

# Feature-basierte Modellierung und Verarbeitung von Produktlinien am Beispiel eingebetteter Software

<sup>1</sup>Christian Berger, <sup>2</sup>Holger Krahn, <sup>1</sup>Holger Rendel, <sup>1</sup>Bernhard Rumpe

<sup>1</sup>RWTH Aachen  
Lehrstuhl für Software Engineering  
Ahornstraße 55  
52074 Aachen  
<http://www.se-rwth.de>

<sup>2</sup>TU Braunschweig  
Institut für Software Systems Engineering  
Mühlenpfordtstraße 23  
38106 Braunschweig  
<http://www.sse-tubs.de>

**Abstract:** Verwandte Baureihen elektronischer Geräte haben typischerweise bezüglich der eingebetteten Software große Ähnlichkeit zueinander. Durch die Etablierung einer Software-Produktlinie lassen sich sowohl Kosten reduzieren als auch die allgemeine Qualität steigern. In dieser Arbeit wird eine Sprache zur Modellierung und Verarbeitung von Software- und Systemvarianten vorgestellt, die modular durch einbettbare Nebenbedingungen zur Prüfung semantischer Korrektheit über die Semantik der bekannten Feature-Diagramme hinaus erweitert werden kann. Diese Modellierung der Variabilität wird durch etablierte Funktionsnetze zur Beschreibung der logischen Architektur ergänzt.

## 1 Einleitung und Motivation

Eingebettete Software zur Steuerung elektronischer Systeme findet sich in vielen Gegenständen des Alltags. Häufig bietet es sich an, dass Software für die Varianten eines elektronischen Geräts in ihren Ausbaustufen und ihren über die Jahre optimierten Baureihen wiederverwendet wird, da die darin enthaltenen Steuerungskomponenten häufig große Ähnlichkeiten zueinander aufweisen. Das gilt für Drucker genau so wie für Fernseher oder Steuergeräte im Flugzeug oder Auto. Aufgrund wachsenden Kundenanspruchs steigt aber die Komplexität der Software und damit der Aufwand in Wartung und Weiterentwicklung.

Weisen Systemkomponenten und insbesondere Software-basierte Teilkomponenten nach Durchführung einer Domänenanalyse [KCH+90] große Übereinstimmungen und damit eine hohe Ähnlichkeit auf, bietet sich die Einführung einer Software-Produktlinie an [CN02]. Software-Produktlinien erlauben hohe Einsparpotenziale, insbesondere in der Wartung und Pflege von Software-Artefakten und ermöglichen eine strukturierte Wiederverwendung der Software für die identifizierten Varianten [PBL05].

In dieser Arbeit wird die Modellierung von System- und insbesondere Software-Komponenten durch spezielle domänenspezifische Sprachen (engl. domain specific languages – DSLs [DKV00, MHS05, GKR+07]) vorgestellt, die um flexible Regelwerke modular ergänzt werden kann. Die Anwendung und Verarbeitung der Feature-DSL wird am Beispiel eines Navigationssystems vorgestellt und ihr Potenzial im Rahmen eines Software-Entwicklungsprozesses diskutiert, der auf Funktionsnetzen basiert.

## 2 Feature-basierte Modellierungssprache und Verarbeitung

Die in Abbildung 1 dargestellte Feature-DSL enthält sämtliche syntaktischen Elemente eines Feature-Diagramms nach [CE00] in textueller und somit maschinenverarbeitbarer Form. Die entworfene DSL und das dazugehörige Werkzeug basieren auf dem MontiCore-Framework [GKR+06, KRV07, GKR+08, KRV08].

```
01 external Constraint;
02
03 FeatureDiagram = "featurediagram" name:IDENT "{"
04   head:FeatureInTree
05   FeatureInTree*
06   "constraints" "{"
07     (Constraint ";")*
08   "}"
09 ";";
10
11 Feature = name:IDENT (optional:["?"])?;
12
13 interface FeatureInTree;
14
15 AllFeature implements FeatureInTree = name:IDENT "="
16   Feature ("," Feature)* ";";
17
18 AlternativeFeature implements FeatureInTree = name:IDENT "="
19   "alt" "(" Feature ("," Feature)* ")" ";";
20
21 OrFeature implements FeatureInTree = name:IDENT "=" "or" "(" Feature
22   ("," Feature)* ")" ";";
23
24 MultipleFeature implements FeatureInTree = name:IDENT "="
27   "[" lowerbound:INT ".." upperbound:INT "]"
28   "(" Feature ("," Feature)* ")" ";";
```

Abbildung 1: Auszug aus der Grammatik der Feature-DSL.

Die zentrale Regel der Grammatik ist in den Zeilen 3-9 zu finden, in denen eine Beschreibung für Feature-Diagramme durch das Schlüsselwort „featurediagram“ mit einem Namen eingeleitet wird. Anschließend folgen in einem Block, der in geschweiften Klammern gefasst ist, die einzelnen Features, von denen das erste den Kopf des Diagramms bildet (Zeile 4). Von der Grammatik werden folgende Features unterstützt: Atomare Features (Zeile 11), aus mehreren benötigten Subfeatures bestehende Features (Zeile 15), alternativ ausschließende Features (Zeile 18), Or-Features, bei denen mehrere Subfeatures ausgewählt werden können (Zeile 21) und die Möglichkeit aus einer Anzahl an gegebenen Features eine bestimmte Menge zu selektieren (Zeile 24).

Das besondere Merkmal hierbei sind modular einbettbare Subsprachen, hier konkret einer Logik-Sprache, zur Definition von Nebenbedingungen (Zeile 1 und 7). Durch das nicht verfeinerte Nichtterminal „Constraint“ in der obigen Grammatik wird eine Austauschbarkeit und zudem eine Erweiterbarkeit der Feature-Sprache gewährleistet, so dass die oben definierte Grammatik und die darauf aufbauenden Werkzeuge wiederverwendet werden können. In den folgenden Beispielen wird für „Constraint“ eine einfache Aussagenlogik verwendet.

### 3 Feature-basierte Modellierung am Beispiel eines Navigationssystems

Als ein einfaches System, in dem die vorgestellten DSLs eingesetzt werden, wird ein Navigationssystem modelliert. Dieses System ist in grafischer und textueller Form in Abbildung 2 dargestellt.

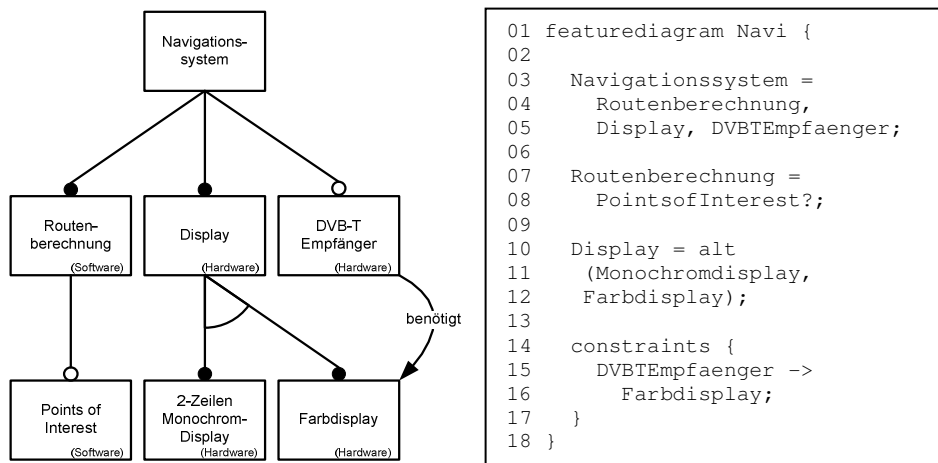


Abbildung 2: Modellierung eines Navigationssystems als Feature-Diagramm.

Das dargestellte Beispiel beschreibt ein vereinfachtes Navigationssystem, von dem vom Diagramm die Teile der Routenberechnung, das Display und ein optionaler DVB-T Empfänger berücksichtigt werden (Zeile 3-5). Die Routenberechnung kann optional durch eine Points of Interest-Datenbank erweitert werden (Zeile 7-8). Beim Display steht alternativ eine Monochrom-Text- oder Farbanzeige zur Auswahl (Zeile 10-12). Bei der Benutzung des DVB-T Empfängers wird letztere zwingend benötigt (Zeile 14-17). Dies ist eine Nebenbedingung, die in der oben erwähnten einfachen Aussagenlogik eingebettet wurde.

Falls es der Kontext erfordert, dass Nebenbedingungen angegeben werden müssen, für die eine einfache Aussagenlogik nicht ausreicht, kann ein anderes Regelwerk verwendet werden. Beispielsweise könnte eine Points of Interest-Datenbank zur Intellectual Property (IP) eines Kunden gehören. Durch die Definition von Constraints in der zur Verfügung stehenden Regelwerk-Sprache kann festgelegt werden, dass diese Komponente nur Produktvarianten dieses Kunden zur Verfügung steht. Dafür ist aber eine komplexeres Regelwerk als die zunächst eingebettete Aussagenlogik nötig, die zusätzlich auf elektronisch verfügbare Vertragswerke zugreift.

Als Erweiterung kann ein Standard- und eine Premium-System des Navigationssystems angegeben werden. Die Beschreibung der Systeme, wie sie im „constraints“-Block auftauchen würde, ist in Abbildung 3 dargestellt.

```
[..]  
01 Standard -> (Routenberechnung & Monochromdisplay);  
02 Premium -> (Routenberechnung & PointsofInterest & Farbdisplay &  
    DVBTEmpfaenger);  
[..]
```

Abbildung 3: Definition von Systemen.

## 4 Funktionsnetze

Eine praktische Anwendung der Modellierungssprache ergibt sich im Zusammenhang mit Funktionsnetzen durch Sichtenbildung, wie sie in [GHK+08a, GHK+08b] beschrieben werden. Sie stellen eine Möglichkeit zur Modellierung der logischen Architektur eines Softwaresystems dar. In [GKPR08] wird beschrieben, wie der Sichtenmechanismus dazu verwendet werden kann, Varianten innerhalb einer automotiven Baureihe zu modellieren. Hier soll nun aufgezeigt werden, welche Ergänzungen zur Etablierung einer Software-Produktlinie notwendig sind.

Der bisherige Ansatz erlaubt nicht die explizite Formulierung und Überprüfung von Nebenbedingungen. Die modulare Erweiterung der Feature-DSL ergänzt daher eine einfache Aussagenlogik.

Bei der Anwendung der Funktionsnetze für eine Produktlinie kann diese als 150%-Modell modelliert werden. Dieses 150%-Modell enthält alle Teile, aus denen ein mögliches Produkt bestehen kann, das folglich einen Ausschnitt dieses Modells darstellt. Die einzelnen Features entsprechen ein oder mehreren Blöcken des Funktionsnetzes. Eine Konfiguration des Feature-Diagramms entspricht einer bestimmten Sicht auf das Funktionsnetz. So werden dann die einzelnen Produkte als Sichten auf dieses Gesamtmodell beschrieben und enthalten nur die für die Variante relevanten Blöcke.

Bei der Verwendung für Software-Produktlinien zeigt sich jedoch, dass bei neuen Varianten gezielt Elemente vom 150%-Modell abweichen sollen und daher nicht über den bisherigen Sichtenmechanismus beschrieben werden können. Daher muss die vorhandene Notation so erweitert werden, dass diese Differenzen beschrieben werden können und in die vorhandene Methodik einbettbar sind. Diese Differenzen werden basierend auf einer Referenzarchitektur angegeben, wie sie bei Software-Produktlinien verwendet wird. An diesem Punkt werden wir in Zukunft arbeiten.

## 5 Verwandte Arbeiten

Eine Grammatik für Feature-Diagramme ist in [DK01] zu finden. An einem Beispiel wird die Benutzung der Sprache verdeutlicht und ein Datenmodell für das Feature-Diagramm entworfen. Die Grammatik bietet jedoch weniger Möglichkeiten als die hier vorgestellte und ist an eine feste Definition von Nebenbedingungen gebunden.

[CE00] beschäftigt sich ebenfalls mit Feature-Diagrammen und deren Nebenbedingungen. Die hier vorgestellte DSL baut auf dieser Darstellung auf. In weiteren Veröffentlichungen werden Erweiterungen wie explizit definierbare Kardinalitäten zu den Diagrammen vorgestellt [CHE05].

In [PBL05] wird im Rahmen der Software-Produktlinien auf Variabilität eingegangen. Hier wird ein Variabilitätsmodell vorgestellt, das dem hier verwendeten ähnlich ist. Der größte Unterschied im Vergleich zu dem hier verwendeten Modell besteht darin, dass dieses explizit auf Variationspunkte und Varianten eingeht und nicht auf Features.

Ein Ansatz, Feature-Diagramme mittels UML-Klassendiagrammen darzustellen, wird in [Gom04] gezeigt. Sämtliche Nebenbedingungen sind im Diagramm angegeben und müssen nicht textuell festgehalten werden, was jedoch bei komplizierteren Konstellationen unübersichtlich werden kann. Austausch- und Erweiterungsmöglichkeiten bestehen im Rahmen der UML 2.0 [OMG07a, OMG07b].

Auch in anderen Arbeiten werden Sichten benutzt, um mögliche Produkte innerhalb einer Produktlinie zu modellieren. So stellt das Fraunhofer ISST in [GKM07] das VEIA-Projekt vor, welches ebenfalls ein 150%-Modell nutzt, um mehrere mögliche Systeme zu beschreiben. Darauf aufbauend wird das System aXBench [aXB] entwickelt, das eine Modellierung mittels Eclipse [Ecl] anbietet. Hier wird aber nicht auf die Möglichkeit eingegangen, bestimmte Systeme mittels Feature-Diagrammen zu spezifizieren.

## 6 Zusammenfassung und Ausblick

In diesem Papier wurde eine textuelle, maschinenverarbeitbare Sprache zur Modellierung von Feature-Diagrammen vorgestellt. Die Besonderheit der entworfenen Sprache stellt die Parametrisierung der Feature-Sprache mit einer nicht weiter definierten Constraint-Sprache dar. Diese kann durch Einbettung einer entsprechenden Grammatik festgelegt werden und stellt damit gleichzeitig eine Erweiterung der Sprache und der Werkzeuge zur Sprachverarbeitung dar. Die Anwendbarkeit der erweiterbaren Feature-DSL wurde an einem Beispiel zur Modellierung eines Navigationssystems demonstriert.

Die dargestellte Feature-DSL kann zum Beispiel bei der Umstellung von eingebetteter Software zur Steuerung elektronischer Geräte auf eine Software-Produktlinie die durch eine Domänenanalyse identifizierten Software- und Systemvarianten modellieren. Dazu lassen sich Funktionsnetze zur Beschreibung der logischen Architektur der Produktlinien und Sichten zur Modellierung der konkreten Varianten einsetzen. Derzeit werden Möglichkeiten zur Erweiterung der Notation untersucht, um Abweichungen der Varianten von einer vorgegebenen Referenzarchitektur der Produktlinie beschreiben zu können, die die vorhandene Sichtenbildung ergänzen sollen.

Die Integration der modularen Feature-DSL in einen Software- und Systementwicklungsprozess wird derzeit im Rahmen der Einführung einer Software-Produktlinie bei einem Hersteller von Steuergeräten untersucht. Hierbei zeichnet sich die modular erweiterbare Feature-DSL als ein vielversprechender Weg ab, unter anderem vorhandene Versionierungssysteme und System-Architekturen zur Generierung funktionsfähiger Software zur Übertragung auf ein Steuergerät zu integrieren.

## Literaturverzeichnis

- [aXB] Fraunhofer-Institut für Software- und Systemtechnik: *aXBench*. <http://axbench.isst.fraunhofer.de/>, 2009-03-24.
- [CE00] Czarnecki, K.; Eisenecker, U. W.: *Generative Programming: Methods, Tools, Applications*. Addison-Wesley, 2000.
- [CHE05] Czarnecki, K.; Helsen, S.; Eisenecker U. W.: *Formalizing Cardinality-based Feature Models and their Specialization*. In: Software Process Improvement and Practice, Special Issue of Best Papers from SPLC04, 2005.
- [CN02] Clements, P.; Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston 2001.
- [DKV00] van Deursen, A.; Klint, P., Visser, J.: *Domain-specific languages: an annotated bibliography*. *SIGPLAN Not., ACM*, 2000, 35, 26-36.
- [DK01] van Deursen, A.; Klint, P.: *Domain Specific Language Design Requires Feature Descriptions*. In: Journal of Computing and Information Technology 2001.
- [Ecl] Eclipse Foundation: *Eclipse*. <http://www.eclipse.org/>, 2009-03-24.
- [GHK+08a] Grönninger, H.; Hartmann, J.; Krahn, H.; Kriebel, S.; Rothhardt, L.; Rumpe, B.: *Modelling Automotive Function Nets with Views for Features, Variants, and Modes*. In: 4th European Congress ERTS – Embedded Real Time Software (2008).

- [GHK+08b] Grönniger H.; Hartmann J.; Krahn, H.; Kriebel, S.; Rothhardt, L.; Rumpe, B.: *View-Centric Modeling of Automotive Logical Architectures*. In: Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IV. Informatik-Bericht 2008-02, CFG-Fakultät, TU Braunschweig, 2008.
- [GKM07] Große-Rhode, M.; Kleinod, E.; Mann, S.: *Entscheidungsgrundlagen für die Entwicklung von Softwareproduktlinien*. Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche technische Systeme, 2007, ISST-Bericht 83/07, 2007.
- [GKR+07] Grönniger, H.; Krahn, H.; Rumpe, B.; Schindler, M.; Völkel, S.: *Textbased Modeling*. 4th International Workshop on Software Language Engineering, 2007.
- [GKR+08] Grönniger, H.; Krahn, H.; Rumpe, B.; Schindler, M. & Völkel, S.: *MontiCore: A Framework for the Development of Textual Domain Specific Languages*. 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, Companion Volume, 2008, 925-926.
- [GKPR08] Grönniger, H.; Krahn, H.; Pinkernell, C.; Rumpe, B.: *Modeling Variants of Automotive Systems using Views*. In: Tagungsband Modellierungs-Workshop MBEFF: Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen (2008).
- [GKR+06] Grönniger, H.; Krahn, H.; Rumpe, B.; Schindler, M.; Völkel, S.: *MontiCore 1.0 – Ein Framework zur Erstellung und Verarbeitung domänenspezifischer Sprachen*. Technischer Report, Informatik-Bericht 2006-04, Institut für Software Systems Engineering, TU Braunschweig, 2006.
- [Gom04] Gomaa, H.: *Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures*. Addison-Wesley Object Technology Series, 2004.
- [KCH+90] Kang, K.; Cohen, S.; Hess, J.; Nowak, W.; Peterson, S.: *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1990.
- [KRV07] Krahn, H.; Rumpe, B.; Völkel, S.: *Integrated Definition of Abstract and Concrete Syntax for Textual Languages*. Proceedings of Models 2007, 2007, 286-300.
- [KRV08] Krahn, H.; Rumpe, B.; Völkel, S.: *MontiCore: Modular Development of Textual Domain Specific Languages*. Proceedings of Tools Europe, 2008.
- [MHS05] Mernik, M.; Heering, J., Sloane, A. M: *When and how to develop domain-specific languages*. ACM Comput. Surv., ACM Press, 2005, 37, 316-344.
- [PBL05] Pohl, K; Böckle, G.; van der Linden, F.: *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.
- [OMG07a] Object Modeling Group: *Unified Modeling Language: Superstructure Version 2.1.2*, November 2007.
- [OMG07b] Object Modeling Group: *Unified Modeling Language: Infrastructure Version 2.0*, November 2007.