realtimecategory>weich</realtimecategory>
      <implementations>1</implementations>
```

```
  <hooks>1</hooks>
</function>

<function>
  <id>sensorsbreakpadsfront</id>
  <name>SensorsBreakPadsFront</name>

  <in>0</in>
  <out>1</out>
  <var>0</var>

  <variability>EG</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>1</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>sensorsbreakpadsrear</id>
  <name>SensorsBreakPadsRear</name>

  <in>0</in>
  <out>1</out>
  <var>0</var>

  <variability>EG</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>1</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>sensorsmotoroil</id>
  <name>SensorsMotorOil</name>

  <in>0</in>
  <out>1</out>
  <var>0</var>

  <variability>EG</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>1</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>sensorssparkplugs</id>
  <name>SensorsSparkPlugs</name>
```

```
  <in>0</in>
  <out>1</out>
  <var>0</var>

  <variability>EV</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>0,25</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>sensorsparticlefilter</id>
  <name>SensorsParticleFilter</name>

  <in>0</in>
  <out>1</out>
  <var>0</var>

  <variability>EV</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>0,5</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>inputvehiclecheck</id>
  <name>InputVehicleCheck</name>

  <in>0</in>
  <out>1</out>
  <var>0</var>

  <variability>EG</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>1</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>cbswdbreakpadsadaptive</id>
  <name>CbsWdBreakPadsAdaptive</name>

  <in>4</in>
  <out>1</out>
  <var>0</var>

  <variability>EG</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>1</implementations>
```

```
  <hooks>2</hooks>
</function>

<function>
  <id>cbswdmotoroiladaptive</id>
  <name>CbsWdMotorOilAdaptive</name>

  <in>4</in>
  <out>1</out>
  <var>0</var>

  <variability>EG</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>1</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>cbswdsparkplugs</id>
  <name>CbsWdSparkPlugs</name>

  <in>4</in>
  <out>1</out>
  <var>1</var>

  <variability>EV</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>0,25</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>cbswdparticlefilteradaptive</id>
  <name>CbsWdParticleFilterAdaptive</name>

  <in>4</in>
  <out>1</out>
  <var>0</var>

  <variability>EV</variability>
  <realtimecategory>weich</realtimecategory>
  <implementations>0,5</implementations>
  <hooks>1</hooks>
</function>

<function>
  <id>cbsdmvehiclecheck</id>
  <name>CbsDmVehicleCheck</name>
```

```
    <in>3</in>
    <out>1</out>
    <var>0</var>

    <variability>EG</variability>
    <realtimecategory>weich</realtimecategory>
    <implementations>1</implementations>
    <hooks>1</hooks>
  </function>

  <function>
    <id>cbscomputeoptimalservicedate</id>
    <name>CbsComputeOptimalServiceDate</name>

    <in>6</in>
    <out>2</out>
    <var>2</var>

    <variability>EG</variability>
    <realtimecategory>weich</realtimecategory>
    <implementations>1</implementations>
    <hooks>1</hooks>
  </function>

  <function>
    <id>cbsdisplay</id>
    <name>CbsDisplay</name>

    <in>1</in>
    <out>2</out>
    <var>1</var>

    <variability>EG</variability>
    <realtimecategory>weich</realtimecategory>
    <implementations>1</implementations>
    <hooks>1</hooks>
  </function>

  <function>
    <id>cbsreset</id>
    <name>CbsReset</name>

    <in>2</in>
    <out>6</out>
    <var>2</var>

    <variability>EG</variability>
    <realtimecategory>weich</realtimecategory>
    <implementations>1</implementations>
```

```
      <hooks>1</hooks>
    </function>

    <function>
      <id>displaybasic</id>
      <name>DisplayBasic</name>

      <in>1</in>
      <out>0</out>
      <var>0</var>

      <variability>EG</variability>
      <realtimecategory>weich</realtimecategory>
      <implementations>1</implementations>
      <hooks>1</hooks>
    </function>

    <function>
      <id>displayenhanced</id>
      <name>DisplayEnhanced</name>

      <in>1</in>
      <out>0</out>
      <var>0</var>

      <variability>EV</variability>
      <realtimecategory>weich</realtimecategory>
      <implementations>0</implementations>
      <hooks>1</hooks>
    </function>

    <function>
      <id>service</id>
      <name>Service</name>

      <in>1</in>
      <out>0</out>
      <var>0</var>

      <variability>EG</variability>
      <realtimecategory>weich</realtimecategory>
      <implementations>1</implementations>
      <hooks>1</hooks>
    </function>

  </functions>

</metrics_pssf>
```

8 Zusammenfassung und Ausblick

Ein wesentlicher Bestandteil in einer modellbasierten Entwicklung ist, Modelle zu bewerten, um die Qualität bei der Entwicklung von Systemen sicherzustellen. Bei einer Produktlinienentwicklung muss zum einen sichergestellt werden, dass Produkte auf Basis der Produktlinie herstellbar sind, zum anderen ist auf Effizienz zu achten: Komponenten, die in nur wenigen Produkten verwendet werden, sind insgesamt ungünstiger, da der relative Aufwand ihrer Erstellung umso höher liegt. Aussagen zu dieser Frage geben beispielsweise die PSSF-Metriken, die ausführlich in [GKM07b] diskutiert wurden.

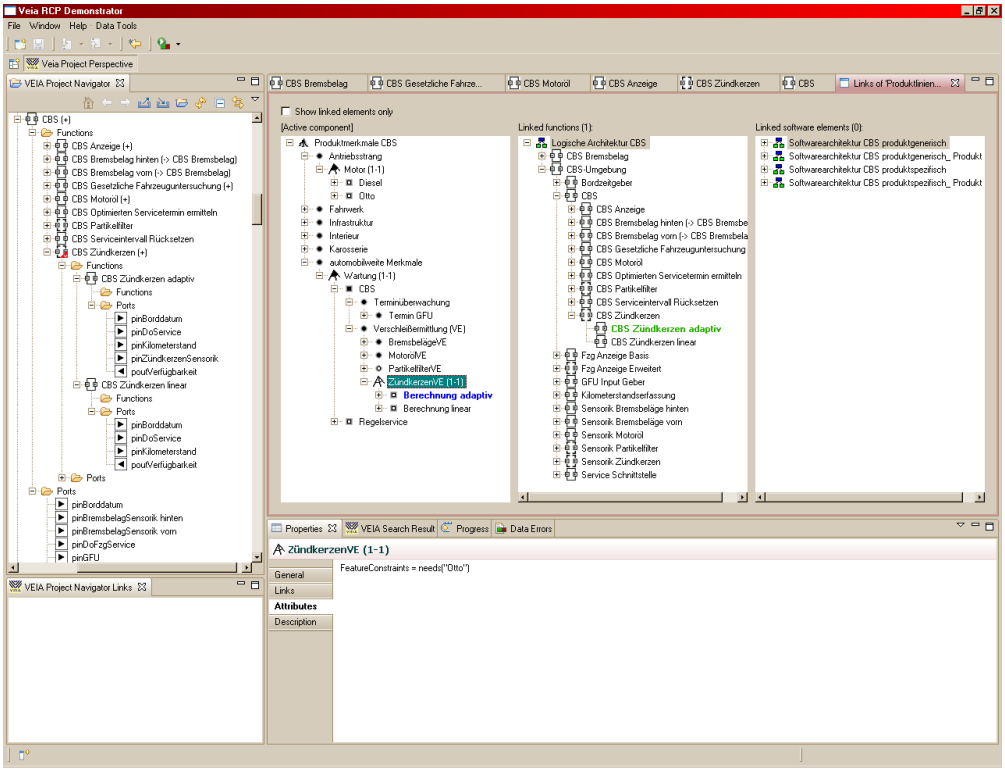
Weiterhin ist darauf zu achten, inwieweit bzw. wie gut Produktlinienassets, d. h. im Rahmen des vorliegenden Berichts insbesondere die Komponenten der Produktlinie, in die jeweiligen Produkte passen. Aussagen hierzu liefern Metriken, die Architekturmodelle hinsichtlich Nutzungsgrade, Kohäsion und Kohärenz messen. Solche Metriken wurden im Rahmen des vorliegenden Berichts auf ihre Anwendbarkeit auf Architekturmodellierungen von Produktlinien gemäß des VEIA-Referenzprozesses [GEKM07] betrachtet.

Die in Kapitel 5 beispielhaft dargestellte Anwendung der Metriken macht deutlich, dass eine Werkzeugunterstützung unbedingt notwendig ist, um Metriken auf (große) Modelle anwenden zu können. Insbesondere ist dies erforderlich in einer Produktlinienentwicklung, da gleich eine ganze Reihe von unterschiedlichen Produkten auf einer gemeinsamen Basis erstellt werden. Die Bewertung einzelner bzw. aller Produkte sowie ihres Bezugs zur Produktlinie stellen besondere Herausforderungen an eine entsprechende Datenerhebung.

Die in Kapitel 5 aufgeführten Eingangsdaten für die Metriken wurden händisch aus den Modellierungen erhoben, die Berechnung der Metriken erfolgte automatisiert mit Hilfe von *OpenOffice Calc* (bzw. *Microsoft Excel*). Angedacht ist eine werkzeuggestützte Datenerhebung, hierzu dienen die in Kapitel 4 vorgestellten Metamodelle sowie die Formatisierung der notwendigen Eingangsdaten als XML, siehe Kapitel 6. Aus diesem Grund erfolgte auch eine ausführliche Diskussion der Anpassung der Metriken auf die VEIA-Modelle in Kapitel 3.

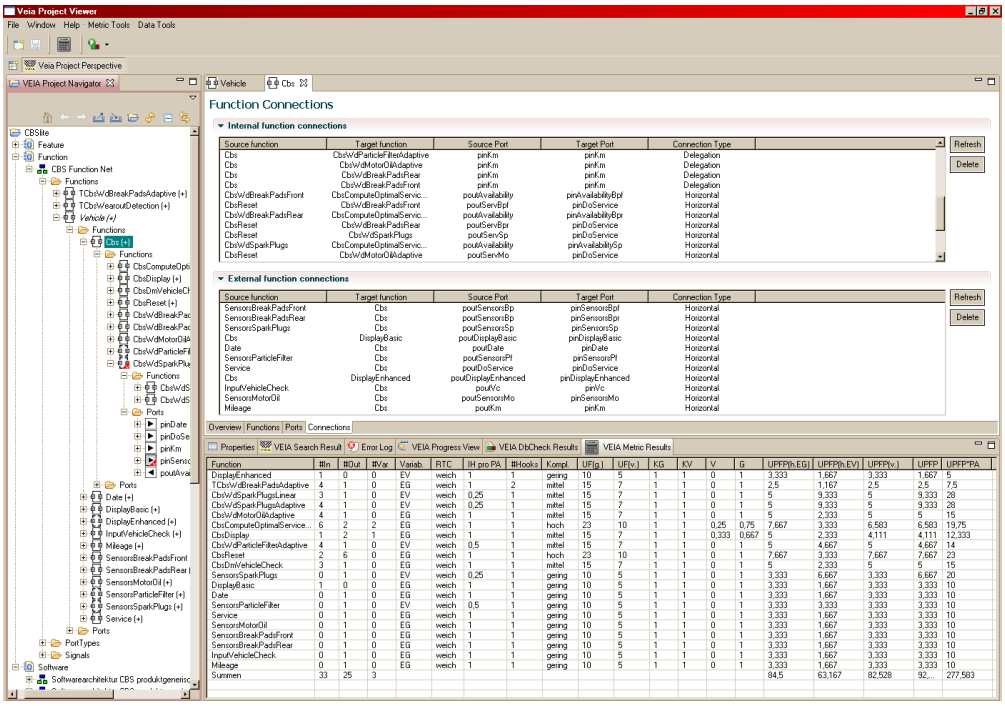
Im Rahmen des VEIA-Projekts wird eine prototypische Umsetzung von Modellierungskonzepten für Produktlinienbeschreibungen in einem CASE-Tool durchgeführt. Die Federführung hat dabei der VEIA-Kooperationspartner PROSTEP IMP GmbH. Mit dem VEIA-Prototyp lassen sich bereits die Artefakte »Produktlinienbeschreibung«, »Funktionsnetz« und »Softwarearchitektur« des VEIA-Referenzprozesses erfassen und miteinander verlinken, vgl. Screenshot in Abbildung 16. Eine

Abbildung 16 Screenshot des VEIA-Prototyps: Feature-, Funktions- und Softwarearchitekturmodelle mit Verlinkung.



erste Anbindung von Metriken ist ebenfalls bereits umgesetzt worden: Mit Hilfe des Prototyps können die für die Anwendung der PSSF-Metriken [GKM07b] notwendigen Eingangsdaten aus dem Funktionsnetz erhoben werden und gemäß der in Kapitel 7 angegebenen DTD als XML exportiert werden. Ein weiterer Prototyp, »Der Metriker« (erstellt vom VEIA-Kooperationspartner Fraunhofer ISST), liest diese Daten ein und führt die Metrikberechnung durch. Die Ergebnisse werden wieder im VEIA-CASE-Tool angezeigt, vgl. Screenshot in Abbildung 17. Nach gleichem Schema lassen sich auch die in dem vorliegenden Bericht diskutierten Metriken werkzeugtechnisch umsetzen.

Abbildung 17 Screenshot des VEIA-Prototyps: Metrikanbindung.



Literatur

[BDW98] *Briand*, Lionel C. ; *Daly*, John W. ; *Wüst*, Jürgen: A Unified Framework for Cohesion Measurement in Object-Oriented Systems. In: *Empirical Software Engineering: An International Journal* 3 (1998), Nr. 1, 65–117. citeseer.ist.psu.edu/briand97unified.html

[BWW99] *Briand*, Lionel C. ; *W.*, Daly. J. ; *Wüst*, Jürgen: A Unified Framework for Coupling Measurement in Object-Oriented Systems. In: *IEEE Trans. on Software Engineering* 25 (1999), January/February, Nr. 1, 91–121. citeseer.ist.psu.edu/briand96unified.html

[DMH01] *Dincel*, Ebru ; *Medvidovic*, Nenad ; *Hoek*, André van d.: Measuring Product Line Architectures. In: *Proc. 4th Int. WS on Software Product-Family Engineering (PFE 2001)* Bd. 2290. Berlin, Germany : Springer-Verlag, 2001, S. 346–352

[DR00] *Duke*, Roger ; *Rose*, Gordon: *Formal Object-Oriented Specification Using Object-Z*. Houndmills, Basingstoke, Hampshire RG21 6XS and London : Macmillan Press, 2000 (Cornerstones of Computing)

[GEKM07] *Große-Rhode*, Martin ; *Euringer*, Simon ; *Kleinod*, Ekkart ; *Mann*, Stefan: Grobentwurf des VEIA-Referenzprozesses / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Januar 2007. – ISST-Bericht 80/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[GKM07a] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Architekturbewertung / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Juli 2007. – ISST-Bericht 82/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[GKM07b] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Entscheidungsgrundlagen für die Entwicklung von Softwareproduktlinien / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Oktober 2007. – ISST-Bericht 83/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[GKM07c] *Große-Rhode*, Martin ; *Kleinod*, Ekkart ; *Mann*, Stefan: Fallstudie »Condition-Based Service«: Modelle für die Bewertung von logischen Architekturen und Softwarearchitekturen / Fraunhofer-Institut für

Software- und Systemtechnik, Abt. Verlässliche technische Systeme. Mollstraße 1, 10178 Berlin, Germany, Oktober 2007. – ISST-Bericht 84/07. Projekt VEIA. http://veia.isst.fraunhofer.de/

[HDM03] *Hoek*, André van d. ; *Dincel*, Ebru ; *Medvidovic*, Nenad: Using Service Utilization Metrics to Assess the Structure of Product Line Architectures. In: *Proc. of the 9th Int. Software Metrics Symposium (METRICS'03)*, 2003, S. 298–308. – Sydney, Australia, September 3-5, 2003

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

[Mis00] *Misic*, Vojislav B.: Coherence equals cohesion-or does it? In: *Proc. 7th Asia-Pacific Software Engineering Conf. APSEC 2000*, 2000, S. 465–469

[Mis01] *Misic*, Vojislav B.: Cohesion is structural, coherence is functional: different views, different measures. In: *Proc. 7th Int. Software Metrics Symposium, METRICS 2001*, 2001, S. 135–144

[Mis06] *Misic*, Vojislav B.: Measuring the Coherence of Software Product Line Architectures / Dep. of CS, University of Manitoba. Wiinipeg, Manitoba, Canada, Juni 2006. – TR 06/03

[MT00] *Medvidovic*, Nenad ; *Taylor*, Richard: A Classification and Comparison Framework for Software Architecture Description Languages. In: *IEEE Transactions on Software-Engineering* 26 (2000), Januar, Nr. 1, S. 70–93