Automatische Synthese von Familienmodellen durch Analyse von block-basierten Funktionsmodellen

Sönke Holthusen[1], Peter Manhart[2], Ina Schaefer[1],
Sandro Schulze[1], Christian Singer[1], David Wille[1]

[1]Technische Universität Braunschweig
Institut für Softwaretechnik und Fahrzeuginformatik
http://www.tu-bs.de/isf
{s.holthusen, i.schaefer, sanschul, c.singer, d.wille}@tu-bs.de

[2] Daimler AG
http://daimler.com
peter.manhart@daimler.com

Abstract: Um die steigende Komplexität in der Softwareentwicklung in der Automobilindustrie zu beherrschen, werden immer öfter modellbasierte Entwicklungsansätze, zum Beispiel mit block-basierten Modellen, genutzt. Modellvarianten werden dabei nicht selten durch Kopieren und Anpassen erzeugt. Dies führt zu einer Modellfamilie mit unterschiedlichen, aber verwandten Modellen und erschwert die Wartung, Qualitätssicherung und Weiterentwicklung der Modelle. In diesem Aufsatz wird ein Ansatz präsentiert, mit dem Unterschiede und Gemeinsamkeiten einer Modellfamilie automatisch gefunden und in ein Familienmodell überführt werden können. Das Familienmodell ermöglicht eine einheitliche Darstellung von Gemeinsamkeiten und Unterschieden aller Modelle und erleichtert so die Wartung, das Refactoring und die Qualitätssicherung der Modellvarianten.

1 Einleitung

Ein Großteil der heute entwickelten Software kommt in komplexen eingebetteten Systemen zum Einsatz. Diese Systeme müssen zumeist hohe Ansprüche an Zuverlässigkeit und Sicherheit erfüllen. Um die Systemkomplexität und die hohen Anforderungen zu bewältigen, ist in den letzten Jahren die modellbasierte Software-Entwicklung in den Fokus gerückt. Speziell in der Automobilbranche hat ein Großteil der dort eingesetzten Softwaresysteme modellbasierte Anteile.

Da es in der Automobilbranche eine Vielzahl von unterschiedlichen Varianten (z.B. unterschiedliche Modelle eines Autos) gibt, besteht die zusätzliche Anforderung, dass die entwickelten Softwaresysteme schnell und zuverlässig an jede einzelne Fahrzeugvariante angepasst werden können. Da die einzelnen Varianten häufig eine große Ähnlichkeit aufweisen, ist ein weit verbreiteter Ansatz das Kopieren und Anpassen bestehender Modelle (*Clone-and-Own*), aus denen dann der eigentliche Quellcode (und damit die Soft-

ware) generiert wird. Allerdings ist diese Vorgehensweise unter mehreren Aspekten als problematisch zu betrachten. Zum einen geht bei diesem Prozess der Wiederverwendung die Information über die Ähnlichkeit zwischen den Modellen verloren oder ist bei dem jeweiligen Entwickler vorhanden, aber nicht ausreichend dokumentiert. Daraus resultiert ein fehlendes Wissen über die Variabilität und die Abhängigkeiten zwischen den Modellvarianten. Weiterhin ist es im Falle eines Fehlers schwierig nachzuverfolgen, in welchen anderen Modellvarianten dieser Fehler ebenfalls enthalten ist. Ebenso ist die Propagierung von Änderungen durch das fehlende Wissen über die Modellabhängigkeiten nicht möglich. Außerdem ist eine effizienter Test der Modelle unmöglich, da jedes Modell einzeln vollständig getestet werden muss, wenn das Wissen über gemeinsame Anteile fehlt.

Um eine effiziente modellbasierte Entwicklung von variantenreichen Softwaresystemen zu ermöglichen und die genannten Probleme zu verringern, ist es notwendig, die Information über Variabilität zwischen Modellen variantenübergreifend darzustellen. Dabei ist zu berücksichtigen, dass die Gemeinsamkeiten aber auch Unterschiede einzelner Varianten erhalten bleiben müssen.

In dieser Arbeit stellen wir einen Ansatz vor, der aus einer Menge ähnlicher, block-basierten Funktionsmodelle automatisch ein übergeordnetes *Familienmodell* erstellt. Dieses beinhaltet die Information über die Gemeinsamkeiten und Unterschiede zwischen den Modellvarianten und ermöglicht es zudem, die Unterschiede genauer zu beschreiben (z.B. als alternativ oder optional). Familienmodelle repräsentieren Variabilität im Lösungsraum, in dem die Modelle realisiert werden, im Gegensatz zu Merkmalmodellen (auch Feature-Modelle) [CE00], die im Problemraum angesiedelt sind. Merkmalmodelle geben eine logische Sicht auf die Domäne und sind daher zur Strukturierung der Realisierung einer Modellfamilie nicht geeignet. Neben der Konzeption des Ansatzes zur automatischen Erzeugung von Familienmodellen stellen wir eine erste, prototypische Implementierung vor und demonstrieren damit die Umsetzbarkeit des vorgeschlagenen Ansatzes. Das resultierende Familienmodell ermöglicht es, Zusammenhänge zwischen einzelnen Modellvarianten zu erkennen und diese Information für die Wartung zu nutzen. Außerdem kann das Familienmodell für die Erstellung zukünftiger Modelle hilfreiche Ansatzpunkte liefern.

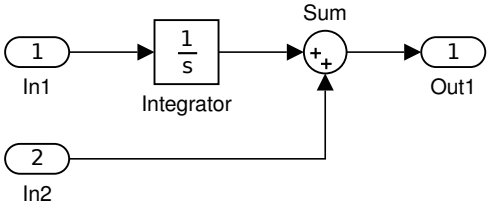
Dieser Aufsatz ist wie folgt aufgebaut: In Abschnitt 2 stellen wir an einem einführendem Beispiel block-basierte Funktionsmodelle und Familienmodelle vor. In Abschnitt 3 beschreiben wir das Problem, das unserer Ansatz lösen soll. Abschnitt 4 führt dann das Konzept der automatischen Synthese von Familienmodellen ein, das in Abschnitt 5 durch eine Implementierung unterstützt wird. In Abschnitt 6 diskutieren wir verschiedene Aspekte zur Erweiterung unseres Ansatzes. Abschließend grenzen wir in Abschnitt 7 unseren Ansatz von bestehenden Ansätzen ab. Abschnitt 8 umfasst eine Zusammenfassung und einen Ausblick auf zukünftige Arbeiten.

2 Motivierendes Beispiel

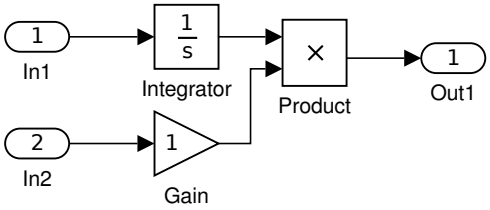
Die modellbasierte Entwicklung verwendet unter anderem Funktionsmodelle, die als block-basierte Datenflussdiagramme dargestellt sind. Beispiele für Werkzeuge, welche

diese Modellierungstechnik unterstützen, sind MATLAB/Simulink[1], ASCET[2] oder SCADE[3]. Ein *Block* hat *Ein- und Ausgänge* und implementiert eine *Funktion*, die aus den Daten an den Eingängen die Daten an den Ausgängen berechnet. Blöcke können durch die Nutzung von *Subsystemen* hierarchisch organisiert werden. Innerhalb eines Modells findet die Kommunikation über gerichtete *Verbindungen* zwischen Blöcken und Ports statt.

Als erstes wird zum Beispiel das in Abbildung 1a gezeigte Modell entwickelt. Es enthält Blöcke vom Typ *Integrator* und *Sum*. Das Modell hat zwei Eingabe- und einen Ausgabe-Port, die mit den Blöcken verbunden sind. Aus diesem Modell könnte durch *Clone-and-Own* das Modell in Abbildung 1b entstanden sein. Hier könnte z.B. ein anderer Sensor am Eingabe-Port *In2* verwendet werden, dessen Eingabe durch den hinzugefügten *Gain*-Block aufbereitet werden muss. Dadurch ist es notwendig, das Zusammenführen der Daten durch einen *Product*-Block anstatt des *Sum*-Blockes durchzuführen. So erhalten wir zwei Modellvariaten, die sich strukturell ähnlich sind, allerdings funktionale Unterschiede aufweisen.



(a) Modellvariante 1



(b) Modellvariante 2

Abbildung 1: Modellfamilie in MATLAB/Simulink

Das Nutzen von *Clone-and-Own* ist bei wenigen Modellvarianten noch vergleichsweise unkritisch. Bei zunehmender Variantenkomplexität wird es allerdings immer schwerer die Modelle zu warten. Die automatische Variantenanalyse hat das Ziel, Variationspunkte in den vorliegenden Modellvarianten zu erkennen und darauf aufbauend ein *Familienmodell* zu generieren, das die Gemeinsamkeiten und Unterschiede der Modellvarianten repräsentiert. Dadurch ist eine erhebliche Erleichterung bei der Erkennung von Variantenstrukturen in Funktionsmodellen zu erwarten. Zudem wird eine (Re-)Strukturierung des Variantenraumes erleichtert.

[1] http://www.mathworks.com
[2] http://www.etas.com/en/products/ascet_software_products.php
[3] http://www.esterel-technologies.com/products/

3 Zielsetzung

Die automatische Erzeugung von Familienmodellen soll es ermöglichen, aus einer Menge an unterschiedlichen, aber verwandten block-basierten Funktionsmodellen eine Repräsentation zu erzeugen, in der die Variabilität der Modellfamilie aufgezeigt wird. In dieser Arbeit befassen wir uns mit den block-basierten Funktionsmodelle, wie sie z.B. in MATLAB/Simulink vorkommen. Andere Modelltypen, zum Beispiel Zustandsautomaten oder physikalische Modelle, werden zur Zeit noch nicht weiter betrachtet.

Variabilität zwischen block-basierten Funktionsmodellen lässt sich in Form von alternativen und optionalen Teilmodellen ausdrücken. Als Teilmodell betrachten wir dabei eine Menge an zusammenhängenden Blöcken inklusive der ein- und ausgehenden Verbindungen (siehe Abbildung 3a). Sind zwei Teilmodelle alternativ, bedeutet dies, dass beide Teilmodelle gegeneinander austauschbar sind. Dafür ist es notwendig, dass die Schnittstellen der Teilmodelle, also die Anzahl an ein- und ausgehenden Verbindungen (siehe Abbildung 3b), identisch sind. Für eine Austauschbarkeit ist es weiterhin notwendig, dass der Kontext der Teilmodelle identisch ist. Der Kontext beschreibt die Menge an Blöcken, die über ein- und ausgehende Verbindungen direkt mit dem Teilmodell verbunden sind (siehe Abbildung 3c). Der gleiche Kontext bestimmt die Stelle im Modell, an der alternative Teilmodelle eingesetzt werden können. Eine solche Stelle im Modell, an der Teilmodelle ausgetauscht werden können, nennen wir Variationspunkt.

Das Ziel der automatischen Erzeugung von Familienmodellen ist die Identifikation von Variationspunkten in Modellen und der austauschbaren Teilmodelle (Alternativen). Außerdem sollen optionale Teilmodelle gefunden werden, die nicht in allen Modellvarianten vorkommen. Ein optionales Teilmodell kann dabei als Alternative zwischen einem Teilmodell und einem leeren Teilmodell aufgefasst werden. Sind die Variationspunkte in der Modellfamilie identifiziert, können die Modelle in ein Familienmodell überführt werden.

Ein Familienmodell stellt die Gemeinsamkeiten und Unterschiede von Funktionsmodellen in Form eines *150%-Modells* dar, in dem Modellteile als optional oder alternativ gekennzeichnet sind. Für die beiden Modelle in Abbildung 1 ergibt sich das Familienmodell in Abbildung 2, wie es in pure::variants[4] dargestellt wird. Zu sehen ist, dass der *Sum*-Block aus dem Modell in Abbildung 1a und der *Product*-Block aus dem Modell in Abbildung 1b Alternativen zueinander sind. Diese werden durch den Doppelpfeil (⇔) als solche gekennzeichnet. Der *Gain*-Block aus dem Modell in Abbildung 1b ist ein optionaler Block, da er nur in diesem Modell vorkommt, und wird durch ein Fragezeichen gekennzeichnet.

4 Vorgehensweise

Um die im letzten Abschnitt eingeführten Variationspunkte zwischen Modellvarianten zu erkennen, nutzen wir eine automatische Synthese von Familienmodellen. Abbildung 4 zeigt den Ablauf der Synthese, die für jedes Modell in drei Schritten erfolgt: (1) Zuordnen,

[4] http://www.pure-systems.com/

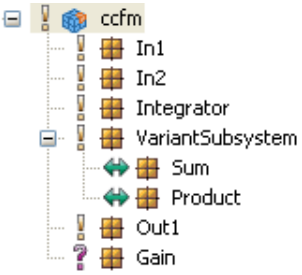
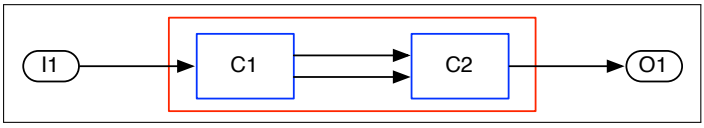
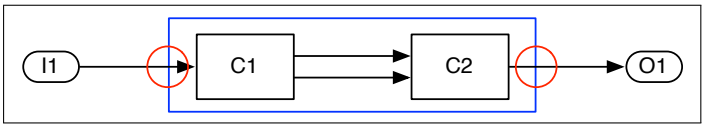


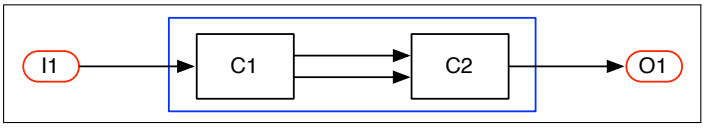
Abbildung 2: Das pure::variants Familienmodell der Modelle aus Abbildung 1



(a) Ein Teilmodell (C1 und C2)



(b) Die Schnittstelle des Teilmodells C1 und C2



(c) Der Kontext des Teilmodells C1 und C2

Abbildung 3: Beispiele für *Teilmodell, Schnittstelle und Kontext*

(2) Entscheiden, und (3) Erweitern. Im Folgenden erklären wir die einzelnen Schritte der Synthese im Detail und erläutern die Anwendung auf eine Menge von Modellvarianten. Unser Ansatz ist zunächst auf flache block-basierte Funktionsmodelle ohne hierarchische Strukturierung beschränkt.

4.1 Zuordnen

In der ersten Phase werden zunächst die Blöcke beider Modellvarianten verglichen und der strukturellen Ähnlichkeit nach zugeordnet. Dazu wird jeder Block der ersten Modellvariante mit jedem noch nicht zugeordneten Block der zweiten Modellvariante verglichen und eine Ähnlichkeit berechnet. Die strukturelle Ähnlichkeit zweier Blöcke setzt sich dabei aus zwei Faktoren zusammen. Der erste ist die gleiche Schnittstelle. Stimmen diese nicht überein, werden die Blöcke als nicht ähnlich betrachtet. Wenn zwei Blöcke die glei-

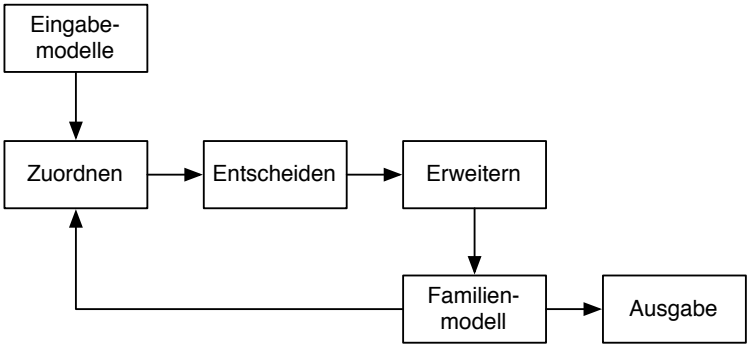


Abbildung 4: Die Schritte der automatische Synthese von Familienmodellen

che Schnittstelle haben, wird die Nachbarschaft der Blöcke betrachtet. Die Nachbarschaft eines Blocks bezeichnet den Kontext des Blocks und die weiteren mit dem Kontext verbundenen Blöcke. Je mehr der Blöcke in der Nachbarschaft gleich sind, desto höher ist die Nachbarschaftsähnlichkeit. Je weiter dabei ein Block vom zu vergleichenden Block entfernt ist, desto weniger Gewicht hat der Block in der Nachbarschaftsähnlichkeit. In diesem Vergleich für die Nachbarschaftsähnlichkeit bedeutet Gleichheit, dass sowohl die Schnittstelle als auch der Blocktyp gleich sein müssen.

Die Blöcke mit der höchsten Ähnlichkeit werden als Paar übernommen und aus der Liste der noch nicht zugeordneten Blöcke entfernt. Durch dieses Vorgehen entsteht eine Menge von zugeordneten Blöcke. Alle Blöcke, für welche dabei kein Gegenstück gefunden werden konnte, werden als nicht-zugeordnet markiert.

Für die Modelle aus Abbildung 1 ergeben sich in dieser Phase die folgenden Paare: Der Integrator-Block aus Modellvariante 1 wird dem Integrator-Block aus Modellvariante 2 zugeordnet und der Sum-Block aus Modellvariante 1 dem Product-Block aus Modellvariante 2. Der Gain-Block aus Modellvariante 2 würde als nicht zugeordnet markiert.

4.2 Entscheiden

Die zugeordneten Blöcke werden nun aufgeteilt auf gleiche und alternative Paare. Bei gleichen Paaren handelt es sich um Blöcke, die neben der strukturellen Ähnlichkeit auch noch den gleichen Blocktyp haben. Bei einem gleichen Paar ist keine weitere Aktion nötig, da der Block bereits im Familienmodell vorhanden ist. Sind die Blöcke des Paares nicht gleich, handelt es sich um Alternativen, die durch einen neuen oder erweiterten Alternativ-Block im Familienmodell dargestellt werden, der beide Blöcke beinhaltet. Ein erweiterter Alternativ-Block wird notwendig, wenn in einem vorherigen Durchlauf bereits Variabilität erkannt und im Familienmodell dargestellt wurde.

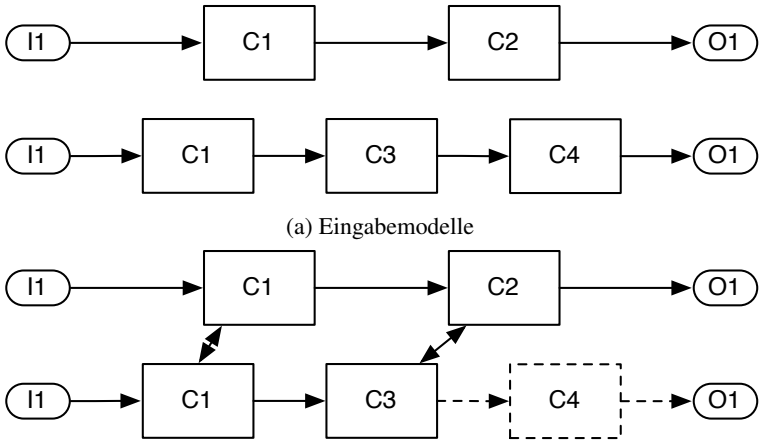
In unserem Beispiel handelt es sich bei dem Paar der Integrator-Blöcke um ein gleiches Paar. Für das Familienmodell ist keine weitere Aktion nötig, da der Block bereits vorhan-

den ist. Das zweite Paar aus Sum- und Product-Block ist nicht gleich und muss deshalb durch eine neue Alternative im Familienmodell dargestellt werden.

4.3 Erweitern

Bei jeder Erzeugung einer Alternative (in Form eines alternativen Subsystem-Blocks) werden die beiden zueinander alternativen Blöcke falls möglich zu größeren Teilsystemen erweitert. Hierzu werden - ausgehend von dem alternativen Block - in beiden Modellen alle Verbindungen zu benachbarten Blöcke betrachtet und alle benachbarten nicht zugeordneten Blöcke, sowie andere alternative Blöcke mit in die gerade neu erzeugte Alternative übernommen. Dies wird über mehrere verbundene Blöcke durchgeführt, bis auf jedem Pfad ein Block eines nicht-alternativen Paares, d.h. eines Paares mit gleichem Blocktyp, oder ein bereits behandelte Block erreicht wurde.

Abbildung 5a zeigt zwei einfache Modelle. In der Zuordnen-Phase wird der Block C1 des ersten Modells dem Block C1 des zweiten Modells zugeordnet, sowie der Block C2 dem Block C3 (siehe Abbildung 5b). In der Entscheiden-Phase wird festgestellt, dass das Paar der Blöcke (C1,C1) gleich ist und das Paar (C2, C3) eine Alternative. In der Erweitern-Phase wird versucht, nicht zugeordnete Blöcke (in diesem Fall C4) unterzubringen, indem vorhandene Alternativen erweitert werden. Die Blöcke C2 und C3 sind bereits Alternativen. Da der Block C4 mit dem Block C3 verbunden ist, wird dieser Block ebenfalls in die Alternative integriert. Nach diesem Schritt sind keine nicht zuordneten Blöcke mehr vorhanden.



(a) Eingabemodelle

(b) Modelle nach dem Zuordnen

Abbildung 5: Beispiel für das Erweitern von Alternativen

Abschließend werden die übrigen nicht zugeordneten Blöcke als optionale Alternativen in das Familienmodell übernommen. Auch hier werden benachbarte nicht zugeordne-

te Blöcke in die entsprechende Option mitaufgenommen, so dass mehrere benachbarte Blöcke ohne Partner als ein optionales Teilmodell zusammengefasst werden. In unserem Beispiel aus Abbildung 1 ist der einzelne Gain-Block in Modellvariante 2 übrig. Dieser wird als Option in das Familienmodell übernommen (siehe Abbildung 2).

4.4 Familien-Mining mit mehreren Modellvarianten

Die vorhergehenden Abschnitte haben die automatische Synthese eines Familienmodells aus zwei Modellvarianten beschrieben. Das erzeugt Familienmodell liegt in dem gleichen Format wie die Eingabemodelle vor und kann daher als Ausgangsmodell für einen weiteren Durchlauf genutzt werden. Bereits vorhandene, in einem früheren Durchgang erkannte Variabilität wird in Schritt 2, Entscheiden, erkannt. Handelt es sich bei einem der zugeordneten Blöcke um eine bereits erkannte Alternative wird diese um den neuen Block erweitert. Eine Option wird zu einer optionalen Alternative. Das entstandene Familienmodell wird so lange erweitert, bis alle Eingabemodelle verarbeitet sind. Ist das Familienmodell vollständig, kann es ausgegeben werden.

5 Implementierung

Die Implementierung des vorgestellten Ansatzes zur automatischen Erzeugung von Familienmodellen ist durch ein Eclipse-Plugin realisiert. Dies basiert auf einem EMF-Metamodell und nutzt *EMFCompare* für den Vergleich von Modellen. Abbildung 6 zeigt die Architektur der Implementierung. Für die prototypische Implementierung verwenden wir Funktionsmodelle in MATLAB/Simulink. Die zu verarbeitenden Funktionsmodelle werden durch den Simulink-Parser von ConQAT[5] eingelesen und in eine Instanz unseres EMF-Metamodells überführt. Dieses Metamodell ist sehr stark an das von ConQAT genutzte angelehnt, um das Überführen von von ConQAT eingelesenen Simulink-Modellen in unser Metamodell zu vereinfachen. Das erste eingelesene Modell wird als Startmodell festgelegt, und alle anderen Modelle werden nacheinander mit diesem Startmodell zusammengeführt.

Der Algorithmus zum Zuordnen von Blöcken (vgl. Schritt 1 in Unterabschnitt 4.1) ist mit EMFCompare realisiert. EMFCompare nutzt zur Berechnung der strukturellen Ähnlichkeit zwischen Blöcken unterschiedlicher Modelle eine *MatchEngine*. Eine MatchEngine wird durch ein System von Regeln realisiert. Es ist möglich, neue Regeln hinzuzufügen und die Gewichte der einzelnen Regeln anzupassen. Für das Erzeugen von Familienmodellen haben wir eine angepasste MatchEngine erstellt. Folgende Regeln wurden dazu von uns im Prototypen implementiert:

- **MatchInterface** - Vergleicht wie in Unterabschnitt 4.1 beschrieben, ob die Schnittstellen zweier Blöcke übereinstimmen.

[5] https://www.conqat.org/

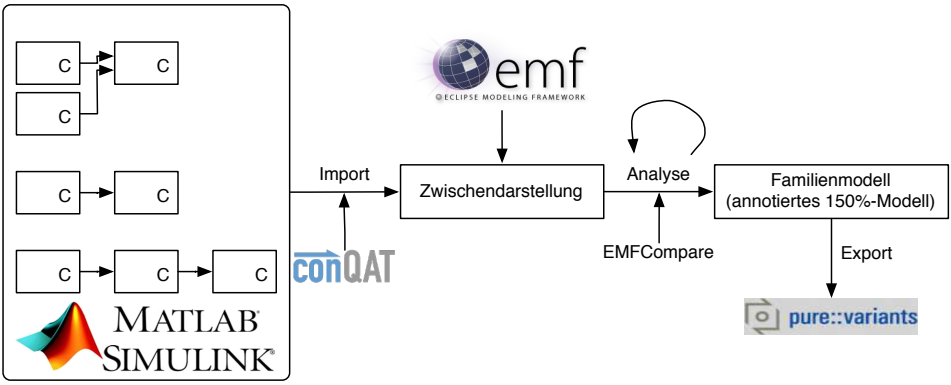


Abbildung 6: Architektur der Implementierung

- **MatchNeighborhood** - Berechnet wie in Unterabschnitt 4.1 beschrieben die Nachbarschaftsähnlichkeit.

Unter Verwendung dieser MatchEngine erzeugt EMFCompare Listen der gefundenen Blockpaare sowie der nicht zugeordneten Blöcke. Die Liste der Paare wird daraufhin durchlaufen und der Blocktyp der Blöcke jedes Paares überprüft (vgl. Unterabschnitt 4.2). Haben die beiden Blöcke einen unterschiedlichen Blocktyp, so wird eine Alternative erzeugt und anstelle des entsprechenden Blockes im Startmodell eingefügt. Für die Darstellung von Alternativen wird ein *VariableSubsystem*-Block aus MATLAB/Simulink genutzt. Dieser erlaubt es, eine Menge an Subsystemen zu hinterlegen und zur Konfigurationszeit eines dieser Subsysteme zur Nutzung auszuwählen. Da dieser Block auch in manuell erzeugten Modellen vorkommen kann, wird ein interner Parameter gesetzt, um anzuzeigen, dass das *VariableSubsystem* während der Synthese des Familienmodells künstlich erzeugt wurde.

Wurden auf diese Weise alle eingegebenen Modelle zusammengeführt, so ist das Familienmodell vollständig. Dieses wird im letzten Schritt in eine Repräsentation des pure::variants Familienmodell überführt und abgespeichert.

6 Diskussion

Mit dem bisher realisierten Ansatz können eine Reihe wichtiger Anwendungsfälle der Praxis abgebildet werden. Doch ist bereits aus den bisherigen Untersuchungen auch Weiterentwicklungspotenzial erkennbar. In diesem Abschnitt diskutieren wir einige dieser Aspekte, die für die automatischen Synthese von Familienmodellen im Weiteren relevant sind.

Wahl der Reihenfolge der Zusammenführung Die Reihenfolge, in der die Modelle dem Synthese-Prozess zugeführt werden, kann einen Einfluss auf die Erkennung von Va-

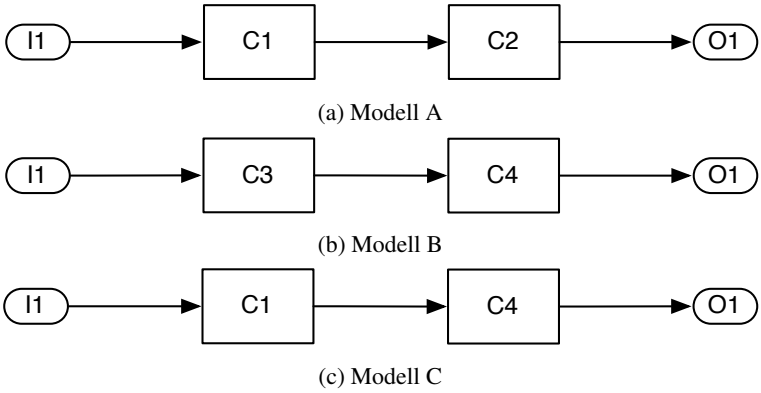


Abbildung 7: Beispiel zur Veranschaulichung der Reihenfolge bei der Zusammenführung

riabilität haben. Sind keine Gemeinsamkeiten vorhanden, werden beide Modelle komplett als Alternativen erkannt. Dies macht eine spätere Zusammenführung schwierig, wenn ein Modell, das Elemente aus den beiden ersten Modellen enthält, hinzugefügt werden soll. Abbildung 7 zeigt drei einfache Modelle mit je zwei Blöcken. Wird die Zusammenführung in der Reihenfolge Modell A, Modell B, Modell C durchgeführt, werden im ersten Schritt die Blöcke C1 und C2 als Alternative zu den Blöcken C3 und C4 erkannt. Wird im zweiten Schritt Modell C verarbeitet, werden die Blöcke C1 und C4 aus Modell C als Alternative zu der im ersten Schritt gefundenen Alternative erkannt. Das Ergebnis der automatischen Synthese ist in Abbildung 8 zu sehen.

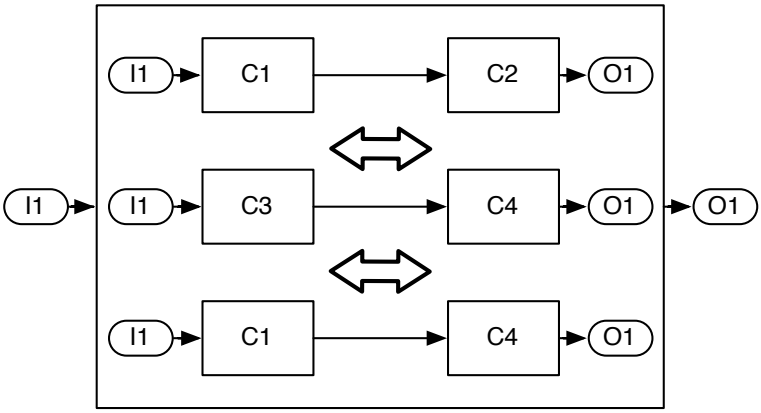


Abbildung 8: Ergebnis der Zusammenführung der Modelle aus Abbildung 7 in der Reihenfolge Modell A, Modell B, Modell C

In unserer Implementierung werden die Modelle in der Reihenfolge der Dateinamen zusammengeführt. Bei diesem Verfahren kann es zu dem oben beschriebenen Problemen kommen. Ist der Zeitpunkt der Erstellung oder der letzten Modifikation bekannt, kann die-

ser zur Bestimmung der Reihenfolge herangezogen werden. Eine weitere Möglichkeit ist das Berechnen einer Ähnlichkeit zwischen den ganzen Modellen und das Zusammenführen mit dem jeweils ähnlichsten Modell, wie man auch in unserem Beispiel in Abbildung 7 sehen kann. Die Modelle A und B haben keine gemeinsamen Blöcke, während Modell A und C mit C1 einen gemeinsamen Block haben. Wird dieser Umstand als eine größere Ähnlichkeit zwischen Modell A und Modell C als zwischen Modell A und Modell B gedeutet und wird die Reihenfolge der Zusammenführung nach der Ähnlichkeit zwischen den Modellen festgelegt, ergibt sich die Reihenfolge Modell A, Modell C, Modell B. Beim Zusammenführen von Modell A und C ergibt sich eine Alternative zwischen den Blöcken C2 und C4. Wird das Modell B verarbeitet, ergibt sich mit C1 und C3 eine weitere Alternative. Der Block C4 ist bereits in einer Alternative vorhanden. Dies führt zu dem Familienmodell in Abbildung 9.

Durch eine Optimierung nach dem eigentlichen Synthese-Durchgang könnte das Modell aus Abbildung 8 in das aus Abbildung 9 überführt werden. Ob es effizienter ist, die Eingangsmodelle in einem Vorverarbeitungsschritt der Ähnlichkeit nach zu sortieren oder das durch die Nutzung einer beliebigen Reihenfolge nicht optimal erzeugte Modell zu vereinfachen bleibt noch zu zeigen.

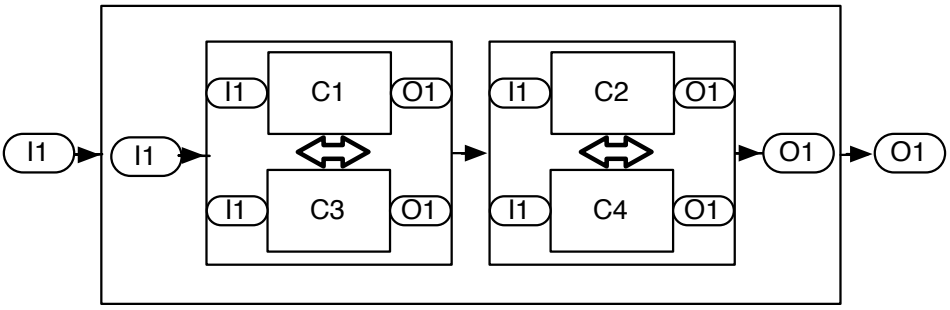
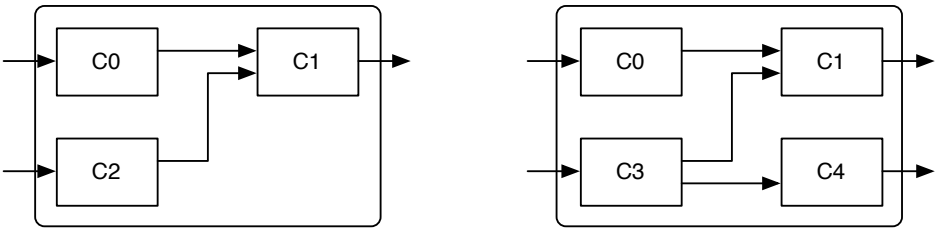


Abbildung 9: Ergebnis der Zusammenführung der Modelle aus Abbildung 7 in der Reihenfolge Modell A, Modell C, Modell B

Granularität der Variationspunkte Ein weiterer Diskussionspunkt, der sich aus den in Abbildung 7 gezeigten Modellen ergibt, ist die Wahl der Größe von Variationspunkten. Jedes Modell mit der gleichen Schnittstelle kann theoretisch als Alternative zueinander gesehen werden. So können Modell A, Modell B und Modell C komplett als Alternativen betrachtet werden, da die Schnittstelle und der Kontext der Modelle übereinstimmen. Das Zusammenführen der Modelle führt zu dem Familienmodell in Abbildung 8. Durch das Anpassen der Reihenfolge der Modellen bei der automatischen Synthese von Familienmodellen kann, wie im vorherigen Abschnitt gezeigt, die Größe der Variationspunkte beeinflusst werden. Durch die Reihenfolge Modell A, Modell C, Modell B erhalten wir bekanntermaßen das in Abbildung 9 gezeigte Familienmodell. In diesem Modell ist mehr Variabilität vorhanden, als durch die drei Eingabemodelle gegeben ist. Zusätzlich zu den drei Eingabemodellen kann nun auch ein Modell erzeugt werden, in dem der Block C3 auf

den Block C2 folgt. Je kleiner die Variationspunkte gewählt werden, desto mehr potentiell mögliche Variabilität kann durch das Familienmodell repräsentiert werden. Es wäre denkbar, die Größe der zu erstellenden Variationspunkte in den Syntheseprozess einzubinden, wenn auch das Finden neuer potentiell möglicher Varianten gewünscht ist.

Variierende Schnittstellen In unserem Ansatz werden Modelle mit variierenden Schnittstellen nicht zugelassen und grundsätzlich als unterschiedlich betrachtet. Jedoch kann es in bestimmten Situationen der Fall sein, dass solche Modelle in ihrer Grundfunktionalität gleich sind. Als Beispiel zeigen wir in Abbildung 10 zwei Modelle, bei denen die Variabilität mit unserem vorgestellten Ansatz nicht ermittelt werden kann. Der Grund sind die sich unterscheidenden Schnittstellen von Modell A in Abbildung 10a und Modell B in Abbildung 10b, da Modell B einen weiteren Ausgang aufweist. Angenommen die Blöcke C0 und C1 sind in beiden Modellen gleich und der einzige Unterschied zwischen den Blöcken C2 und C3 besteht in dem zusätzlichen Ausgang, dann können beide Modelle dieselbe Funktionalität realisieren. So könnte der zusätzliche Block C4 beispielsweise ein weiteres Ergebnis errechnen oder eine Logging-Funktionalität zur Verfügung stellen. Damit wären diese beiden Modelle trotz der veränderten Schnittstelle in ihrer Grundfunktionalität gleich. Um solche Modellvarianten zu erzeugen, ist eine komplexere Metrik für die Zuordnung von ähnlichen Blöcken notwendig, welche wir in zukünftigen Arbeiten entwickeln wollen.

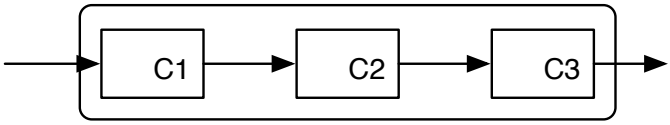


(a) Modell A (b) Modell B

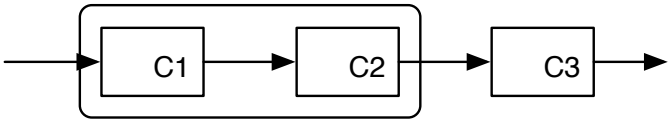
Abbildung 10: Zwei Modelle mit variierenden Schnittstellen

Hierarchien Ein weiterer Mechanismus in block-basierten Funktionsmodellen, der in dieser Arbeit nicht weiter betrachtet wurde, ist die hierarchische Dekomposition von Blöcken durch die Verwendung von Subsystemen. Unser Ansatz kann Ähnlichkeit zwischen Modellen nur erkennen, wenn alle Blöcke in einer Hierarchieebene liegen. So würde in Abbildung 11 erkannt werden, dass die Blöcke *C1* und *C2* in den Modellen A und B zusammenhängend vorkommen. Wird der Block *C3* in Modell B in das Subsystem gezogen, könnte mit *C1, C2, C3* ein größeres Modell erkannt werden. Für die Erkennung wäre es am einfachsten, alle vorhandenen Hierarchien aufzulösen und mit dem dann flachen Modell weiterzuarbeiten. Da es passieren kann, dass durch das Entfernen von Subsystemen zu Änderungen in der Quelltexterzeugung kommen kann (zum Beispiel bei der Nutzung von *Atomic Subsystems*), ist es jedoch nicht immer sinnvoll, Hierarchien zu entfernen. Zudem

werden Hierarchien eingeführt, um die Übersicht zu steigern. Diese Übersicht geht durch die automatische Synthese von Familienmodellen verloren, wenn diese nicht so durchgeführt wird, dass nach der Durchführung Hierarchien wiedergestellt werden.



(a) Modell A



(b) Modell B

Abbildung 11: Zwei Modelle mit unterschiedlichen Hierarchien

Limitierungen des Ansatzes Eine systematische Produktlinienentwicklung ist unserem Ansatz, der Konstruktion einer Modellfamilie aus bestehenden Einzelmodellen, grundsätzlich überlegen. In der Frühphase der Entwicklung von Produktvarianten entstehen in der Praxis jedoch häufig Varianten durch *Clone-and-Own*. Da der Mehraufwand für eine Produktlinienentwicklung in dieser Phase noch nicht offensichtlich ist. Unser Ansatz dient in solchen Situationen dazu, das Refactoring in eine Produktlinie erleichtern.

Weiterhin haben wir unseren Prototypen nur mit kleinen Beispielmodellen getestet. In der Praxis muss der Ansatz jedoch mit wesentlich größeren und komplexeren Modellen funktionieren. Aus diesem Grund werden wir den Ansatz im Zuge eines Industrieprojektes mit einer Modellfamilie von Fahrerassistenzsystemen evaluieren und weiterentwickeln.

7 Verwandte Arbeiten

In der Literatur gibt es verschiedene Ansätze, die sowohl die Ähnlichkeit als auch Unterschiede zwischen Modellen adressieren. Im Folgenden stellen wir einige dieser Ansätze vor und grenzen sie von unserem in dieser Arbeit beschriebenen Ansatz ab.

Deissenboeck et al. haben einen Ansatz entwickelt, um Replikate in MATLAB/Simulink-Modellen, auch bekannt als Modell-Klone, zu erkennen [DHJ+08]. Dabei werden alle Subsysteme bis zur höchsten Hierarchieebene aufgelöst, wobei die einzelnen Elemente (z.B. Blöcke) wesentliche Informationen zur Wiedererkennung beibehalten. Anschließend wird das Modell in einen Graphen überführt, auf dem dann die Klonerkennung durchgeführt wird. Die Blöcke werden dabei zu Knoten und die Verbindungen zu Kanten. Die Funktionalität wurde in dem Tool *CloneDetective* implementiert, welches in ConQAT integriert ist.

Pham et al. haben in ihrem Tool *ModelCD* zwei Algorithmen zur Modell-Klonerkennung, *eScan* und *aScan*, entwickelt [PNN+09]. Während *eScan* sehr ähnlich wie *CloneDetective* arbeitet, erkennt *aScan* auch Klone, die sich leicht unterscheiden (zum Beispiel durch das Kopieren und Anpassen vorhandener Modelle).

Alle oben genannten Ansätze erkennen allerdings nur Gemeinsamkeiten in MATLAB/Simulink-Modellen und geben somit keine Informationen über Unterschiede zwischen Modellen. Im Gegensatz dazu kann der von uns vorgestellte Ansatz auch Unterschiede auffinden und klassifizieren (z.B. als optional oder alternativ).

Neben Ansätzen zur Klonerkennung gibt es Ansätze, die es ermöglichen, Unterschiede zwischen Modellen zu erkennen (*ModelDiff*). Viele dieser Werkzeuge sind kommerziell, zum Beispiel *SiDiff*[6] oder *SimDiff*[7]. Diese Werkzeuge erlauben es zwar, Unterschiede zu erkennen, bieten jedoch keinerlei Informationen über Variabilität zwischen Modellen. Da sie darüber hinaus auch keine Informationen über Gemeinsamkeiten zwischen Modellen enthalten, sind sie ebenfalls ungeeignet für für das Erzeugen von Familienmodellen.

Ryssel et al. haben einen Ansatz entwickelt, der das Ziel verfolgt, eine Bibliothek von ähnlichen Subsystemen anzulegen, um eine spätere Wiederverwendung in unterschiedlichen Projekten zu vereinfachen [RPK12]. Dazu werden die betrachteten Modelle in die *implementation component configuration language (ICCL)* überführt, in denen dann die Variabilität explizit im Modell gespeichert wird. Um die Variationen zwischen Modellen zu finden, berechnen Ryssel et al. die Ähnlichkeit zwischen zwei Komponenten. Diese besteht aus zwei Teilen. Für die *lokale Ähnlichkeit* werden die Typen, die Namen, sowie strukturbeschreibende (z. B. die Schnittstelle eine Blockes) und verhaltensbestimmende Parameter in Betracht gezogen. Die *Nachbarschaftsähnlichkeit* ist der zweite Teil des Ansatzes. Für diese überführen Ryssel at al. die zu vergleichenden Modelle in einen Graphen und berechnen für alle Blöcke vom gleichen Typ die lokale Ähnlichkeit. Anschliessend werden die Graphen nach ihrer Ähnlichkeit gruppiert. Die so entstehenden Cluster können in die *ICCL* transformiert und sprachunabhängig gespeichert werden. Obwohl der Ansatz in den Grundzügen unserem Ansatz ähnelt, gibt es wesentliche Unterschiede: Der Ansatz von Ryssel et al. verfolgt generell ein anderes Ziel, was sich auch auf die Modellrepräsentation mittels ICCL auswirkt. Des Weiteren muss nach der Cluster-Bildung manuell in den Prozess der Variabilitätsfindung eingegriffen werden. Schließlich weicht auch die Definition des Ähnlichkeitsmaßes von der in unserem Absatz erheblich ab.

In weiteren Arbeiten beschreiben Ryssel et al. wie Variationspunkte und Abhängigkeiten zwischen diesen Variationspunkten in einer Menge von MATLAB/Simulink Modellen erkannt und in einem Merkmalsmodell (Featuremodell) dargestellt werden können [RPK10]. Darüber hinaus wird ein Ansatz vorgestellt, um Merkmalsmodelle aus einer Beschreibung von Produkten in einer Produkt-Merkmals-Tabelle zu erstellen [RPK11].

Zuletzt ist noch ein interessanter Ansatz von Rubin et al. zu nennen, der auch das Zusammenführen von Modellen zu SPLs verfolgt [RC10]. Die grundlegende Idee ist dabei, Klassen und Stateflow-Diagramme zu vergleichen und manuell zu vereinfachen und zusammenzuführen. Im Gegensatz zu unserem Ansatz fokussieren sich Rubin et al. also auf

[6] http://pi.informatik.uni-siegen.de/Projekte/sidiff/
[7] http://www.ensoftcorp.com/simdiff/

UML-Modelle und setzen keine automatische Erkennung um.

8 Zusammenfassung und Ausblick

In dieser Arbeit haben wir einen Ansatz vorgestellt, der es ermöglicht, eine Menge einfacher block-basierter Funktionsmodelle einzulesen, auf Gemeinsamkeiten und Unterschiede hin zu analysieren und diese Informationen in ein Familienmodell zu überführen. In unseren zukünftigen Arbeiten planen wir eine Evaluation unseres Ansatzes auf Modellvarianten von industrieller Größe anhand einer Fallstudie im Bereich Fahrerassistenzsysteme, sowie eine Erweiterung des Ansatzes zur Erzeugung von Familienmodellen für Modelle mit hierarchischen Strukturen und variierenden Schnittstellen.

Literatur

[CE00] Krzysztof Czarnecki und Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[DHJ+08] F. Deissenboeck, B. Hummel, E. Jurgens, B. Schatz, S. Wagner, J.F. Girard und S. Teuchert. Clone detection in automotive model-based development. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, Seiten 603–612. IEEE, 2008.

[PNN+09] Nam H. Pham, Hoan Anh Nguyen, Tung Thanh Nguyen, Jafar M. Al-Kofahi und Tien N. Nguyen. Complete and accurate clone detection in graph-based models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, Seiten 276–286, Washington, DC, USA, 2009. IEEE Computer Society.

[RC10] Julia Rubin und Marsha Chechik. From products to product lines using model matching and refactoring. *Proc. of SPLC Wrksp.(MAPLE 2010)*, 2010.

[RPK10] Uwe Ryssel, Joern Ploennigs und Klaus Kabitzsch. Automatic variation-point identification in function-block-based models. In *Proceedings of the ninth international conference on Generative programming and component engineering*, GPCE '10, Seiten 23–32, New York, NY, USA, 2010. ACM.

[RPK11] Uwe Ryssel, Joern Ploennigs und Klaus Kabitzsch. Extraction of feature models from formal contexts. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, Seiten 4:1–4:8, New York, NY, USA, 2011. ACM.

[RPK12] Uwe Ryssel, Joern Ploennigs und Klaus Kabitzsch. Automatic library migration for the generation of hardware-in-the-loop models. *Science of Computer Programming*, 77(2):83–95, 2012.