Testfallgenerierung aus Statecharts und Interaktionsdiagrammen

Dehla Sokenou
Fakultät IV - Elektrotechnik und Informatik, Fachgebiet Softwaretechnik,
Sekr. FR5-6, TU Berlin, Franklinstr. 28/29, D-10587 Berlin
dsokenou@cs.tu-berlin.de

Zusammenfassung

UML-Modelle bieten durch die Vielfalt der Ausdrucksmöglichkeiten ihrer Diagramme unterschiedliche Sichten auf ein System. Gleichzeitig führt dieses zu einer Verteilung von Information über viele Diagramme und erschwert somit auch das Testen auf Basis von UML-Modellen. Daher müssen Verfahren entwickelt werden, die möglichst viele Sichten der Modelle zum Testen berücksichtigen. Vorgestellt wird ein Ansatz, der Statecharts und Interaktionsdiagramme zur Testfallgenerierung kombiniert.

1 Einleitung

Der Zustandsraum eines objektorientierten Systems ist meist zu groß, um einen Test nur auch ansatzweise vollständig durchzuführen. Damit ist die Aussagekraft eines Tests gering. Um dieses Problem zu lösen, ist eine geeignete Auswahl der Testfälle nötig. Ein pragmatischer Ansatz ist die Konzentration auf typische Anwendungsfälle des Systems, die dem Entwickler besonders wichtig waren.

UML-Interaktionsdiagramme (Sequenz- und Kollaborationsdiagramme) beschreiben mögliche Nachrichtenfolgen zwischen Objekten in einem System. Sie stellen in der Regel typische Normal- und Fehlerfälle dar. Werden sie als Grundlage für den Test verwendet, kann insbesondere das Kommunikationsverhalten des zu testenden Systems untersucht werden.

Auf den ersten Blick scheint der Test auf Basis von Interaktionsdiagrammen intuitiv zu sein. Jedes Interaktionsdiagramm definiert einen Testfall bzw. eine Menge von Testfällen, wenn einer der in der UML 2.0 [8] definierten Operatoren wie *alt* (Alternative), *opt* (Option), *loop* (Iteration) oder *par* (Parallelität) verwendet wird. Interaktionsdiagramme definieren die an einem Szenario beteiligten Objekte und somit die Beziehungen von Objekten innerhalb eines Systems.

Es ist jedoch festzustellen, daß die modellierten Szenarien nur einen Teilausschnitt des Systemverhaltens darstellen. Meist gibt es weder einen zeitlichen Bezug (wann tritt das modellierte Verhalten auf) noch sind die Voraussetzungen für ein Szenario gegeben (in welchem Zustand müssen sich die beteiligten Objekte befinden). Die modellierten Szenarien beschreiben nur mögliche Nachrichtenfolgen, es wird keine Aussage über Vollständigkeit getroffen. Auf eine Nachricht kann das modellierte Verhalten auftreten, muß es aber nicht. Andere Szenarien als die modellierten sind möglich.

Interaktionsdiagramme enthalten in der Regel keine Informationen, die als Sollwerte benutzt werden können. Es gibt zwar den Operator *neg*, der Nachrichtenfolgen kennzeichnet, die nicht erlaubt sind, so daß das Auftreten einer entsprechenden Sequenz als fehlgeschlagener Test zu werten ist. Überwiegend werden jedoch positive Sequenzen modelliert.

Gleichzeitig eignen sich Interaktionsdiagramme gut für den Integrationstest, da sie Nachrichten modellieren, die zwischen verschiedenen Objekten im System geschickt werden. So sind sie eine gute Basis für die Generierung von Testfällen.

Einen Teil der Probleme beim Test auf Basis von Interaktionsdiagrammen versucht der vorgestellte Ansatz zur Testfallgenerierung zu lösen. Es werden zusätzliche Informationen aus UML-Statecharts und OCL-Constraints zum Test herangezogen, um Sollwerte zu gewinnen und die am Szenario beteiligten Objekte zu initialisieren.

Im Abschnitt 2 wird die grundlegende Technik zur Gewinnung von Testfällen aus Sequenzdiagrammen und Statecharts anhand eines Beispiels beschrieben und mit anderen Ansätzen verglichen. Zusätzlich wird die Initialisierung von Testszenarien mit Hilfe von Zustandsautomaten und die Gewinnung zusätzlicher Testfälle aus der Kombination von Statecharts und Sequenzdiagrammen gezeigt. Abschnitt 3 gibt einen kurzen Überblick über die Gewinnung von Sollwerten aus Statecharts und OCL-Constraints. Eine Zusammenfassung und ein Ausblick finden sich in Abschnitt 4.

2 Testfälle aus Statecharts und Sequenzdiagrammen

Einige Ansätze beschreiben die Generierung von Testfällen aus Zustandsautomaten. Die meisten streben eine vollständige Überdeckung des Zustandsautomaten durch Traversierung an.

In [5] wird beispielsweise durch Traversierung

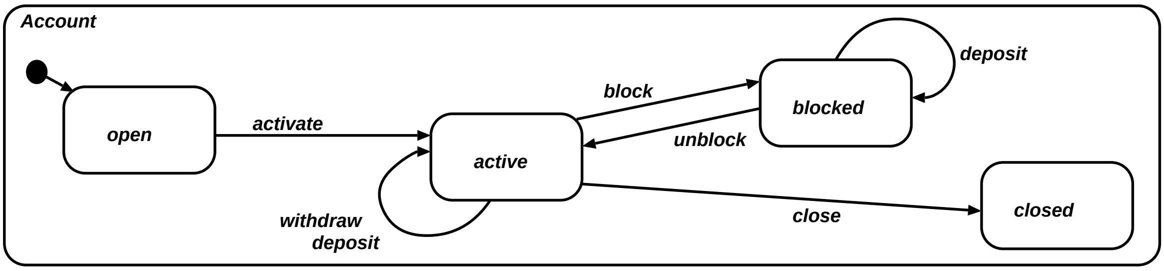

Abbildung 1: Statechart der Klasse `Account`.

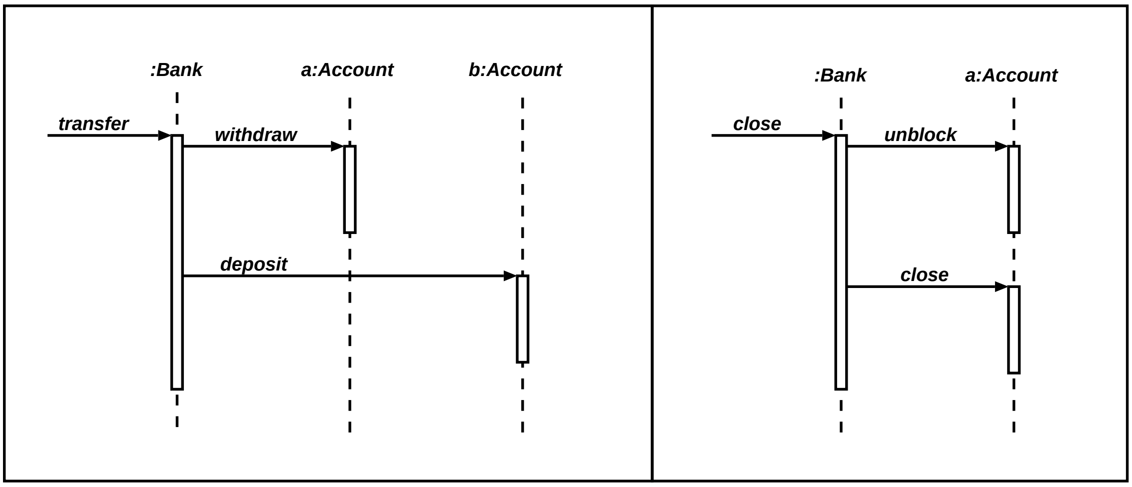

Abbildung 2: Szenarien Überweisung und Schließung eines gesperrten Kontos.

des Zustandsautomaten ein Baum erstellt, der die Zustände und Transitionen vollständig überdeckt. Dieser dient als Grundlage für die Erstellung von Testfällen. Jeder Zyklus im Zustandsautomaten wird beim Test einmal ausgeführt. [6] beschreibt eine Technik, die zu einer größeren Anzahl von Testfällen führt. Grundlage ist ebenfalls die Traversierung eines Zustandsautomaten, in diesem Fall eines UML-Statecharts. Durch nichtdeterministische Auswahl der Zweige kann die Menge der Testfälle beliebig groß werden. Zyklen können dabei mehrfach durchlaufen werden.

Beide Techniken erzeugen Testfälle, die zu einer gleichmäßigen Überdeckung des Zustandsautomaten führen, was nicht unbedingt dem realen Einsatz des betrachteten Systems entspricht. Ein Zustand im Statechart kann viele konkrete Zustände des zugehörigen Objekts repräsentieren oder nur einen Zustand (betrachtet man z.B. ein Konto, so könnte ein modellierter Zustand `empty` nur den konkreten Zustand eines Objekts repräsentieren, in dem der Kontostand gleich Null ist, ein Zustand `credit` dagegen alle konkreten Zustände, in denen der Kontostand größer als Null ist). Zudem wird jeweils nur ein Statechart betrachtet, so daß sich die Verfahren im objektorientierten Test nur für den Klassentest eignen, nicht jedoch für den Integrationstest.

Für den Integrationstest besonders geeignet scheinen die Interaktionsdiagramme. Einige Ansätze generieren Testfälle für den Integrationstest aus Sequenzdiagrammen (z.B. [2]). Probleme bereiten jedoch die Initialisierung der beteiligten Objekte und die Gewinnung von Sollwerten. Als Lösung wird meist das Sequenzdiagramm mit Informationen angereichert.

Unsere Idee ist nun, die Vorteile beider Techniken zu kombinieren, um so die Statecharts für den Integrationstest nutzbar zu machen und die in den Sequenzdiagrammen fehlenden Informationen aus den Statecharts zu gewinnen. Dabei versuchen wir, ausschließlich die in der UML 2.0 spezifizierten Sprachbestandteile vorauszusetzen und dem Entwickler der Modelle möglichst viel semantischen Spielraum zu lassen.

Der vorgestellte Ansatz basiert auf den *UML Protocol Statemachines*, die zur Modellierung von Objektlebenszyklen dienen. Ereignisse sind Methodenaufrufe (*Call Events*). Methoden, die zustandsverändernd sind, können nur in den Zuständen aufgerufen werden, in denen sie im Statechart eine Transition auslösen, in allen anderen Zuständen ist ihr Aufruf verboten (implizite Vorbedingung). Zustandsneutrale Methoden (*Query-Methoden*) werden im Statechart nicht modelliert und sind in jedem Zustand aufrufbar. Es werden keine explizten Aktionen und Aktivitäten modelliert[1]. Eine Übertragung der vorgestellten Technik auf die zweite Art der UML-Statecharts, die *Behavioural State Machines*, ist nur bedingt möglich.

Wir konzentrieren uns auf Sequenzdiagramme. Dies ist keine Einschränkung, da sich Sequenzdiagramme mit wenig Informationsverlust in Kollaborationsdiagramme transformieren lassen und umgekehrt und somit die präsentierte Technik auf Kollaborationsdiagramme übertragbar ist. Wir nehmen an, daß die Sequenzdiagramme sequentielles Verhalten beschreiben, daß heißt, der Aufruf einer Methode an einem Objekt wird zunächst vollständig abgearbeitet, bevor die nächste Methode desselben Objekts ausgeführt wird[2].

Als Beispiel dient das in Abbildung 1 dargestellte Statechart eines Bankkontos. Nach der Erzeugung eines Kontos muß es zunächst aktiviert werden, dann sind Ein- und Auszahlungen möglich, solange das Konto nicht gesperrt oder geschlossen wird. Ist das Konto gesperrt, kann nur noch eingezahlt werden.

Abbildung 2 zeigt zwei mögliche Szenarien, an denen Objekte von Typ Konto beteiligt sind. Links ist eine Überweisung von einem Konto auf ein anderes modelliert, rechts die Schließung eines gesperrten Kontos.

Unser Ansatz berücksichtigt zwei Testphasen, den Klassen- und den Integrationstest.

Für den Klassentest werden die am Szenario beteiligten Objekte einzeln betrachtet. Es werden die entsprechenden Nachrichtenfolgen an ein Objekt als Testsequenzen für dieses Objekt benutzt. Die in der UML 2.0 definierten Operatoren in Sequenzdiagrammen dienen dabei als Methodensequenzspezifikation, so daß der Ansatz aus [4] auf Sequenzdiagramme übertragbar ist. So dient z.B. der *loop*-Operator mit unterer und oberer Schranke zur Definition, wie oft ein Zyklus im Statechart beim Test durchlaufen werden soll.

Beim Integrationstest versuchen wir, die modellierten Szenarien zu provozieren. Jedes Szenario steht dabei für eine Menge von Testfällen, die durch Initialisierung und die Betrachtung von Fehlerfällen, die zum Test der Robustheit des Systems dienen, genauer spezifiziert werden.

Die folgenden beiden Abschnitte betrachten die Initialisierung der am Szenario beteiligten Objekte und die Gewinnung weiterer Testfälle (in der Regel Fehlerfälle) gesondert.

2.1 Initialisierung von Testszenarien

Wie bereits in [2] beschrieben, ist eine Initialisierung der am Szenario beteiligten Objekte notwendig. Anders als die dort beschriebenen Lösungen verfolgt der hier vorgestellte Ansatz die Gewinnung der Informationen zur Initialisierung aus den UML-Statecharts.

Für das Szenario `Überweisung` ist es nötig, daß das `Account`-Objekt `a` sich in einem Zustand befindet, in dem der Methodenaufruf `withdraw` möglich ist, dies gilt analog für das `Account`-Objekt `b` und die Methode `deposit`. Aus dem Statechart für die Klasse `Account` läßt sich ermitteln, daß sich das Objekt `a` im Zustand `active` befinden muß, das Objekt `b` in einem der Zustände `active` oder `blocked`. Damit gibt es zwei mögliche gültige Testfälle für das entsprechende Szenario, jeweils einen für jeden möglichen gültigen Zustand des Objekts `b`.

Bei Nachrichtenfolgen an ein Objekt, wie im Szenario `Schließung` an das Objekt `a`, ist ein Zustand des Systems zu finden, in dem diese Nachrichtenfolge möglich ist. In unserem Beispiel ist das der Zustand `blocked`. Wird in diesem Zustand die Methode `unblock` aufgerufen, wird `a` in den Zustand `active` überführt. Im aktiven Zustand ist ein Aufruf der zweiten Methode `close` möglich.

Zu berücksichtigen ist, daß eine korrekte Initialisierung der beteiligten Objekte keine Garantie darstellt, daß das System sich verhält, wie die Modellierung der Sequenzdiagramme es vorsieht. Dies ist nur dann der Fall, wenn die modellierten Sequenzdiagramme das System vollständig modellieren.

2.2 Betrachtung zusätzlicher Testfälle

Die im letzten Abschnitt beschriebene Initialisierung deckt alle Fälle ab, in denen ein modelliertes Szenario ablaufen kann. Beim Testen sollte man jedoch auch mögliche Fehlerfälle betrachten, um die Robustheit des Systems zu untersuchen.

Im Beispiel ist eine Überweisung nur möglich, wenn sich beide Konten im aktiven Zustand befinden, das Empfängerkonto kann darüberhinaus gesperrt sein.

Die Bank muß die Vorbedingung von Methoden einhalten, die sie an anderen Objekten aufruft. In unserem Fall ist das Verhalten der Bank zu prüfen, wenn eines der beiden Kontos noch nicht aktiviert oder bereits geschlossen ist oder wenn das Konto des Auftraggebers gesperrt ist. In allen diesen Fällen darf es zu keinem Verlust von Geld kommen.

Es müssen also zusätzliche Testfälle erzeugt werden, die speziell das Verhalten der beteiligten Objekte im Fehlerfall prüfen. Dazu wird vor der Ausführung des betrachteten Szenarios eines der beteiligten Objekte in einen Zustand versetzt, in dem ein entsprechender Methodenaufruf nicht möglich ist (z.B. das Objekt `a` in den Zustand `blocked`). Zu erwarten ist in diesem Fall, daß die Bank die Überweisung nicht ausführt und somit die Vorbedingung der Methoden nicht verletzt (Sollwert).

3 Gewinnung von Sollwerten

Die vorgestellte Technik eignet sich zur Testfallgenerierung, nicht zur Bestimmung von Sollwerten.

[1] Als Aktion eines Methodenaufrufs gilt in der Semantik der UML implizit die Ausführung der entsprechenden Methode.

[2] Diese Einschränkung verbietet nicht den Einsatz von Operatoren wie dem Paralleloperator *par* in Sequenzdiagrammen.

Zur Sollwertbestimmung wird eine andere Technik verwendet. Als Testorakel dienen Statecharts, die ausführbar gemacht werden (ähnlich wie in [7]), in Kombination mit Zusicherungen in Form von OCL-Constraints. Für diese Kombination definiert die UML 2.0 einen präzise Semantik.

Im Beispiel ist für das Szenario `Überweisung` eine Nachbedingung der Methode `transfer` in OCL formuliert worden. Diese sagt aus, daß die Summe aller Kontostände nach einer Überweisung gleich bleiben muß. Die Wahrheit dieser Aussage wird zur Laufzeit des Tests überprüft.

Die Statecharts werden mit Hilfe von Object Teams [3] implementiert, dabei werden anders als in [7] die Statecharts-Eigenschaften Hierarchie, Historie, Parallelität und Nichtdeterminismus unterstützt.

Die ausführbaren Statecharts laufen parallel zum zu testenden System. Die zu testenden Objekte benachrichtigen ihr zugeordnetes Statechart bei jedem Methodenaufruf und prüfen, ob der Aufruf im aktuellen Zustand des zu testenden Objekts gültig und der resultierende Zustand korrekt ist.

Damit ist die Berechnung der Sollwerte unabhängig von der Testfallgenerierung. Beide Techniken können separat verwendet werden, so daß auch ein anderes Testorakel für das vorgestellte Verfahren zur Testfallgenerierung möglich ist.

4 Zusammenfassung

Der vorgestellte Ansatz bietet insbesondere im Hinblick auf die Initialisierung von Testszenarien Vorteile. Durch die Gewinnung von Informationen aus verschiedenen UML-Modellen ist eine Anreicherung einzelner Diagramme nicht notwendig. Es werden keine Erweiterungen definiert, die nicht konform sind zum UML-Standard und es wird auf die Einführung neuer Sprachbestandteile, z.B. neuer Stereotypen, verzichtet.

Die Voraussetzung für die Anwendung der Technik ist die Konsistenz der einzelnen Modelle zueinander. So müssen Namen von Elementen, z.B. Methoden, in allen Modellen gleich sein und sich auch in der Implementierung wiederfinden. Es sollte, insbesondere im Hinblick auf die Sollwertberechnung, ein Mapping zwischen den abstrakten Zuständen im Statechart und den konkreten Zuständen in den Objekten (definiert durch Belegungen der Instanzvariablen) geben. Nur so kann eine Überprüfung des Zustandes der zu testenden Objekte vorgenommen werden. Zudem beschränkt sich der Ansatz auf die *Protocol State Machines*, deren Semantik eng eingegrenzt ist.

Zur Zeit arbeiten wir an der praktischen Umsetzung der vorgestellten Testfallgenerierungstechnik. Zustände, in denen eine bestimmte Methodensequenz möglich ist, sollen durch Anbindung an ein Modelchecking-Tool bestimmt werden. Dazu wird das Testziel negiert (*Trap Property*, z.B. *es gibt keinen Zustand, in dem die betrachtete Methodensequenz möglich ist*). Der Modelchecker liefert dann den gesuchten Zustand als Gegenbeispiel. Diese Technik wird für ein ähnliches Problem unter anderem in [1] zum Testen eingesetzt.

Die Umsetzung der Sollwertgewinnung aus UML-Statecharts ist weitgehend abgeschlossen. Zur Zeit wird diese um die Sollwerte aus den OCL-Constraints erweitert.

Ziel ist es, sowohl Testfälle als auch Sollwerte aus den UML-Modellen zu gewinnen. In Zukunft sollen zudem zusätzlich Testdaten aus UML-Modellen gewonnen werden können.

Literatur

[1] A. Engels, L. Feijs, S. Mauw. Test Generation for Intelligent Networks Using Model Checking. In *3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, *Lecture Notes in Computer Science*, Band 1217, Enschede, Niederlande, 1997. Springer-Verlag.

[2] F. Fraikin, T. Leonhardt. SeDiTeC - Testing Based on Sequence Diagram. In *17th IEEE International Conference on Automated Software Engineering (ASE)*, Edinburgh, Großbritannien, 2002.

[3] S. Herrmann. Object Teams: Improving Modularity for Crosscutting Collaborations. In *Objects, Components, Architectures, Services, and Applications for a Networked World (Net.ObjectDays Conference)*, *Lecture Notes In Computer Science*, Band 2591, Erfurt, 2002. Springer-Verlag.

[4] S. H. Kirani, W.-T. Tsai. Method Sequence Specification and Verification of Classes. *Journal of Object-Oriented Programming*, 7(6), 1994.

[5] D. Kung, N. Suchak, J. Gao, P. Hsia, Y. Toyoshima, C. Chen. On Object State Testing. In *Proceedings of 18th International Computer Software and Applications Conference (COMPSAC)*, Taipeh, Taiwan, 1994. IEEE Computer Society Press.

[6] D. Seifert, S. Helke, T. Santen. Test Case Generation for UML Statecharts. In *Perspectives of System Informatics (PSI)*, *Lecture Notes In Computer Science*, Band 2890, Novosibirsk, Russland, 2003. Springer-Verlag.

[7] D. Sokenou. Ein Werkzeug zur Unterstützung zustandsbasierter Testverfahren für Java-Klassen. In *13. Treffen der Fachgruppe Test, Analyse und Verifikation von Software (TAV)*, *Softwaretechnik-Trends*, Band 19, München, 1999. Gesellschaft für Informatik (GI).

[8] *Unified Modeling Language Specifications, Version 2.0.* Object Management Group (OMG), *http://www.uml.org*, 2004.