JUST SIMPLIFY

Heuristische Duplikaterkennung auf Modellen

Dr. Heiko Dörr, Dr. Ingo Stürmer
Model Engineering Solutions
Friedrichstrasse 55
10117 Berlin
doerr@model-engineers.com
stuermer@model-engineers.com

Abstract: Die modellbasierte Entwicklung hat sich für die Entwicklung eingebetteter Software industriell etabliert. Die Qualität der Modelle, aus denen die eingebettete Software generiert wird, ist entscheidend für die Sicherheit des entwickelten Systems. JUST SIMPLIFY ist ein technisches Verfahren, das die Wartbarkeit und damit die Qualität von Software-Modellen verbessert. Das Verfahren erkennt Duplikate in Modelle, um diese dann in wiederverwendbaren Modellbibliotheken auszulagern. Diese Restrukturierung der Modelle senkt die Komplexität und steigert die Wartbarkeit und Qualität der Modelle. Referenzmodelle dienen gleichzeitig zur Validierung der Verfahren.

1 Industrielle Herausforderungen

Bei der Entwicklung eingebetteter Software hat sich aus Effizienzgründen und vor allem zur Erhöhung der Softwarequalität (Fehlerfreiheit, Sicherheit, Robustheit, etc.) die modellbasierte Entwicklung industriell etabliert. Entscheidend ist Softwarequalität insbesondere dann, wenn Software in sicherheitsrelevanten Systemen zum Einsatz kommt. Als markanteste Beispiele seien hier die Entwicklung von Steuergerätesoftware im Fahrzeug und Steuerungsfunktionen im Luftfahrzeugbereich genannt. Ein für die modellbasierte Entwicklung eingebetteter Software gebräuchliches Modellierungs- und Simulationswerkzeug, das im industriellen Umfeld weite Verbreitung[1] gefunden hat, ist *MATLAB/Simulink/Stateflow* [MAT2013] . Während MATLAB als Basisumgebung fungiert, stellen die Erweiterungen Simulink und Stateflow graphische Editoren und Simulatoren für Blockschaltbilder bzw. Statecharts[2] zur Verfügung. Aus diesen graphischen Modellen kann die Software für das eingebettete System mittels Codegeneratoren automatisch generiert werden. Die Qualität der Modelle, die zur Generierung der eingebetteten Software verwendet werden, ist somit entscheidend für die Korrektheit und Sicherheit des zu entwickelnden Systems.

[1] Weltweit ist von ca. 100.000 Simulink/Stateflow-Anwendern auszugehen.
[2] Statecharts dienen der Modellierung von Zustandsautomaten.

Modelle, die für die Entwicklung von realen Anwendungen im Fahrzeug eingesetzt werden, haben häufig mehr als 20.000 Simulink-Blöcke und bestehen aus über 300.000 einzelnen Beschreibungselementen. Diese Größe und die damit verbundene Komplexität der Modelle führen dazu, dass diese Modelle nur noch schwer zu testen, zu warten und zu erweitern sind. Dadurch steigt die Gefahr potenzieller Fehler in der generierten Software. Gängige Entwicklungsstandards[3] für sicherheitsrelevante Software verlangen daher, dass die zu entwickelnde Software bzw. die dafür verwendeten Modelle bestimmte Qualitätskriterien erfüllen, wie z.B. Lesbarkeit, Wartbarkeit, Testbarkeit und niedrige Komplexität auf Modulebene. Für Hersteller von Software für das Automobil besteht ein zwingender Bedarf an neuartigen technischen Lösungen, um die Wartbarkeit von Software-Modellen qualitativ hochwertig und zeiteffizient zu verbessern.

Der aus den Software-Modellen generierte Programmcode muss möglichst effizient sein, d.h. die Funktionalität muss bei relativ geringen Ausführungszeiten und mit geringem Speicherverbrauch erbracht werden. Diese Anforderungen lassen sich bei großen Software-Modellen aber nur dann gleichzeitig realisieren, wenn [MAT2013] die Modellarchitektur so gestaltet ist, dass diese modular und wartbar aufgebaut ist, und [ASH 2011] gleichartige Funktionalität als Bibliotheken oder wiederverwendbare Funktionen zusammengefasst werden. Das Auffinden von gleichartiger Funktionalität in bestehenden Modellen und die Restrukturierung der Modellarchitektur unter Verwendung von Modellbibliotheken und Funktionen ist eine große Herausforderung, weil bisher keine Werkzeugunterstützung vorhanden ist. In der industriellen Praxis erfolgt die Restrukturierung der Modellarchitektur daher bislang manuell, was sehr zeitaufwändig und hochgradig fehleranfällig ist.

2 Problembeschreibung

Der Entwickler eines eingebetteten Systems zerlegt die zu modellierende Funktionalität mit dem Modellierungswerkzeug *MATLAB/Simulink/Stateflow* in so genannte Subsysteme, sodass ein Gesamtsystem als modulare Hierarchie von Subsystemen dargestellt wird. Ein Subsystem ist somit eine hierarchische Gliederungseinheit in einem Simulink-Modell. Die Strukturierung von Modellen ist eine große Herausforderung in der modell-basierten Entwicklung. Bei der Verwendung von *MATLAB/Simulink/ Stateflow* zur Entwicklung von eingebetteter Software sind Bibliotheken das Mittel zur Strukturierung eines Modells. Eine Bibliothek besteht dabei aus verschiedenen, gegebenenfalls parametrierten Subsystemen, den sogenannten Bibliotheksblöcken. Ein solcher Block kann in einem Simulink-Modell über eine Referenz instanziiert und die vorgegebenen Parameter für die jeweilige Instanz spezialisiert werden. Während der Simulation oder auch Code-Generierung löst Simulink die Referenzen auf und nutzt die in der Bibliothek vorliegende Definition des Subsystems. Diese Wiederverwendung von in Bibliotheken ausgelagerten Subsystemen steigert die Wartbarkeit und Testbarkeit der Modelle signifikant, da Änderungen und Tests im Wesentlichen auf die Bibliotheken

[3] Beispiele: ISO 26262 im Automobilbereich, EN50128 im Bereich der Bahntechnik oder DO-178B im Luftfahrzeugbereich

beschränkt sind. Zudem wird der generierte Code deutlich kleiner, da für Bibliotheken und referenzierte Subsysteme nur einmal Code generiert wird.

In *MATLAB/Simulink/Stateflow*-Modellen werden Bibliotheken in zwei Szenarien eingesetzt. Erstens werden wohlbekannte und domänenspezifische Funktionen in Bibliotheken abgelegt und für die Erstellung verschiedener Modelle genutzt. *MATLAB/Simulink/Stateflow* etwa bietet ein breites Spektrum solcher Bibliotheken an. Diese Einsatzweise ist in der klassischen Programmierung wohl bekannt. Das zweite Szenario der Bibliotheksverwendung trägt ebenfalls zur Strukturierung von Modellen bei. Hier werden jedoch Bibliotheken spezifisch für ein Modell erstellt. Gleichartige aber anwendungsspezifische Subsysteme, die an verschiedenen Stellen eines Modells verwendet werden, werden durch einen Bibliotheksblock repräsentiert. In dieser Variante dienen Bibliotheken der Abstraktion, die in anderen Programmiersprachen und -paradigmen eher durch Funktions- oder Klassendefinitionen realisiert werden kann.

Häufig wird dieser Bibliotheksmechanismus zur Strukturierung eines Modells jedoch nicht verwendet. Im Gegenteil werden Subsysteme durch Kopieren und wiederholtes Einfügen mehrfach im Modell dupliziert. Es entstehen so genannte Modell-Duplikate, also Subsysteme als verschiedene Kopien eines gemeinsamen Ausgangs-Subsystems. Diese Erzeugung von Duplikaten ist in der traditionellen Softwareentwicklung als „Cloning“ bekannt. “Clones are segments of code that are similar according to some definition of similarity.”[4]

Abbildung 1 zeigt die Darstellung des strukturellen Aufbaus eines Modells in Form einer Baumdarstellung. Graue Elemente des Strukturbaumes repräsentieren Subsysteme und Bibliotheken, die Funktionalität kapseln, weiße Elemente repräsentieren Basisblöcke, die nicht weiter unterteilt werden. Die beiden Subsysteme „Original“ und „Kopie“ sind Duplikate, d.h. „Kopie“ wurde durch Kopieren des Subsystems „Original“ in das Modell eingefügt.

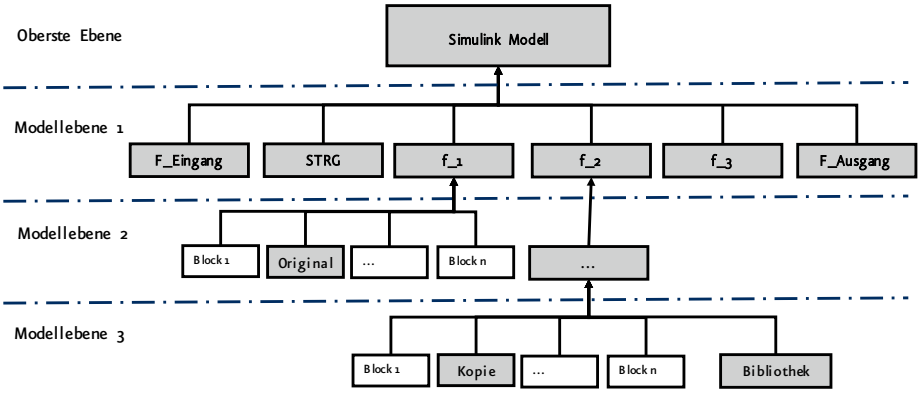


Abbildung 1: Darstellung der hierarchischen Modellstruktur

[4] Ira Baxter, 2002

Es ist gerade bei großen Modellen nicht ausgeschlossen, dass das „Original“-Subsystem noch in andere Teile des Modells hineinkopiert wurde und so eine ganze Gruppe von Duplikaten im Modell existiert, die durch einen einzigen Bibliotheksblock ausreichend und angemessen beschrieben sein könnten. Wenn die Duplikate einer Gruppe identisch sind, bezeichnen wir sie im Folgenden auch als *vollständige Duplikate*.

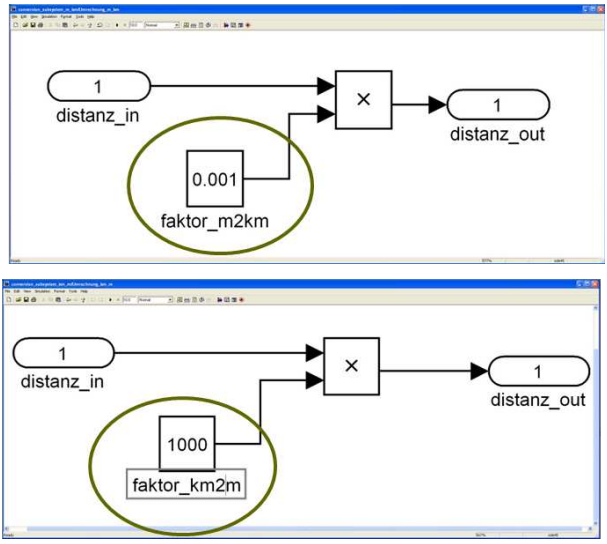


Abbildung 2: Annähernde Duplikate zur Umrechnung von Längenmaßen

In der Regel werden originale Subsysteme nicht nur identisch wiederverwendet, sondern an den neuen Einsatzort angepasst. Der Entwickler nutzt in diesem Fall die im kopierten Subsystem vorliegende Lösungsidee und wandelt diese für seine Zwecke ab. Ein einfaches Beispiel hierfür ist etwa die Umrechnung von Längenmaßen: Die Modellstruktur hierfür ist immer gleich, der wesentliche Unterschied ist der Umrechnungsfaktor. Dieser wird je nach Einsatzort unterschiedlich definiert werden. Dieses Subsystem zur Umrechnung hat großes Potential für die Auslagerung in einen Bibliotheksblock. Häufig haben die Entwickler diesen Schritt nicht durchgeführt und es finden sich Duplikate im Modell mit den oben beschriebenen Nachteilen. Duplikate, die im Zuge des Kopierens verändert wurden, bezeichnen wir im Folgenden als *annähernde Duplikate*.

Je nach Umfang und Art der Anpassung, die im Zuge eines Kopiervorgangs vorgenommen wird, können Duplikate einer Gruppe einer von 5 Ähnlichkeitsgraden zugeordnet werden (nach [GK+2010]):

- Grad 0: Duplikate einer Gruppe sind vollständig identisch
- Grad 1: Duplikate einer Gruppe unterscheiden sich nur bezüglich des Layouts und der Kommentare

- Grad 2: Duplikate einer Gruppe unterscheiden sich auch bezüglich der frei wählbaren Namen
- Grad 3: Duplikate einer Gruppe unterscheiden sich auch bezüglich der enthaltenen Blöcke
- Grad 4: Duplikate einer Gruppe realisieren gleiche Funktionalität

Duplikate des Ähnlichkeitsgrads 0 lassen sich ohne großen Aufwand in einen Bibliotheksblock auslagern. Der gemeinsame Ursprung von Duplikaten des Ähnlichkeitsgrads 3 lässt sich hingegen gar nicht so einfach rekonstruieren, geschweige denn eine Auslagerung in eine Bibliothek vornehmen.

Wie in der traditionellen Softwareentwicklung haben Duplikate auch in der modellbasierten Entwicklung negative Auswirkungen auf die Qualität des Produktes. Die Qualität des Modells und auch die Qualität des generierten Codes verschlechtern sich, da Größe und Gesamtkomplexität des Modells und des generierten Codes deutlich zunehmen. Gleichzeitig verschlechtert sich durch die wiederholte Verwendung identischer oder ähnlicher Modellteile die Test- und Wartbarkeit der Modelle. Modell-Duplikate führen zu einem erhöhten Testaufwand, da die Duplikate separat getestet werden müssen. Auch die Wartbarkeit verschlechtert sich, da im Falle einer notwendigen Änderung alle Kopien, die von der Änderung betroffen sind, geändert werden müssen.

Um diese potentiellen Fehlerquellen zu vermeiden, fordern die Normen für sicherheitsrelevante Software neben Test- und Wartbarkeit eine gute Modularisierung. Für schlecht strukturierte Modelle müssen daher in einem manuellen Verfahren z.B. im Rahmen eines dedizierten Strukturierungs-Review Modell-Duplikate identifiziert und durch Instanzen von Bibliotheksblöcken ersetzt werden. Realistische Modelle enthalten bis zu 300.000 Blöcke. Daher stößt die manuelle Suche nach Duplikaten schnell an Aufwandsgrenzen und ist fehleranfällig. Für die industrielle Praxis ist eine Werkzeugunterstützung für die automatisierte Identifikation von Modell-Duplikaten essentiell.

3 Lösungsweg

3.1 Zielsetzung

JUST SIMPLIFY ist ein Verfahren, das erstens Duplikate in Modellen identifiziert und zweitens diese Duplikate in wiederverwendbare Modellbibliotheken auslagert. Dazu werden im Einzelnen folgende Schritte durchgeführt:

1. Auffindung von Modell-Duplikaten für Simulink- und Stateflow-Modelle auf Basis von Metriken zur Messung der Modellkomplexität.
2. Analyse und Bewertung von Modell-Duplikaten unter Einsatz von Ähnlichkeitsmaßen zur Auswahl geeigneter Restrukturierungsverfahren.

3. Automatische Generierung von wiederverwendbaren Modellbibliotheken aus den Modell-Duplikaten und Restrukturierung der Modellarchitektur durch Ersetzung der Duplikate durch die generierten Modellbibliotheken oder Funktionen.

In zahlreichen Kundenprojekten haben wir einen Zusammenhang zwischen der Modellkomplexitätsanalyse und dem Duplizieren von Modellteilen beobachtet. Die von uns eingesetzte, für Simulink / Stateflow angepasste Halstead-Metrik berechnet die Komplexität eines Simulink-Subsystems als Funktion der Breite von Ein- und Ausgabeschnittstellen, sowie der Komplexität der arithmetischen Basis-Blöcke

In die Komplexität eines Stateflow-Diagramms gehen die Anzahl der Transitionen und Zustände sowie ebenfalls die Komplexität der arithmetischen Operationen ein, die in Zuständen und an Transitionen berechnet werden. Simulink und Stateflow unterstützen originär hierarchische Modelle. Daher werden zudem lokale und globale Komplexität unterschieden: die lokale Komplexität erfasst die Komplexität des Subsystems oder Zustandsdiagramms auf der betrachteten Ebene. Globale Komplexität erfasst die Komplexität aller Subsysteme bzw. Zustandsdiagramme, die hierarchisch unter einem betrachten Subsystem oder Zustandsdiagramm angeordnet sind. In der Berechnung der globalen Komplexität werden die lokalen Komplexitätswerte aller unter dem betrachteten Subsystem bzw. Zustandsdiagramm befindlichen Subsysteme bzw. Zustandsdiagramme addiert. Nicht weiter verfeinerte Subsysteme besitzen demnach gleiche lokale wie globale Komplexität.

Subsysteme, die durch Kopieren im Modell dupliziert wurden, besitzen gleiche lokale und globale Komplexitätswerte. JUST SIMPLIFY beruht auf der Erkenntnis, dass gleiche bzw. ähnliche Komplexitätswerte von Subsystemen signifikante Anzeichen für vollständige bzw. annähernde Duplikate sind.

3.2 Übersicht über das Verfahren

Das Vereinfachungsverfahren macht sich die beschriebene Beobachtung zu Nutze, um Modell-Duplikate automatisch aufzufinden. Zunächst wird die Halstead-Metrik für die Subsystemen und Blöcke des zu überarbeitenden Modells berechnet. Die berechneten Modell-Metriken dienen als Eingabe für die Erkennung der Modell-Duplikate. Das Ersetzungsverfahren restrukturiert in einem weiteren Schritt für die gefundenen Gruppen von Duplikaten die Modellarchitektur semiautomatisch. Das Ersetzungsverfahren erfolgt in zwei Teilschritten: erstens die Erstellung von Bibliotheksblöcken als Vereinheitlichung von Gruppen von Duplikaten, zweitens den Austausch von Duplikaten durch Instanzen dieser Bibliotheksblöcke.

- Schritt 1: Analyse zur Identifikation von Gruppen von Duplikaten unter Einsatz von Modell-Metriken
- Schritt 2.1: Generierung von Bibliotheksblöcken für Gruppen von Duplikaten
- Schritt 2.2: Ersetzung der Duplikate durch Instanzen der jeweiligen Bibliotheksblöcke

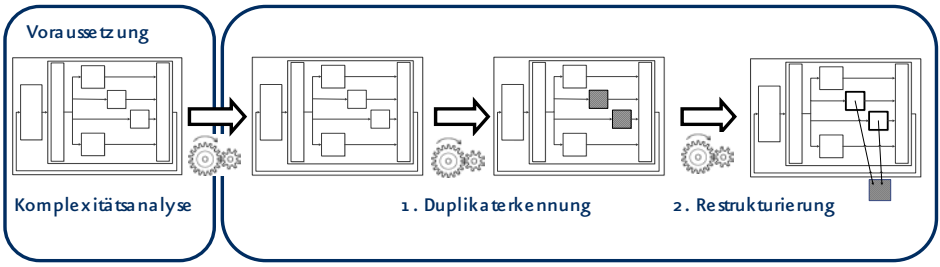


Abbildung 3: Einbettung der Analyse- und Ersetzungsverfahren

3.3 Analysephase

Im Gegensatz zu bekannten Analyseverfahren für das Auffinden von Modell-Duplikaten zielt unsere Heuristik nicht darauf ab, beliebige funktional identische Modellteile aufzudecken, sondern vor allem Modell-Duplikate, die durch die zuvor beschriebene Praxis des Kopierens von Subsystemen entstanden sind. JUST SIMPLIFY berücksichtigt daher bei der Analyse die Informationen über den strukturellen Aufbau des Modells, um gezielt nach dieser Art von Modell-Duplikaten zu suchen.

3.4 Ersetzungsphase

Um die Überarbeitung eines Modells nach dem Aufdecken von Modell-Duplikaten zu unterstützen, wird eine Restrukturierung von Modellen vorgenommen (Schritt 2 Restrukturierung). Die Erzeugung von Bibliotheksblöcken für vollständige identische Duplikate ist dabei eine einfache Aufgabe, da ein Bibliotheksblock direkt aus einem Duplikat erzeugt werden kann. Im zweiten Teilschritt werden die Duplikate gegen entsprechende Instanzen des erzeugten Bibliothekblocks ausgetauscht.

Die Behandlung annähernder Duplikate ist deutlich komplexer. Im ersten Ersetzungsteilschritt *Generierung eines Bibliotheksblocks* werden für eine Gruppe von annähernden Duplikaten die Unterschiede zwischen den einzelnen Duplikaten identifiziert. Diese lassen sich auf typische Anpassungsoperationen zurückführen, die während des Kopiervorgangs vorgenommen wurden. Die Umbenennung oder Ersetzung eines Parameters, wie in Abbildung 2 gezeigt, ist ein häufig auftretendes Beispiel. Diese Anpassungsoperationen liefern die Grundlage für eine Klassifikation der Gruppen von Duplikaten. Für jede Klasse von Duplikaten sind eine oder mehrere Vereinheitlichungsoperationen definiert, die bei der Bildung eines Bibliotheksblocks eingesetzt werden können, um die verschiedenen annähernd gleichen Duplikate auf einen gemeinsamen Bibliotheksblock zurückzuführen. Auf Grund der Entstehungsgeschichte der Duplikate korrelieren die Vereinheitlichungsoperationen stark mit den ursprünglich vorgenommenen Anpassungsoperationen.

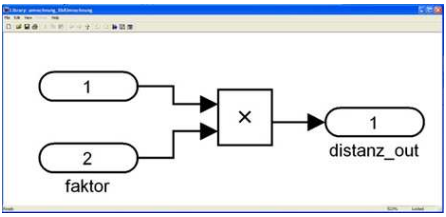


Abbildung 4: Bibliotheksblock, durch Vereinheitlichung aus Duplikaten entstanden

Der zweite Teilschritt des Ersetzungsverfahrens ersetzt die einzelnen Modell-Duplikate durch Instanzen von Bibliotheksblöcken. Für jede Klasse von Duplikaten wird beschrieben, wie die Ersetzung im Einzelnen erfolgt. Reicht es etwa aus, bei der Instanziierung nur einen Parameter zu setzen oder müssen etwa umfangreichere Anpassungen vorgenommen werden, um die Funktionalität des Modells zu erhalten. Die Anpassungen bei der Ersetzung sind eng verknüpft mit den Vereinheitlichungs-operationen, die bei der Bildung des Bibliotheksblocks eingesetzt wurden. Das Ersetzungsverfahren definiert daher für jede Klasse insbesondere von annähernden Duplikaten eine aufeinander abgestimmte Vorgehensweise von Erzeugung eines Bibliotheksblocks und Ersetzen eines Duplikats.

Wie in anderen automatischen Ersetzungsverfahren auch, wird die Ersetzung von Duplikaten in der Regel semi-automatisch erfolgen. Nur bei expliziter Freigabe durch den Modellierer wird eine vollautomatische Ersetzung vorgenommen werden. Durch die Nutzerinteraktion behält der Modellierer die Hoheit über das Modell und die Verantwortung für dessen Korrektheit. Bei der Ersetzung von annähernden Duplikaten wird der Modellierer darüber hinaus noch eine geeignete Parametrierung des instanziierten Bibliotheksblocks vornehmen.

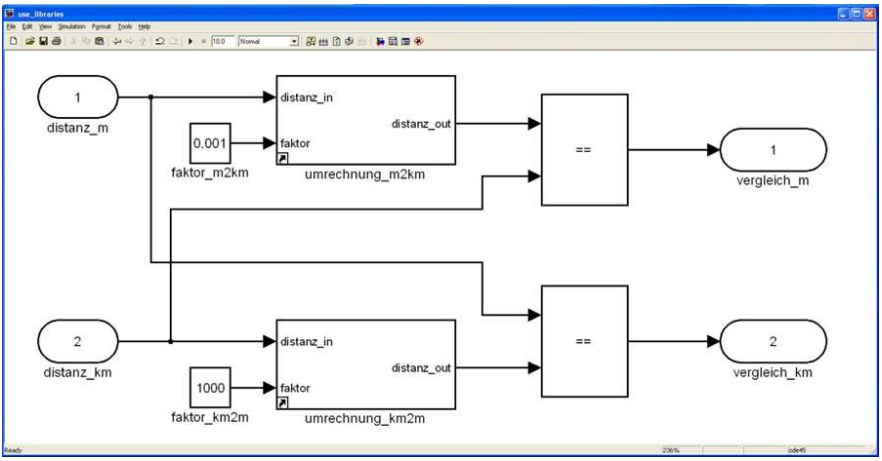


Abbildung 5: Eingesetzte Bibliotheksblöcke

3.5 Zusammenfassung

JUST SIMPLIFY reduziert den manuellen Aufwand für das Auffinden und Ersetzen von Modell-Duplikaten deutlich. Das Verfahren konzentriert sich dabei auf Modell-Duplikate, die in der Praxis häufig auftreten. JUST SIMPLIFY automatisiert weiterhin die Restrukturierung von Modellen, in der gefundene Duplikate durch Instanzen von Bibliotheksblöcken ersetzt werden.

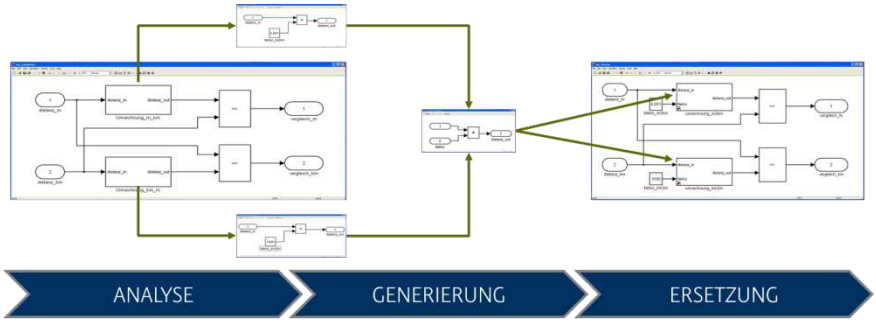


Abbildung 6: JUST SIMPLIFY

JUST SIMPLIFY (1) erfordert keine rechenintensive Abbildung des Modells auf einen Graphen oder eine Zeichenkette, (2) berücksichtigt die Strukturmerkmale des Modells, (3) sucht gezielt nach den zuvor beschriebenen Typen von Duplikaten, für die eine Überarbeitung des Modells besonders sinnvoll sind, und (4) wird mit geeigneten Techniken für die Überarbeitung eines Modells kombiniert.

4 Related Work

Die in der Literatur bekannten Ansätze zur Duplikaterkennung in Modellen lassen sich in zwei Gruppen einteilen: *graph-basierte* und *text-basierte* Verfahren.

4.1 Graph-basierte Ansätze zur Duplikaterkennung

Die graph-basierten Ansätze [ASH 2011], [DH+2010], [GQ2012], [Hol2011], [Pet 2012], [PN+2009] bilden ein Modell auf einen Graphen ab, wobei nur eine Teilmenge der im Modell vorhandenen Informationen für den Aufbau der Graphstruktur verwendet wird. In diesem Graphen werden durch unterschiedliche Heuristiken isomorphe Teilgraphen identifiziert, die Modell-Duplikaten entsprechen. Die Abbildung auf den Graphen berücksichtigt nicht die hierarchische Struktur des Modells. Daher übersehen diese Ansätze häufig Duplikate, die durch das Kopieren von Subsystemen entstanden sind.

Ryssel et al. [RPK2012] transformieren das Modell ebenfalls in einen gerichteten Graphen, der Subsysteme, Blöcke und Ports als Knoten repräsentiert. Knoten werden durch Kanten verbunden, die aus Signalen und der Zuordnung von Ports zu Subsystemen

abgeleitet sind. Die Ähnlichkeit von Knoten wird paarweise für alle Knoten der Graphen ermittelt. In der resultierenden Ähnlichkeitsmatrix werden Cluster ähnlicher Knoten ermittelt und diese dann als Grundlage für die Generierung von variationsbehafteten Bibliotheken genutzt.

Die verallgemeinerte Betrachtung der Duplikaterkennung auf Basis einer Graph-Repräsentation des zugrundeliegenden Modells erlaubt den Rückgriff auf einen reichen Fundus an Graphalgorithmen. In der Regel besitzen diese Verfahren eine hohe algorithmische Komplexität, so dass die Anwendung auf großen Modellen schwer skaliert. Die Berechnung der Ähnlichkeitsmatrix bei Ryssel hat quadratischen Aufwand.

Die in unserem Verfahren zu Grunde liegende heuristische Berechnung der Modellkomplexität benötigt nur eine Baumtraversierung und hat damit nur einen logarithmischen Aufwand.

4.2 Text-basierte Ansätze zur Duplikaterkennung

Das textbasierte Verfahren [AG+2012] für die Erkennung von Modell-Duplikaten basiert auf einem Ansatz für textuelle Programmiersprachen. Die Datei eines zu untersuchenden Modells (Textformat) wird durch einen Parser eingelesen und die enthaltene Information so aufbereitet, dass Duplikate durch Zeichenkettenvergleiche identifiziert werden können. Dieses Verfahren erlaubt es, Duplikate unterschiedlicher Granularität zu erkennen, so dass eine gezielte Suche nach duplizierten Subsystemen möglich ist. Die Anwendbarkeit dieses Verfahrens für industrielle Modelle ist jedoch bislang in der Literatur nicht nachgewiesen.

4.3 Übertragbarkeit auf andere Modellierungssprachen

Die Vorgehensweise von JUST SIMPLIFY lässt sich grundsätzlich auf andere Modellierungssprachen übertragen. Wesentliche Aufgabe ist hierbei zunächst die Definition einer charakteristischen Komplexitätsmetrik, die die Erkennung von Duplikaten leistet. Die Berechnung der Komplexität eines Modells ist der Ausgangspunkt der heuristischen Duplikaterkennung. Für spezifische Modellierungs-sprachen ist ein Komplexitätsmaß jeweils so zu definieren, dass die ähnliche Komplexität auf Duplikate hinweist. Dabei sind die spezifischen Sprachkonzepte hinsichtlich ihrer Signifikanz für die Duplikate zu bewerten. Die Anzahl von Attributen einer UML-Klasse etwa ist ein Komplexitätsmaß, dass zwar ein notwendiges, aber offensichtlich nicht hinreichendes Kriterium für Ähnlichkeit ist.

Weiterhin ist die Ersetzung von Duplikaten durch Bibliotheken offensichtlich abhängig von der Existenz und Ausrichtung eines Bibliothekskonzepts. Im einfachsten Fall muss es dabei ausreichen, die Duplikate durch eine geeignete Reorganisation eines Modells zu reduzieren und so die Qualität eines Modells zu verbessern.

5 Ergebnisse und Ausblick

Die Analyse von Modellen wurde an einer Vielzahl von Modellen durchgeführt. Im Folgenden wird exemplarisch gezeigt, wie die heuristische Ermittlung von Duplikaten auf Basis der Modellkomplexität tragfähige Ergebnisse liefert.

Name	Level	Info	Local Complexity	Global Complexity	Cyclomatic Complexity
Subsystem_1	5	-	**51**	51	4
Subsystem_2	5	-	**51**	51	4

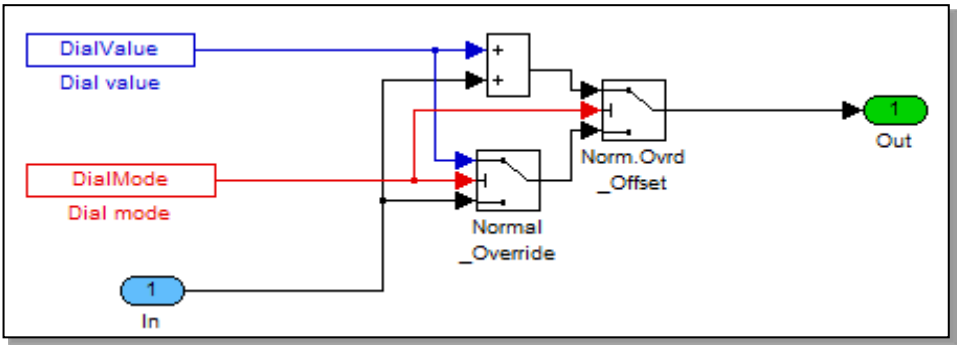
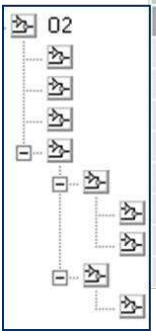


Abbildung 7: Ergebnis der Analyse für vollkommene Duplikate

Die Analyse des komplexen Gesamtmodells identifizierte zwei Subsysteme mit gleicher Komplexität. Da das Subsystem nicht weiter verfeinert ist, sind die Werte für lokale und globale Komplexität für jedes Subsystem ebenfalls gleich. Die Sichtung der Subsysteme bestätigte die heuristisch ermittelten Duplikate. Im obigen Fall wurde tatsächlich das gleiche Subsystem unter den Bezeichnungen „Normal_Override“ und „Norm.Ovrd_ Offset“ an zwei Stellen im Gesamtmodell eingesetzt, ohne dass mit Hilfe eines Bibliotheksblocks eine saubere Strukturierung vorgenommen wurde. Diese kann nun nach Erkennung des Duplikats durch ein automatisiertes Ersetzungsverfahren erfolgen.

Abbildung 8 zeigt die Komplexitätswerte bei annähernden Duplikaten. Die Analyse hat zwei Subsysteme mit der gleichen lokalen Komplexität (Local Complexity = 153 bzw. 48) und vergleichbarer globaler Komplexität identifiziert. Die in den Subsystemen enthaltenen Unterstrukturen weisen ebenfalls gleiche lokale Komplexität auf. Lediglich das letzte Subsystem auf Ebene 5 unterscheidet sich wiederum in der globalen Komplexität (Global Complexity = 559 bzw. 592). Unterschiedliche Teilmodelle sind somit hier zu finden. Die weitergehende Analyse der Komplexitätsergebnisse ergibt nun, dass die Quelle des Unterschieds im letzten Subsystem auf Ebene 6 liegt. Dieses Subsystem ist in den beiden betrachteten Instanzen tatsächlich verschieden realisiert, was bereits durch die unterschiedliche lokale Komplexität gezeigt ist. In der Instanz 2 wurde in dem Subsystem auf Ebene 6 ein weiteres Subsystem hinzugefügt, sodass lokale wie globale Komplexität auf Ebene 6 verschieden sind. Bei der Durchsicht und

nachfolgenden Überarbeitung des Modells kann hier ebenfalls ein Bibliotheksblock geeignet erstellt werden, der dieses Subsystem als aktivierbare Variante enthält.



	Instanz 1				Instanz 2			
	Level	Info	Local Complexity	Global Complexity	Level	Info	Local Complexity	Global Complexity
02	4	-	153	817	4	-	153	850
	5	-	51	51	5	-	51	51
	5	-	51	51	5	-	51	51
	5	-	3	3	5	-	3	3
	5	-	48	559	5	-	48	592
	6	-	167	439	6	-	167	439
	7	-	225	225	7	-	225	225
	7	-	47	47	7	-	47	47
	6	-	72	72	6	-	79	105
					7	-	26	26

Abbildung 8: Ergebnis der Analyse für annähernde Duplikate (Unterschiede markiert)

Für größere Modelle mit mehr als 3.000 Subsystemen wurde die Duplikaterkennung ebenfalls durchgeführt. Am Beispiel zweier Modelle seien die Ergebnisse im Detail erläutert. Die Werte sind ohne Berücksichtigung der jeweils aus den Modellen referenzierten Bibliotheken berechnet. Die Berechnung der Komplexität erfolgt mit linearem Aufwand in Abhängigkeit von der Anzahl der Elemente im Modell und skaliert demzufolge mit der Größe der Modelle.

Für Modell A aus der Domäne Exterior wurde eine globale Komplexität von nahezu 75.000 und eine maximale lokale Komplexität von fast 3.000 berechnet. Die maximale Verschachtelungstiefe beträgt 10 Ebenen bei einer Gesamtzahl von rund 3200 Blöcken. JUST SIMPLIFY erkannte in Modell A über 350 Duplikate, was die Erwartungen an die Anzahl von Duplikaten deutlich übertraf. Die nähere Analyse des Modells deckte eine systematische Vorgehensweise in der Modellierung auf. Ursprüngliche Referenzen auf Bibliotheksblöcke wurden während der Modellerstellung aufgelöst und Instanzen der Bibliotheksblöcke im Modell eingesetzt.

Modell B aus der Domäne Interior ist deutlich kleiner mit einer globalen Komplexität von rund 9.500. Das größte Subsystem ist jedoch mit einer maximalen lokalen Komplexität von 2.250 fast gleich groß. Die maximale Verschachtelungstiefe beträgt 9 Ebenen bei einer Gesamtzahl von etwas mehr als 700 Blöcken. JUST SIMPLIFY erkannte in Modell B 55 Duplikate in 5 Gruppen. Die Analyse der Duplikate bestätigte die ursprüngliche Annahme, dass die Duplikate durch Kopieren erzeugt wurden. Die Verifikation der Ergebnisse wird durch die automatische Generierung von Bezeichnern durch Simulink erkennbar: Duplikate in einem Subsystem, die durch Kopieren erzeugt wurden, werden von Simulink automatisch über Postfix-Indizes eindeutig bezeichnet. Die Generierung von Bibliotheksblöcken und die Ersetzung von Duplikaten durch Instanzen von Bibliotheksblöcken werden den Umfang der Modelle deutlich verkleinern und die Wartbarkeit signifikant verbessern.

Unsere Studien zeigten zudem, dass unsere Heuristik verbessert werden kann, um den Anteil erkannter ähnlicher Duplikate zu erhöhen. Dies lässt sich dadurch erreichen, dass für die Duplikaterkennung weitere Ähnlichkeitsmaße berechnet werden, die zusätzlich zu den Komplexitätswerten beispielsweise Block-Parameter, block-verknüpfende Linien und andere Merkmale wie beispielsweise Layout-Informationen berücksichtigt werden. Damit lässt sich ein Abstandsmaß zwischen den gefundenen Duplikaten bestimmen, dass qualitativ und quantitativ deutlich bessere Ergebnisse für die Duplikaterkennung als bisherige Analysen liefern.

Referenzen

[MAT2013] Mathworks 2013. http://www.mathworks.com/products/product_listing/index.html.

[ASH 2011] Bakr A., Bernhard S., Benjamin H.: Semantic clone detection for model-based development of embedded systems. In *MoDELS*, 2011: pp. 258–272.

[AG+2012] M.H.A., J.R.C. et al.: Models are code too: Near-miss clone detection for simulink models. In *IEEE 28th International Conference on Software MaintenanceICSM 2012*.

[DH+2010] Florian D., Benjamin H. et al.: Model clone detection in practice. In *Proceedings of the 4th International Workshop on Software Clones*, IWSC ’10. ACM. New York, NY, USA, 2010; pp. 57-64.

[GQ2012] Francesco G., Jochen Q.: Type 2 clone detection on ASCET models. In *14. Workshop Software-Reengineering*, 2012.

[GK+2010] Nicolas G., Jens K et al.: Issues in clone classification for dataflow languages. In *Proceedings of the 4th International Workshop on Software Clones*, IWSC ’10. ACM. New York, NY, USA, 2010; pp. 83-84.

[Hol2011] Wolfgang H.: Metriken zur Qualitätsmessung von Simulink-Modellen in der modellgetriebenen Softwareentwicklung. Master’s thesis, Universität Ulm, 2011.

[Pet 2012] Hauke P.: Clone detection in MATLAB Simulink models. Master’s thesis, Technical University of Denmark, DTU Informatics, Email: reception@imm.dtu.dk, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark, 2012.

[PN+2009] Nam H. P., Hoan Anh N. et al.: Complete and accurate clone detection in graph-based models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE ’09. IEEE Computer Society. Washington, DC, USA, 2009; pp. 276-286.

[RPK2012] Uwe R., Joern P., Klaus K.: Automatic library migration for the generation of hardware-in-the-loop models. In *Science of Computer Programming 77 (2012)*; pp. 83 – 95.