Heinrich C. Mayr, Martin Pinzger (Hrsg.): INFORMATIK 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 1523

Automatische Duplikateliminierung in Aktivitätsdiagrammen von Fahrzeugfunktionen

Martin Beckmann[1] und Aaron Schlutter[2]

Abstract: Die Spezifikation von Fahrzeugfunktionen ist eine komplexe Aufgabe. Zum Umgang mit dieser Komplexität werden zur Beschreibung der Funktionen grafische Modellierungssprachen wie UML verwendet. Bei der Modellierung können Duplikate entstehen, welche Ausgangspunkt für Fehler und Inkonsistenzen in der weiteren Entwicklung sind. Dieser Beitrag widmet sich der Eliminierung von Duplikaten, welche bei der Spezifikation von Fahrzeugfunktionen mittels UML Aktivitätsdiagrammen auftreten. Es wird dargestellt, wie in UML Aktivitätsdiagrammen identifizierte Duplikate automatisiert eliminiert werden, ohne die ursprüngliche Funktionalität zu verändern. Mehrfach auftretende Elemente werden zusammengefasst und durch das Einfügen von weiteren Elementen und Verbindungen zusammengesetzt.

Keywords: Automatische Duplikateliminierung, UML Aktivitätsdiagramme, Modellbasierte Spezifikation

1 Einführung

Zur Beherrschung der zunehmenden Komplexität bei der Entwicklung von Fahrzeugsoftware wird vermehrt modellbasierte Softwareentwicklung verwendet [Br06]. Verhaltensmodelle wie das UML *Aktivitätsdiagramm* [UM15, S. 371] können dazu genutzt werden, um Funktionsabläufe zu beschreiben und grafisch darzustellen. Die Erstellung eines solchen Modells im Rahmen der Spezifikation kann daher frühzeitig dazu beitragen, Fehler, Widersprüche und Inkonsistenzen aufzuzeigen [AD97]. Eine mögliche Anwendung des Modells im weiteren Entwicklungsprozess ist die automatische Generierung von Bestandteilen einer textuellen Spezifikation. Diese Generierung ist eines unserer weiteren aktuellen Forschungsvorhaben.

Während der Erstellung der Funktionsmodelle können innerhalb des Modells Redundanzen auftreten. Diese entstehen oft unabsichtlich, da bei deren Erstellung viele Personen beteiligt sind und trotz des Einsatzes von grafischen Werkzeugen der Überblick schwierig ist. Die Redundanzen entstehen jedoch auch absichtlich [LEM02]. Beispielsweise werden redundante Elemente eingefügt, um die Modelle besser zu gruppieren, wenn ein einzelnes Element in mehreren Zusammenhängen innerhalb der Fahrzeugfunktion auftritt. Die dabei entstehenden Duplikate können jedoch Schwierigkeiten bei der Wahrung der Konsistenz verursachen, wenn die Diagramme in weiteren Entwicklungsschritten verwendet werden. Duplikate sollten daher entfernt werden.

[1] Technische Universität Berlin, Daimler Center for Automotive IT Innovations, Ernst-Reuter-Platz 7, 10587 Berlin, martin.beckmann@tu-berlin.de
[2] Technische Universität Berlin, Daimler Center for Automotive IT Innovations, Ernst-Reuter-Platz 7, 10587 Berlin, aaron.schlutter@tu-berlin.de

Dieser Beitrag widmet sich der Eliminierung identifizierter Duplikate in UML *Aktivitätsdiagrammen*. Bei der Eliminierung werden mehrfach auftretende Elemente zusammengefasst. Bestehende Verbindungen der zusammengefassten Elemente werden übernommen und durch zusätzliche Elemente und Verbindungen so zusammengesetzt, dass die Funktionalität des ursprünglichen *Aktivitätsdiagramms* erhalten bleibt.

Dieser Beitrag ist wie folgt aufgebaut. Im nächsten Kapitel werden zunächst Grundlagen erläutert und anschließend werden anhand eines einführenden Beispiels Annahmen zum Verständnis dieses Beitrags dargelegt. Eine Übersicht über verwandte Arbeiten ist im dritten Kapitel enthalten. In Kapitel 4 wird das Konzept zur Eliminierung der Duplikate vorgestellt und im fünften Kapitel wird das Konzept auf das einführende Beispiel angewendet. Der Beitrag wird mit einer Zusammenfassung und einem Ausblick in Kapitel 6 abgeschlossen.

2 Grundlagen

Im Rahmen dieser Arbeit werden UML *Aktivitätsdiagramme* zur Beschreibung von Fahrzeugfunktionen betrachtet. *Aktivitätsdiagramme* sind Petri-Netz ähnliche Graphen [UM15, S. 283]. Mithilfe von Tokens (Marken) wird die Ausführungsreihenfolge von Elementen und damit der mögliche Ablauf des *Aktivitätsdiagramms* modelliert. Für die Ausführung eines Knotens muss sich ein Token in diesem befinden. Das Token kann bei Abschluss der Ausführung des Knotens an weitere Knoten weitergegeben werden. Das Verhalten eines Petri-Netzes beziehungsweise *Aktivitätsdiagramms* kann durch einen Erreichbarkeitsgraph dargestellt werden. Ein Erreichbarkeitsgraph ist ein gerichteter Graph, der alle möglichen Reihenfolgen (Pfade) zur Ausführung der Knoten zeigt. Ein Knoten des Erreichbarkeitsgraphs stellt einen möglichen Markierungszustand aller Knoten des Petri-Netzes dar.

In Abbildung 1 ist ein *Aktivitätsdiagramm* der Funktion *Ladeparameter berechnen* dargestellt. Verschiedene Knoten (*ActivityNodes*) sind über Kanten (*ActivityEdges*) miteinander verbunden. Die Funktion hat verschiedene Auslöser, welche als *AcceptEventAction* Elemente dargestellt sind. Neben den Auslösern werden bei der Ausführung der Funktion auch *Actions* durchlaufen. Die Zusammenhänge dieser Elemente wird durch *ControlNodes* gesteuert. Auslöser und *Actions* können mit *JoinNodes* UND-verknüpft oder mit *MergeNodes* ODER-verknüpft werden. Weitere in diesem Beitrag genutzte *ControlNodes* sind *InitialNodes*, *DecisionNodes* und *ForkNodes*. Es werden nur diese Elemente verwendet. Der Grund dafür ist, dass die von einem Industriepartner bereitgestellten Fahrzeugfunktionen ausschließlich mithilfe dieser Elemente modelliert sind.

Die *Actions* mit dem Präfix *Check* stellen Überprüfungen von Bedingungen dar. Schlägt diese Überprüfung fehl, so endet der Fluss in diesem Knoten. Der Fluss eines Tokens von einem Auslöser zu einer Überprüfung bildet daher eine UND-Verknüpfung von Auslöser und Überprüfung ab. Eine Möglichkeit, zur Ausführung der *Action Ladeparameter berechnen* zu gelangen, besteht, wenn der Auslöser *Verbindung "eingesteckt"* aktiviert wird und die darauffolgenden Überprüfungen *V<10 km/h*, *Kl.15 aus* und *JM start nicht aktiv* zu

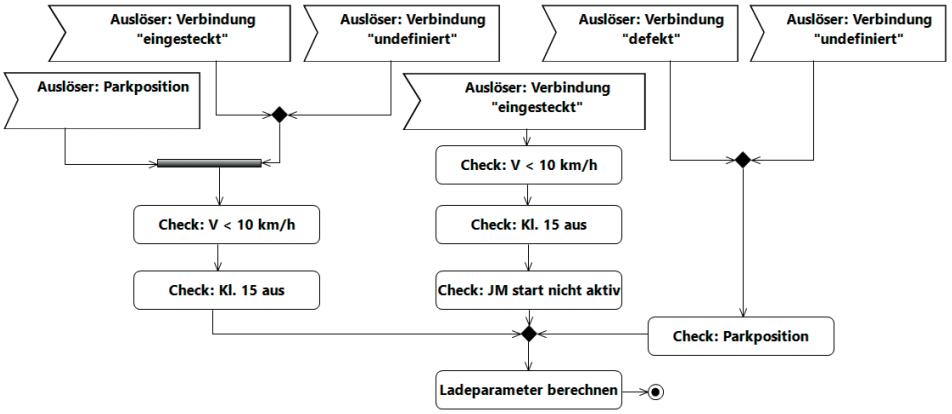


Abb. 1: Funktionsablauf der Fahrzeugfunktion *Ladeparameter berechnen*

wahr ausgewertet werden. Besitzt eine *Action* oder *AcceptEventAction* mehrere ausgehende Kanten, so wird an allen ausgehenden Kanten ein Token angeboten [UM15, S. 401]. Dies entspricht der Anwendung von *ForkNodes*. Treten bei anderen Elementen wie zum Beispiel *DecisionNodes* mehrere ausgehende Kanten auf, so wird nur ein einzelnes Token an einen nachfolgenden Knoten gegeben, welcher das Token akzeptiert [UM15, S. 373, S. 387f].

In der in Abbildung 1 dargestellten Fahrzeugfunktion sind folgende Duplikate enthalten: die Auslöser *Verbindung "eingesteckt"* und *Verbindung "undefiniert"* sowie die Überprüfungen *V<10 km/h* und *Kl.15 aus*. Im Rahmen dieses Beitrags besteht ein Duplikat aus Knoten, welche dieselbe Bedeutung besitzen. Beispielsweise bilden die beiden Überprüfungen mit dem Text *V<10 km/h* ein Duplikat. Es wird davon ausgegangen, dass die Duplikate bereits identifiziert sind. Die Identifikation von Duplikaten bei der Spezifikation ist beispielsweise mit textuellen Vergleichen möglich [Da01]. Es ist jedoch zu beachten, dass innerhalb einer Aktivität auch ähnliche Elemente den gleichen Namen haben können [UM15, S. 372] und trotzdem kein Duplikat darstellen. Dies ist beispielsweise der Fall, wenn von den Elementen mit gleichen Namen verschiedene Aktivitäten aufgerufen werden.

Für die in dieser Arbeit betrachteten Fahrzeugfunktionen und deren *Aktivitätsdiagramme* ist dabei die exakte Reihenfolge zur Ausführung der Knoten nicht relevant. Vielmehr ist es notwendig, dass jeweils alle Vorgänger ausgeführt worden sind und die Gesamtanzahl der Ausführungen nicht verändert wird. Unter Umständen kann durch das Umformungen des in Kapitel 4 vorgestellten Konzeptes die Ausführungsreihenfolge der einzelnen Knoten verändert werden. Dies erfolgt durch die mehrfache, zeitversetzte Ausführung von Auslösern. Der Sachverhalt wird in Kapitel 4 genauer erläutert.

3 Verwandte Arbeiten

In [CST02] werden verschiedene Arten von Redundanzen identifiziert, die in UML Modellen auftreten können. Dabei findet eine Beschränkung auf die Diagrammtypen *Anwendungsfalldiagramm*, *Zustandsdiagramm* und *Sequenzdiagramm* bei der Verwendung während der objektorientierten Analyse statt. Betrachtet werden Redundanzen, die zwischen verschiedenen Modellen auftreten. Außerdem werden Vorteile aufgezeigt, die durch die Beseitigung der Redundanzen entstehen wie zum Beispiel die Wahrung der Konsistenz bei Modifikationen und die erleichterte Nutzung in späteren Phasen der Entwicklung.

Die Umformung von UML Modellen unter Beibehaltung des Verhaltens ist das Thema von [Su01]. Es werden Operationen vorgestellt, die genutzt werden können, um Mängel im Design zu beseitigen. Ebenso wird begründet, wieso das Verhalten der Modelle bei der Durchführung der Operationen erhalten bleibt. Der Beitrag stellt Operationen für *Klassendiagramme* und *Zustandsdiagramme* vor. Auf den Aspekt der Eliminierung von Redundanzen zur Verbesserung der Modelle wird nicht eingegangen.

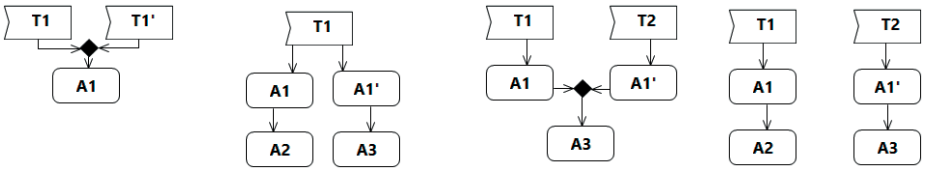
Zwei Operationen, die auf *Aktivitätsdiagrammen* Anwendung finden, sind in [BSF03] beschrieben. Die beiden vorgestellten Operationen behandeln die Reihenfolge der Ausführung von *Actions* und ob diese parallelisiert oder sequenziert auszuführen sind. Zum Thema der Umformung von UML Modellen existieren eine Reihe weiterer Beiträge. Nach Kenntnis der Autoren widmet sich jedoch keine Arbeit der Eliminierung von Duplikaten.

4 Konzept zum Eliminieren von Duplikaten

Das im Folgenden vorgestellte Konzept zum Eliminieren von Duplikaten in *Aktivitätsdiagrammen* besteht aus zwei Teilen. Zum einen werden redundante Elemente zusammengefasst und zum anderen muss die Funktionalität vom *Aktivitätsdiagramm* sichergestellt werden. Dies wird erreicht, indem ein- und ausgehende Kanten der zusammengefassten Knoten weitergeführt werden oder unter Umständen indem neue *ControlNodes* eingefügt und mit dem zusammengefassten Knoten verbunden werden. Nach der Zusammenfassung der Knoten kann zusätzlich nach nicht mehr notwendigen Kanten und *ControlNodes* gesucht werden. Diese Elemente zeichnen sich dadurch aus, dass nach deren Entfernen aus dem *Aktivitätsdiagramm* dessen Funktionalität nicht verändert wird.

In Abbildung 2 sind beispielhaft verschiedene Ausschnitte aus *Aktivitätsdiagrammen* dargestellt. Die Knoten, deren Text mit *T* beginnt, stellen Auslöser dar. Die anderen Knoten sind *Actions*. Ein Hochkomma im Text des Knotens bezeichnet einen redundanten Knoten.

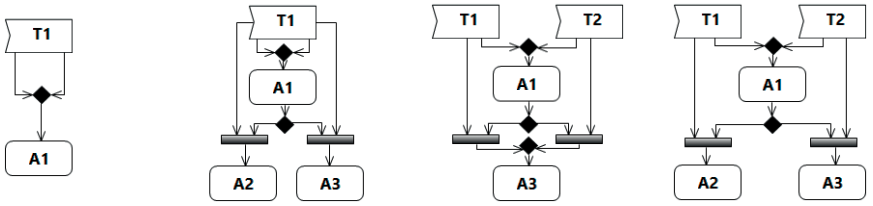
In der ersten Situation in Abbildung 2a sind die Knoten des Duplikats die Auslöser der Funktion. Die Auslöser besitzen keine eingehenden Kanten. In Abbildung 3a ist die duplikatfreie Darstellung zu sehen. Die Auslöser sind in einem Knoten zusammengefasst und alle ausgehenden Kanten haben als neue Quelle den zusammengefassten Knoten und als Ziel ein vor der *Action* befindliches *DecisionNode*. Die Kanten können nicht entfernt oder zusammengeführt werden, da vor der Eliminierung durch beide Auslöser zwei Tokens er-



(a) Situation 1 (b) Situation 2 (c) Situation 3 (d) Situation 4

Abb. 2: Unterscheidung verschiedener Situationen

stellt werden, sodass die *Action A1* zweimal ausgeführt wird. Durch die Umformung wird beim Auslösen von *T1* die *Action A1* ebenfalls zweimal ausgeführt.



(a) Situation 1 (b) Situation 2 (c) Situation 3 (d) Situation 4

Abb. 3: Situationen ohne Duplikate

In den anderen Situationen in Abbildung 2b, 2c und 2d besitzen die redundanten Knoten eingehende Kanten. In der zweiten Situation ist der Vorgänger identisch und die Nachfolger unterschiedlich. In der dritten Situation sind die Vorgänger unterschiedlich und der Nachfolger ist identisch. In der vierten Situation sind sowohl Vorgänger als auch Nachfolger unterschiedlich. In diesen Situationen werden zusätzliche *ControlNodes* und Kanten eingefügt, um die Funktionalität beizubehalten. Diese zusätzlichen *ControlNodes* und Kanten sind im vierten Fall notwendig, damit durch den Auslöser *T1* nach der Beendigung von *Action A1* lediglich *Action A2* aktiviert werden kann. Der *JoinNode* stellt sicher, dass Auslöser *T1* und *Action A1* zuvor ausgeführt worden sind. Der *MergeNode* dient in der dritten und vierten Situation dazu, dass *Action A1* durch ein einzelnes Token von *T1* oder *T2* ausgeführt werden kann. Der *DecisionNode* leitet das Token am Ausgang von *Action A1*, an den entsprechenden *JoinNode* weiter. In der zweiten und dritten Situation werden dabei *ControlNodes* und Kanten eingefügt, welche nicht notwendig sind und wieder entfernt werden können. Um ein einheitliches Konzept für diese Situationen zu ermöglichen, werden diese Situationen jedoch gleich behandelt. Eine Vereinfachung ist im Anschluss möglich.

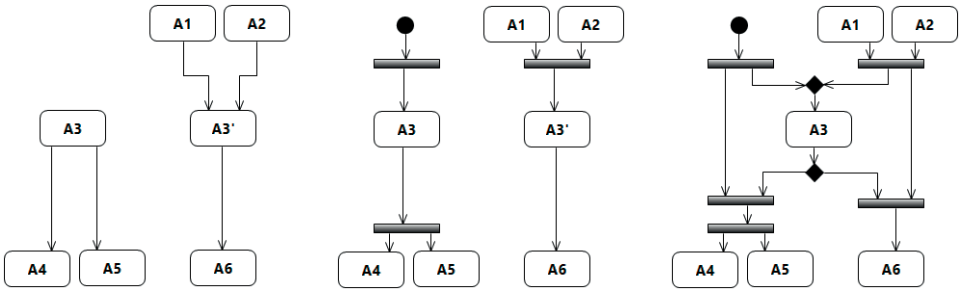
Für die Umformung der redundanten Knoten werden zwei Fälle unterschieden:

1. Keiner der Knoten eines Duplikats hat eingehende Kanten.
2. Mindestens einer der Knoten des Duplikats hat mindestens eine eingehende Kante.

Für den ersten Fall werden die betroffenen Knoten zu einem Knoten zusammengefasst und alle ausgehenden Kanten der ursprünglichen Knoten haben als Quelle den zusammengefassten Knoten. Im zweiten Fall werden ebenfalls die redundanten Knoten in einen Knoten zusammengefasst. Alle eingehenden Kanten werden über einen neuen *MergeNode* gebündelt zu dem zusammengefassten Knoten geführt. Aus dem zusammengefassten Knoten geht eine Kante zu einem *DecisionNode*. Von dort aus gehen Kanten zu *JoinNodes*. Zu den *JoinNodes* führen zudem eingehende Kanten von den Vorgängern der redundanten Knoten. Handelt es sich bei den Vorgängern nicht um *Actions*, werden zusätzlich noch *ForkNodes* hinter diesen eingefügt. Diese dienen stattdessen als Vorgänger, um die Funktionalität von mehreren ausgehenden Kanten bei *Actions* nachzustellen. Die ausgehende Kante des *JoinNodes* führt zu dem dazugehörigen Nachfolger des redundanten Knotens. Sind keine Nachfolger vorhanden, entfallen die entsprechenden Elemente und Verbindungen für die ausgehende Kante.

Besonderheiten

Die in Abbildung 2 dargestellten Situationen decken nicht alle Möglichkeiten der beiden Fälle ab. Es ist denkbar, dass ein Duplikat existiert, welches Knoten besitzt, von denen ein Knoten einen Vorgänger hat, ein anderer Knoten jedoch keinen Vorgänger besitzt. Des Weiteren ist es möglich, dass Knoten von Duplikaten mehr als einen Vorgänger oder Nachfolger besitzen. Ein Beispiel, in denen solche Konstellationen auftreten, ist in Abbildung 4 dargestellt.

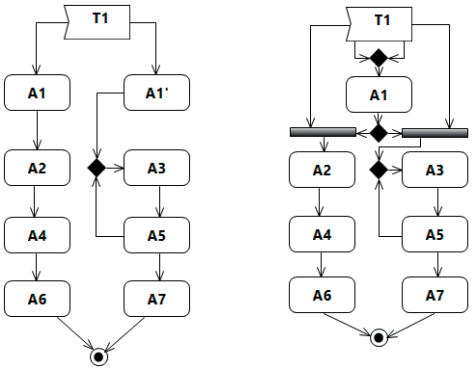


(a) Ausgangssituation (b) Explizite Verbindungen (c) Duplikatfrei

Abb. 4: Sonderfall

In Abbildung 4a besitzt die *Action A3* keinen Vorgänger und zwei Nachfolger. Die *Action A3'* hat zwei Vorgänger und einen Nachfolger. Für den Knoten ohne Vorgänger ist ein *InitialNode* als Vorgänger zu ergänzen. Um mit den Situationen mit mehreren Vorgängern und Nachfolgern umzugehen, müssen zunächst die impliziten Verbindungen ([UM15, S. 401]) explizit dargestellt werden. Bei *Action A3* wird durch die ausgehenden Kanten eine implizite Parallelisierung vorgenommen (vgl. *ForkNode*). Bei *Action A3'* wird durch die mehrfachen Vorgänger eine implizite UND-Verknüpfung erzeugt. In Abbildung 4b sind das Einfügen eines *InitialNodes* und die expliziten Verbindungen dargestellt. Der

zusätzliche *ForkNode* nach dem *InitialNode* ist notwendig, da der Knoten des Duplikats Nachfolger haben kann. In Abbildung 4c ist die Eliminierung nach dem vorgestellten Konzept dargestellt. Als Vorgänger und Nachfolger für die Umformung dienen an den entsprechenden Stellen die eingefügten Elemente der expliziten Verbindungen.



(a) Zyklus (b) Zyklus duplikatfrei

Abb. 5: Zyklus im *Aktivitätsdiagramm*

Eine weitere Besonderheit stellen Zyklen in den *Aktivitätsdiagrammen* dar. Ein solcher Zyklus und dessen Lösung nach dem vorgestellten Konzept ist in Abbildung 5a und in in Abbildung 5b dargestellt. Das Konzept lässt sich korrekt auf eine Reihe von möglichen Zyklen anwenden. Es ist jedoch noch nicht abschließend geklärt, ob dies immer dazu führt, dass die Funktionalität erhalten bleibt.

Vergleich der Erreichbarkeitsgraphen

Allgemein ist für den Erhalt der Funktionalität eines *Aktivitätsdiagramms* bei der Zusammenführung von redundanten Knoten darauf zu achten, dass alle möglichen Pfade des *Aktivitätsdiagramms* erhalten bleiben. Die möglichen Pfade können aus dem Erreichbarkeitsgraphen des *Aktivitätsdiagramms* ermittelt werden. In Abbildung 6 ist der Erreichbarkeitsgraph (*T1*, *T2*, *A1*, *A1'*, *A2*, *A3*) mit der Startmarkierung (1,1,0,0,0,0) für das Aktivitätsdiagramm aus Abbildung 2d und in Abbildung 7 der Erreichbarkeitsgraph (*T1*, *T2*, *A1*, *A2*, *A3*) mit der Startmarkierung (1,1,0,0,0) für das Aktivitätsdiagramm aus Abbildung 3d dargestellt. Für den zweiten Erreichbarkeitsgraphen wird angenommen, dass die neu eingefügten *ControlNodes* direkt ausgeführt werden. Diese Elemente sind daher zur besseren Vergleichbarkeit nicht separat in der Markierung aufgeführt.

Die möglichen Pfade des Erreichbarkeitsgraphen in Abbildung 6 für das *Aktivitätsdiagramm* aus Abbildung 2d lauten:

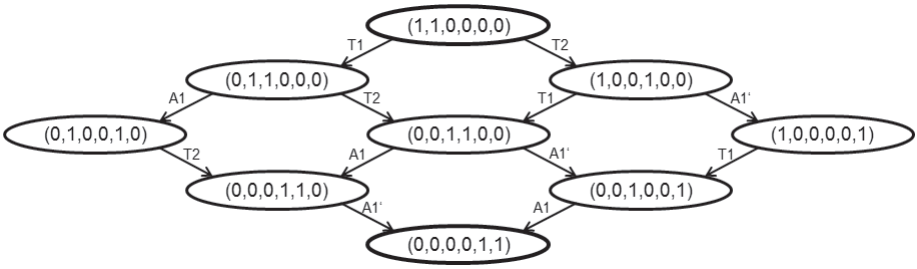


Abb. 6: Erreichbarkeitsgraph (*T1*, *T2*, *A1*, *A1'*, *A2*, *A3*) des *Aktivitätsdiagramms* aus Abbildung 2d

1. *T1*, *A1*, *T2*, *A1'*
2. *T1*, *T2*, *A1*, *A1'*
3. *T1*, *T2*, *A1'*, *A1*
4. *T2*, *T1*, *A1*, *A1'*
5. *T2*, *T1*, *A1'*, *A1*
6. *T2*, *A1'*, *T1*, *A1*

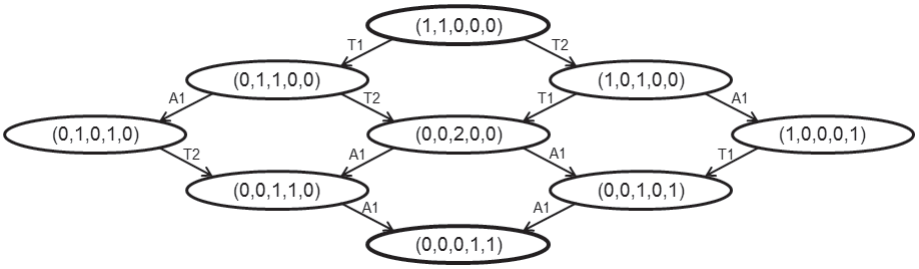


Abb. 7: Erreichbarkeitsgraph (*T1*, *T2*, *A1*, *A2*, *A3*) des *Aktivitätsdiagramms* aus Abbildung 3d

Die möglichen Pfade des Erreichbarkeitsgraphen in Abbildung 7 für das *Aktivitätsdiagramm* aus Abbildung 3d lauten:

1. *T1*, *A1*, *T2*, *A1*
2. *T1*, *T2*, *A1*, *A1*
3. *T2*, *T1*, *A1*, *A1*
4. *T2*, *A1*, *T1*, *A1*

Unter der Voraussetzung *A1=A1'* sind die möglichen Pfade der Erreichbarkeitsgraphen aus Abbildung 6 und 7 äquivalent. Die Pfade Nr. 1 und 6 aus Abbildung 6 sind äquivalent zu den Pfaden Nr. 1 und 4 aus Abbildung 7. Die Pfade Nr. 2 und 3 aus Abbildung 6 sind äquivalent zum Pfad Nr. 2 aus Abbildung 7. Das Gleiche gilt für die Pfade Nr. 4 und 5 aus Abbildung 6 und den Pfad Nr. 3 aus Abbildung 7.

Im Vergleich der beiden Erreichbarkeitsgraphen ist zudem zu erkennen, dass die Anzahl und Verteilung der Token in jedem Knoten (d.h. Zustand im Aktivititätsdiagramms) immer gleich sind, unter der Bedingung *A1=A1'* und dass die Anzahl der Token von *A1* und *A1'* aufsummiert werden.

Der Erreichbarkeitsgraph ist nur bedingt geeignet, den vollständigen Ablauf eines Aktivitätsdiagramms darzustellen. Zum einen gilt ein Erreichbarkeitsgraph immer nur für eine bestimmte Startmarkierung von Token. Für eine andere Startmarkierung als (1,1,0,0,0,0) in Abbildung 6 ergibt sich ein neuer Erreichbarkeitsgraph. Zum anderen gibt es keine vollständige Semantik von Aktivitätsdiagrammen. Im Gegensatz zu Knoten in Petri-Netzen muss eine *Action* nicht zwangsläufig in jedem Schaltschritt vollständig ausgeführt werden in dem Sinne, dass eingehende Token angenommen und nach der Aktivierung innerhalb des Zeitschritts direkt weitergegeben werden. Ein weiterer Unterschied besteht darin, dass ein Auslöser jederzeit ausgeführt werden kann und einen neuen Token produziert.

Durch diese Abweichungen kann es vorkommen, dass unter bestimmten Umständen der zuvor bestehende Ablauf einzelner Fälle nicht exakt identisch nach der Umformung besteht. Im vierten Fall aus Abbildung 3d kann durch Auslöser *T1* die *Action A1* ausgeführt werden. Anschließend liegt an dem *DecisionNode* nach *A1* ein Token. Der Auslöser *T2* kann nun direkt *A3* ausführen, ohne dass direkt zuvor *A1* ein weiteres Mal ausgeführt worden ist. Der Pfad entspricht dann (auszugsweise) *T1*, *A1*, *T2*, *A3*.

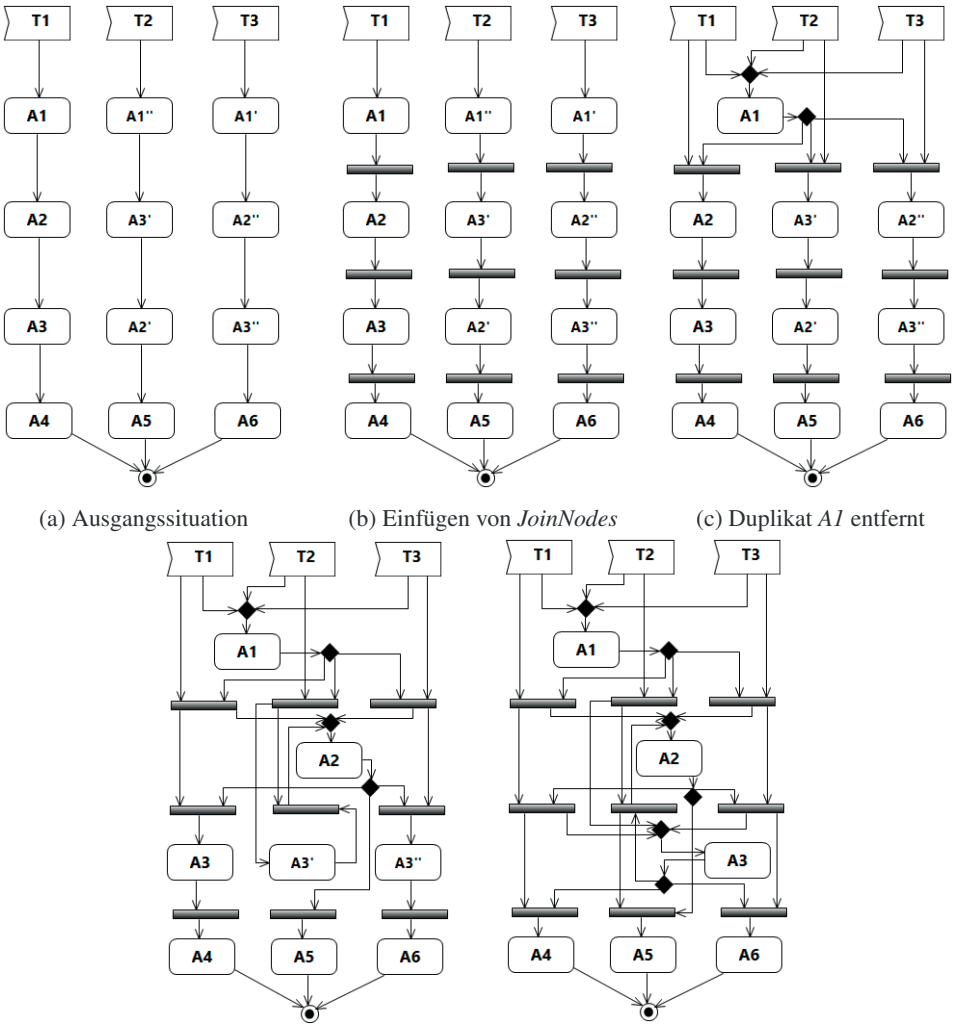
Der entsprechende Erreichbarkeitsgraph (*T1*, *T2*, *A1*, *A2*, *A3*) hat die Markierung (0,0,1,0,1), wobei zusätzlich noch ein Token am *JoinNode* zwischen *T1* und *A2* am Eingang von *T1* vorhanden ist. In diesem Fall wird durch die Ausführung von *A1* anschließend auch *A2* ausgeführt. Der fortgeführte Pfad lautet *T1*, *A1*, *T2*, *A3*, *A1*, *A2*. Im Vergleich dazu ist ein möglicher Pfad für die parallele Ausführung von beiden Auslösern *T1*, *T2*, *A1*, *A3*, *A1*, *A2*. In beiden Fällen werden die gleichen *Actions* ausgeführt. Diese Eigenschaft ist für die in dieser Arbeit betrachteten *Aktivitätsdiagramme* ausreichend.

Alternativ kann auch mithilfe von speziellen, wertbehafteten *ObjectTokens* erreicht werden, dass diese nur von bestimmten *Actions* akzeptiert werden. Auf diese Weise kann die zuvor beschriebene Konstellation vermieden werden. Als weitere Möglichkeit können Bedingungen an den Kanten verwendet werden, die sicherstellen, woher ein Token stammt.

Algorithmisches Vorgehen

Die Fälle sollen nachfolgend an der Umformung in Abbildung 8 verdeutlicht werden. Dabei werden schrittweise alle Duplikate ausgehend vom Startknoten entfernt und zudem wird gezeigt, dass das vorgestellte Konzept auch für mehrere Duplikate hintereinander angewendet werden kann, auch wenn diese nicht in gleicher Abfolge vorliegen. Aus Darstellungsgründen wird auf *ForkNodes* verzichtet, die notwendig sind, wenn eingefügte *JoinNodes* als Vorgänger verwendet werden.

In der Ausgangssituation in Abbildung 8a sind drei unterschiedliche Auslöser *T1*, *T2* und *T3* als Startknoten und drei unterschiedliche *Actions A4*, *A5* und *A6* als Endknoten vorhanden. Im ersten Schritt in Abbildung 8b werden zunächst nach allen Knoten der identifizierten Duplikate *JoinNodes* eingefügt. Die *JoinNodes* ändern den Ablauf des *Aktivitätsdiagramms* nicht, sondern dienen als die direkten Nachfolger der redundanten Knoten.



(a) Ausgangssituation (b) Einfügen von *JoinNodes* (c) Duplikat *A1* entfernt

(d) Duplikat *A2* entfernt (e) Duplikat *A3* entfernt

Abb. 8: Vorgehen der Duplikateliminierung

Beim Entfernen des ersten Duplikats bestehend aus den Knoten *A1*, *A1'* und *A1"* wird vor dem zusammengefassten Knoten ein *MergeNode* und hinter dem Knoten ein *DecisionNode* eingefügt. Alle Vorgänger werden mit dem *MergeNode* verbunden. Der *DecisionNode* erhält ausgehende Kanten zu den folgenden *JoinNodes*. Anschließend werden die Vorgänger und Nachfolger der ursprünglichen redundanten Knoten miteinander verbunden.

Die gleiche Umformung wird im Folgenden für die Duplikate *A2*, *A2'* und *A2"* in Abbildung 8d sowie für *A3*, *A3'* und *A3"* in Abbildung 8e angewendet. Hierbei ist erkennbar,

dass die Umformung auch angewendet werden kann, wenn sich die Abfolge der Duplikate unterscheidet. Zudem ist es unerheblich, in welcher Reihenfolge die Duplikate eliminiert werden.

5 Anwendung am Beispiel

Zur Eliminierung aller identifizierten Duplikate in einem *Aktivitätsdiagramm* wird das in Kapitel 4 präsentierte Konzept für alle Duplikate angewendet. Abgeschlossen wird es mit der Minimierung der Anzahl der Kanten und der eingefügten Elemente.

Bei der Anwendung des Konzepts auf das Beispiel aus Kapitel 2 müssen die vier dort herausgestellten Duplikate betrachtet werden. Das Endergebnis ist in Abbildung 9 dargestellt.

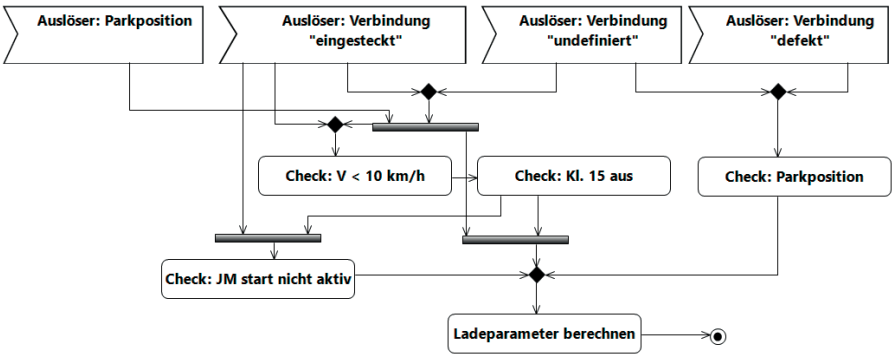


Abb. 9: Funktionsablauf der Fahrzeugfunktion *Ladeparameter berechnen* duplikatfrei

Die redundanten Knoten der Duplikate von Auslöser *Verbindung "eingesteckt"* und von Auslöser *Verbindung "undefiniert"* sind zusammengeführt. Die ausgehenden Kanten sind den einzelnen verbleibenden Knoten zugeordnet. Eingehende Kanten sind nicht vorhanden. Ebenfalls sind die Knoten der Duplikate von den Überprüfungen *V<10 km/h* und *Kl.15 aus* zusammengeführt. Zum Erhalt der Funktionalität sind zwei *JoinNodes* und ein *MergeNode* im Diagramm hinzugekommen. Alle weiteren eingefügten *ControlNodes* und Kanten, die während des Vorgehens entstanden sind, werden durch Vereinfachungen wieder entfernt.

Im vorliegenden Beispiel sind vier Duplikate eliminiert worden, welche aus jeweils zwei identischen Elementen bestehen. Das Vorgehen ist jedoch unabhängig von der Anzahl der identischen Knoten anwendbar. Ebenso gelten keine Beschränkungen für die Anzahl der Knoten aus verschiedenen Duplikaten, die hintereinander im Ablauf stehen. Dies liegt darin begründet, dass das Vorgehen schrittweise angewendet wird und somit immer vergleichbare Situationen auftreten.

Ein Nachteil der Eliminierung der Duplikate besteht laut Meinung von Anwendern darin, dass die Darstellung an Übersichtlichkeit verlieren kann, da die zuvor bereits bekannte Anordnung nicht mehr sofort erkennbar ist. Diagramme sind jedoch ab einer bestimmten

Größe nur noch eingeschränkt für Menschen sofort verständlich [Mi56]. Darüber hinaus ist die Übersichtlichkeit als Basis für weitere Schritte in der Entwicklung nicht das primäre Kriterium. Unabhängig davon kann eine übersichtlichere Darstellung mit Duplikaten als bereits bekannte Darstellung für den Anwender weiterhin verwendet werden. In diesem Fall ist eine Synchronisation bei Änderungen der beiden Diagramme beispielsweise durch eine entsprechende Verlinkung anzustreben.

6 Zusammenfassung und Ausblick

Im vorliegenden Beitrag wird erstmalig ein Konzept präsentiert, um in UML *Aktivitätsdiagrammen* auftretende Duplikate automatisch zu eliminieren. Bei dem vorgestellten Konzept bleibt die Funktionalität erhalten. Dies ermöglicht eine bessere Verwendung des Diagramms in darauffolgenden Schritten der Entwicklung. Es sind zwei Fälle dargestellt, welche unterschieden werden. Je nach Fall wird eine andere Umformung angewendet. Das resultierende Ergebnis kann anschließend auf nicht mehr notwendige *ControlNodes* und Kanten untersucht werden, die durch die Duplikateliminierung entstehen. Diese *ControlNodes* und Kanten sind wieder zu entfernen. Zusätzlich erfolgt eine Betrachtung von auftretenden Besonderheiten der vorgestellten Fälle und von Zyklen.

Es handelt sich dabei um einen ersten Schritt zur Eliminierung von Duplikaten unter Beibehaltung der Funktionalität. In zukünftigen Arbeiten wird adressiert, inwiefern die hier dargestellten Fälle vollständig sind. Damit einhergehend wird an dem formalen Nachweis gearbeitet, dass die Nutzung des vorgestellten Konzepts das Verhalten der UML *Aktivitätsdiagramme* erhält. Darüber hinaus beschäftigen sich die Autoren mit weiteren Elementen, die in *Aktivitätsdiagrammen* vorkommen und hier noch nicht betrachtet worden sind.

Angrenzend an diese Vorhaben stellt die Identifikation der redundanten Elemente einen weiteren Aspekt dar, dessen Automatisierung zu erheblichen Zeitersparnissen führt und hilft die Qualität der Diagramme zu verbessern.

Literaturverzeichnis

[AD97] Apfelbaum, Larry; Doyle, John: Model Based Testing. In: Software Quality Week Conference. S. 296–300, 1997.

[Br06] Broy, Manfred: Challenges in Automotive Software Engineering. In: Proceedings of the 28th international conference on Software engineering. ACM, S. 33–42, 2006.

[BSF03] Boger, Marko; Sturm, Thorsten; Fragemann, Per: Refactoring Browser for UML. In (Aksit, Mehmet; Mezini, Mira; Unland, Rainer, Hrsg.): Objects, Components, Architectures, Services, and Applications for a Networked World: International Conference NetObjectDays, NODe 2002 Erfurt, Germany, October 7–10, 2002 Revised Papers. Springer Berlin Heidelberg, S. 366–377, 2003.

[CST02] Costal, Dolors; Sancho, Maria-Ribera; Teniente, Ernest: Understanding Redundancy in UML Models for Object-Oriented Analysis. In (Pidduck, Anne Banks; Ozsu, M. Tamer;

Mylopoulos, John; Woo, Carson C., Hrsg.): Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Toronto, Canada, May 27–31, 2002 Proceedings. Springer Berlin Heidelberg, S. 659–674, 2002.

[Da01] och Dag, Johan Natt; Regnell, Björn; Carlshamre, Pär; Andersson, Michael; Karlsson, Joachim: Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven Development. In: Proc. 7th Int. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01). 2001.

[LEM02] Liu, WenQian; Easterbrook, Steve; Mylopoulos, John: Rule-Based Detection of Inconsistency in UML Models. In: Workshop on Consistency Problems in UML-Based Software Development. Jgg. 5, 2002.

[Mi56] Miller, George A: The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychological review, 63(2):81, 1956.

[Su01] Sunyé, Gerson; Pollet, Damien; Traon, Yves; Jézéquel, Jean-Marc: Refactoring UML Models. In (Gogolla, Martin; Kobryn, Cris, Hrsg.): UML 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools: 4th International Conference Toronto, Canada, October 1–5, 2001 Proceedings. Springer Berlin Heidelberg, S. 134–148, 2001.

[UM15] UML, OMG: UML 2.5 Specification. URL http://www.omg.org/spec/UML/2.5/, 2015.