Testfallgenerierung aus Statecharts und Interaktionsdiagrammen

Dehla Sokenou

TU Berlin

Softwaretechnik

Motivation

- Warum Testen mit Hilfe von UML?
  - UML verbreitete Spezifikationssprache in der Objektorientierung
  - Früher Beginn der Testaktivitäten
  - Nutzung spezifikationsbasierter Testfälle zur Aufdeckung von Fehlern zwischen Modellierung und Implementierung

Motivation

- Interaktionsdiagramme beschreiben (mögliche) Nachrichtenfolgen
  - Scheinbar waren die modellierten Szenarien dem Entwickler besonders wichtig
- Es fehlt an
  - Sollwerten
  - Semantik (besser in UML 2.0)
- Aber: Gut geeignet für Integrationstest
  - Kommunikation zwischen Objekten
- Kombination mit anderen Modellen kann Probleme lösen

Testfälle aus Statecharts

- Einige Ansätze zur Testfallgenerierung aus Statecharts

⇒ Probleme

- Statechart beschreiben Objektverhalten ⇒ Klassentest
- Meist vollständige Überdeckung angestrebt
- Meist gleichmäßige Überdeckung
- Auswahl der Testfälle?

Testfälle aus Interaktionsdiagrammen

- Einige Ansätze zur Testfallgenerierung aus Interaktionsdiagrammen

⇒ Integrationstest

⇒ Probleme

  - Vollständigkeit
  - Sollwerte
  - Initialisierung von Szenarien

  ⇒ Lösung meist Anreicherung von Diagrammen

Lösung

- Kombination Statechart, Interaktionsdiagramm und OCL-Constraints
- Grundidee
  - Reihenfolge der Methodenaufrufe aus Interaktionsdiagramm
  - Initialisierung / Fehlerfälle aus Statecharts
  - Sollwerte aus Statecharts und OCL-Constraints

Grundlagen

- Basis ist UML 2.0

⇒ Sequenzdiagramme

- Neue Operatoren (z.B. alt, opt, loop, par)

⇒ Protocol State Machines

⇒ OCL-Constraints

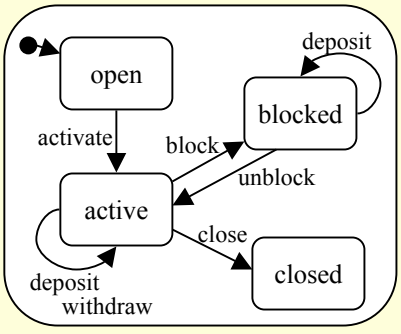
- Zusicherungen (Vor-, Nachbedingungen, Klassen- und Zustandsinvarianten)

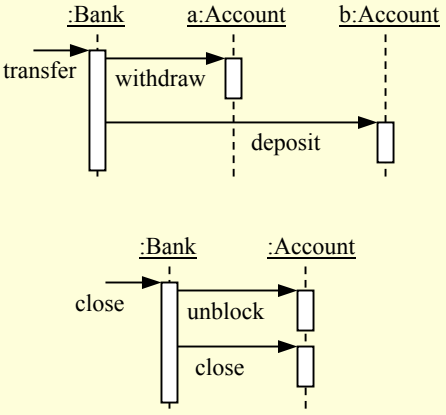
⇒ Semantik

- Kombination Statecharts - OCL-Constraints

Ein Beispiel

- Bankkonto





Klassentest

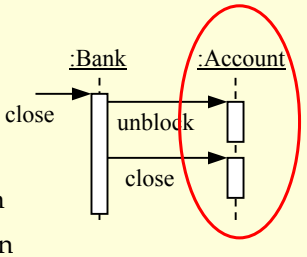
- Am Szenario beteiligte Objekt einzeln betrachtet
  - Methodensequenz

    hier: Objekt vom Typ *Konto*

    Methodensequenz:

    unblock; close
  - Besondere Bedeutung Operatoren

    als Methodensequenzspezifikation

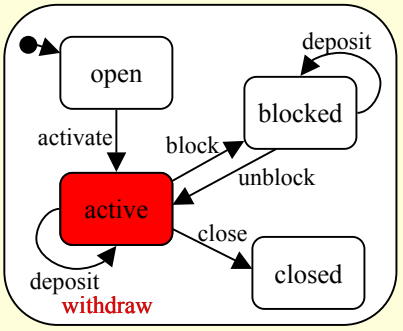


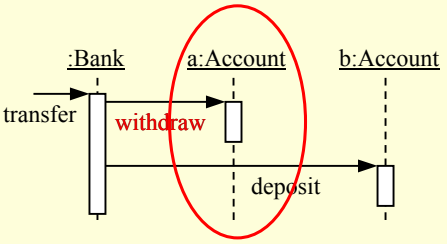
Integrationstest

- Betrachtung Gesamtszenario
- Provozierung der modellierten Szenarien
  - Szenarien meist nicht vollständig modelliert
  - Initialisierung
- Jedes Szenarium definiert Menge von Testfällen
  - Einzelne Testfälle unterscheiden sich in Startzuständen der beteiligten Objekte
  - Betrachtung von Normal- und Fehlerfällen

Initialisierung von Szenarien

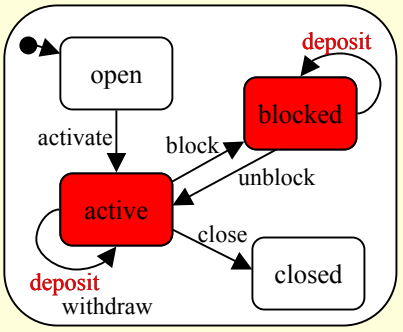
- Identifikation aller Zustände in Statecharts, in denen Szenario möglich (Normalfälle)

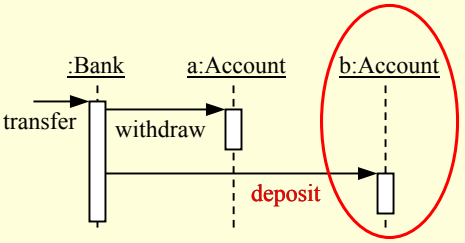




Initialisierung von Szenarien

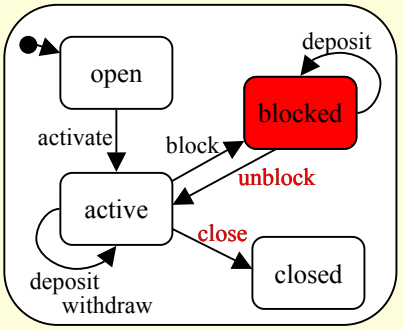
- Identifikation aller Zustände in Statecharts, in denen Szenario möglich (Normalfälle)

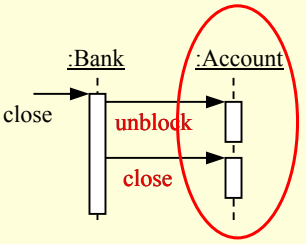




Initialisierung von Szenarien

- Initialisierung bei Methodensequenzen
  - Modelchecking (*Trap-Property*)



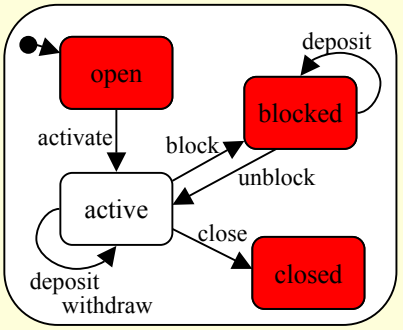


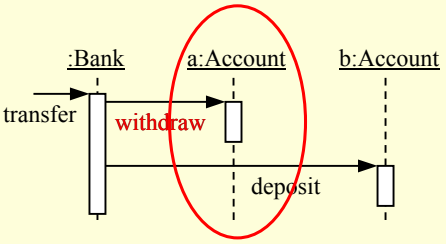
Initialisierung von Szenarien

- Für Szenario *Überweisung* 2 Testfälle
  - Initialisierung Objekt *a*: *new; activate*
    Initialisierung Objekt *b*: *new; activate*
    Objekt *Bank*: *transfer*
  - Initialisierung Objekt *a*: *new; activate*
    Initialisierung Objekt *b*: *new; activate; block*
    Bank-Objekt: *transfer*
- Für Szenario *Schließung* 1 Testfall
  - Initialisierung *Account*-Objekt *a*: *new; activate; block*
    Bank-Objekt: *close*

Fehlerfälle

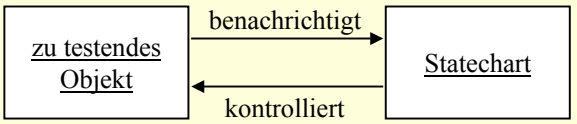
- Konstruktion von Zuständen, in denen Szenario nicht möglich ist (Fehlerfälle)





Testorakel

- Generierung ausführbarer Statecharts (als Object Teams)
  - Eventbasierte Berechnung der Folgezustände
  - Auswertung zur Laufzeit, dadurch keine vorzeitige Sollwertberechnung für einzelnen Testfall nötig
  - Aspektorientierte Einbindung in zu testendes System



Testorakel

- Statechart-Eigenschaften Hierarchie, Historie, Parallelität und Nichtdeterminismus unterstützt
- Kombination mit OCL-Constraints
- Testorakel unabhängig von Testfallgenerierung
- Keine Veränderung des zu testenden Systems
  - Keine Änderung Quellcode
  - Kein Neucompilieren

Voraussetzungen

- Modelle müssen vorhanden sein
- Konsistenz zwischen einzelnen Modellen und zwischen Modell und Implementierung
  - Namen
  - Mapping zwischen abstrakten Statechart-Zuständen und konkreten Objektzuständen (Zustandsinvariante)
- Beschränkung auf Protocol State Machines

Zusammenfassung

- Informationen aus Statechart erhöhen Testbarkeit von Sequenzdiagrammen
  - Initialisierung
  - Sollwerte
- Keine Erweiterung der UML nötig
- Weitere Testfälle leicht zu spezifizieren durch Erstellen neuer Sequenzdiagramme

Ausblick

- Zur Zeit Umsetzung des vorgestellten Ansatzes
  - Statecharts als Testorakel fast abgeschlossen
  - Erweiterung um OCL in Arbeit
  - Anbindung an Modelchecker zur Testfallgenerierung
- Weitere Modelle einbeziehen
- Testdaten erzeugen