“Bewertender Vergleich und Erweiterung unterschiedlicher UML-Simulatoren zur Bestimmung der Modellüberdeckung”

Halbzeitvortrag zur Diplomarbeit
von Dominik Schindler
am 19.10.2006

Übersicht

1. **Einleitung**
    1. Motivation
    2. Aufgabe
    3. Beispiel
2. **Anforderungskatalog**
    1. Pflichtanforderungen
    2. Optionale Anforderungen
    3. Simulatoren
3. **Was kann wie simuliert werden?**
    1. Simulierbare UML Modelle
    2. Elemente von Zustandsmaschinen und deren Einschränkungen
4. **Eclipse UML2 Modeling Framework**
5. **Simulator**
    1. Genereller Ablauf
    2. Transitionen/Trigger
    3. Guards/Operationen
6. **Ausblick**
7. **Demonstration**

1. Einleitung

Motivation

Aufgabenstellung

Beispiel

1. Einleitung

- Heutige Softwaresysteme besitzen eine hohe Komplexität und hohe Sicherheitsanforderungen
- Formale und semiformale Entwicklungssprachen und -methoden wie die Unified Modeling Language (UML) haben deshalb an Bedeutung gewonnen
- **Vorteil der UML:** Das Verhalten von Modellen kann (eingeschränkt) von Werkzeugen automatisch simuliert werden
- Dazu existieren einige Werkzeug mit zum Teil sehr unterschiedlichen Eigenschaften

1. Motivation

- Durch die Simulation eines Modells können frühzeitig Fehler erkannt werden
  - Während der Spezifikation ist es relativ leicht und preiswert, Fehler zu korrigieren (man muss nur die Spezifikation ändern)
  - In der Designphase ist das Beheben von Fehlern schon aufwändiger (man muss sowohl die Spezifikation als auch das Design ändern)
  - Ist die Software schon fast fertig gestellt oder gar ausgeliefert, kann die Behebung von Fehlern sehr teuer werden

  → Je früher das Verhalten simuliert wird, desto besser

  → **Aber:** In den frühen Phasen kann weniger simuliert werden, da weniger Information bzw. evtl. sogar zu wenig Information zur Verfügung steht

1. Motivation

- Das Verhalten eines Modells ist manuell schwer bis unmöglich zu simulieren
  - Große Modelle sind nur schwer überschaubar
  - Der Aufwand steht in keinem Verhältnis zum Nutzen
  - Die Qualität der Ergebnisse ist besser, da weniger fehleranfällig
  - Eine automatische Simulation ist schneller als eine manuelle
  - Manuelle Simulation ist demotivierend

1. Motivation

Nachteil einer automatischen Simulation

- Einschränkungen bezüglich der Mächtigkeit der UML 2.0 (siehe später)
- Verhalten kann nicht mehr in natürlicher Sprache spezifiziert werden
- Simuliert werden kann nur die unterstützte UML Version
- Simulationswerkzeuge erfordern Einarbeitungszeit
- Kein Nichtdeterminismus möglich

2. Aufgabenstellung

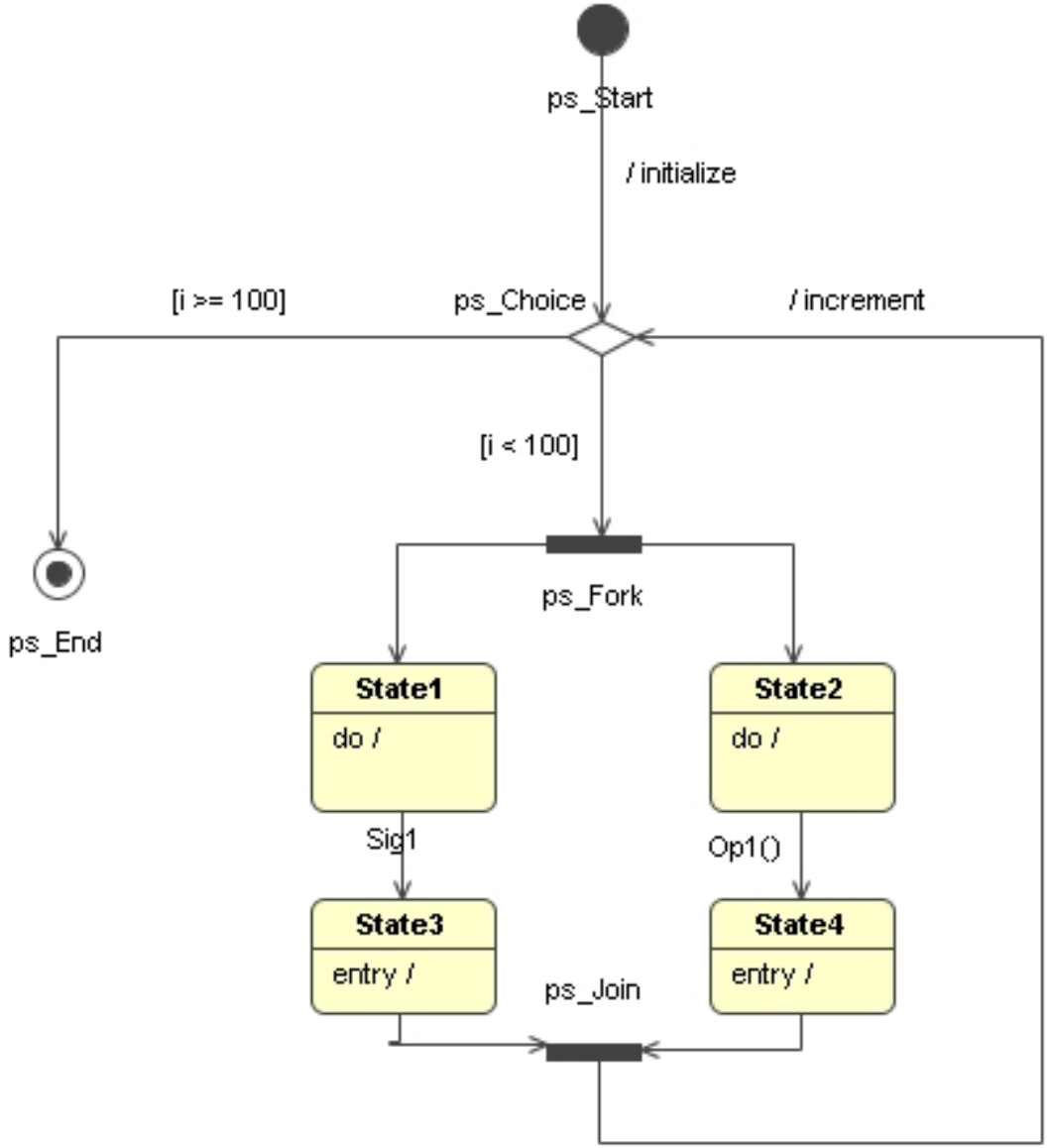
- Es sind zunächst Eigenschaften zu identifizieren, die ein Simulationswerkzeug bei der automatischen Simulation von UML-Modellen unter bestimmten Eingabedaten zur Messung der Modellüberdeckung benötigt.

- Anschließend sollen verschiedene existierende Werkzeuge im Hinblick auf die Erfüllung dieser Eigenschaften verglichen und bewertet werden.

- Darauf aufbauend soll für ausgewählte Modelle eine Schnittstelle definiert werden, die die Anbindung eines solchen Modellsimulators an das bestehende Projekt ermöglicht.

2. Beispiel

- Trigger:
  - CallTrigger: Op1()
  - SignalTrigger: Sig1
- Guards:
  - i >= 100
  - i < 100
- Effekte:
  - Init(): Initialisiert die Variable i := 0
  - Inc(): i := i + 1



2. Anforderungskatalog

Pflichtanforderungen

Optionale Anforderungen

Simulatoren

2. Pflichtanforderungen

- Automatische Simulation des UML Modells
  - zu simulierendes Modell
  - Liste der Ereignisse
  - evtl. die anfängliche Belegung der verwendeten Variablen

  ➢ Pfad durch das dynamische Modell, Liste der überdeckten Elemente

- Automatische Auswertung der Ausdrücke von Guards und Operationen
  - Das Verhalten der Operationen muss durch Ausdrücke modelliert werden
  - Verwendete Ausdrücke müssen „berechenbar“ sein
  - Z.B. durch einen Expression Parser

2. Pflichtanforderungen

- Standardisierte Formate
  - Eingabe: z.B. XMI (XML Metadata Interchange)
  - Ausgabe: z.B. XML
- Schnittstelle zu bestehendem Projekt
  - Existierender Simulator: Definition einer Schnittstelle
    - API
    - Scriptsprache
  - Neuentwicklung: Entwicklung unter Berücksichtigung des verwendeten Datenmodells
- Muss die UML 2.0 unterstützen (1)
  - Die UML 2.0 wurde gegenüber der UML 1.3 verbessert und erweitert, z.B.
    - Ein- und Austrittspunkte sowie Terminatoren wurde eingeführt
    - Tiefe History-Zustände können auch Ziel einer Transition innerhalb des enthaltenen Zustands sein (also nicht nur von außen)
    - ...
  - UML 2.0 ist Standard

2. Pflichtanforderungen

- Alle gängigen Elemente eines Diagramms müssen unterstützt werden
  - Bsp. Zustandsautomat
    - Regionen
    - Einfache und komplexe Zustände
    - Gängige Pseudozustände (Start- und Endzustand, Auswahl, Fork/Join, usw.)
    - Transitionen
      - Guards
      - gängige Trigger (Signal-, Call und AnyTrigger)
    - Effekte und Operationen

2. Optionale Anforderungen

- Sollte mehrere dynamische UML Modelle unterstützen
  - bzw. eine einfache Übertragung auf andere dynamische UML Modelle sollte möglich sein
  - z.B. auf Aktivitäten-Diagramme
- Sollte Deadlocks erkennen können, d.h.,
  - Es sind keine ausgehenden Transitionen mehr vorhanden, die schalten können und der aktuelle Zustand ist keine Endzustand
  - Zwei Transitionen warten auf ein Signal, das der jeweils andere erzeugt
- Sollte wenig Kosten und nicht zu kompliziert sein
  - Außerdem keine weiteren externen Programme benötigen
- Sollte plattformunabhängig sein

2. Simulatoren

Artisan Studio von Artisan:

- „UML-2.0- und SysML-basiertes Modellierungstools für die Entwicklung von Echtzeitssystemen und -software“

Automatische Simulation des Modells	✓
Automatische Auswertung der Ausdrücke von Guards und Operationen	✓
Standardisierte Formate	✓ (XMI, Rational Rose)
Schnittstelle zu bestehendem Projekt	✓ (VBScript)
UML 2.x Unterstützung	✓
Unterstützung aller gängigen Elemente eines Diagramms	✓
Anzahl unterstützter dynamischer UML Modelle	2
Erkennen von Deadlocks	?
Sollte wenig Kosten und nicht zu kompliziert sein	×
Plattformunabhängigkeit	× (nur Windows)

2. Simulatoren

TAU G2 von Telelogic

- „Modellgetriebene Entwicklungsumgebung, die anspruchsvolle Funktionalitäten für jede Phase des Lebenszyklus bietet.“

Automatische Simulation des Modells	✓
Automatische Auswertung der Ausdrücke von Guards und Operationen	✓
Standardisierte Formate	✓/ × (Rational Rose)
Schnittstelle zu bestehendem Projekt	?
UML 2.x Unterstützung	✓
Unterstützung aller gängigen Elemente eines Diagramms	× (Parallelität, Hierarchie)
Anzahl unterstützter dynamischer UML Modelle	2
Erkennen von Deadlocks	✓
Sollte wenig Kosten und nicht zu kompliziert sein	×
Plattformunabhängigkeit	✓

2. Simulatoren

- Existierende Simulatoren erfüllen die Anforderungen nur ungenügend

- **Entwicklung eines Simulators, der möglichst alle Anforderungen erfüllt**

- Weitere Vorteile:
  - Die Entwicklung kann in Hinblick auf das verwendete Datenmodell des bestehenden Projekts erfolgen
  - Der Quelltext des Simulators ist verfügbar, d.h., er ist
    - anpassbar
    - testbar
    - wartbar

3. Was kann simuliert werden?

Dynamische UML Modelle

Elemente eines Zustandsautomaten und deren Einschränkungen

3. Was kann simuliert werden?

- Eine Simulation macht nur bei dynamischen Modellen Sinn
- In der UML werden diese auch als *Verhaltensdiagramme* bezeichnet
  - Use-Case Diagramm
  - Aktivitätendiagramm
  - Zustandsautomat
  - Sequenzdiagramm
  - Kommunikationsdiagramm
  - Timing-Diagramm
  - Interaktionsübersichtsdiagramm
- Diese Modelle stellen die Verhaltensspezifikation aus jeweils unterschiedlichen Blickwinkeln dar

3. Was kann simuliert werden?

- Ein Zustandsautomat spezifiziert das Verhalten durch Zustände, die ein Classifier einnehmen kann, und Übergänge zwischen den Zuständen

- Vereinfachende Annahmen:

  - Der Zustandautomat befindet sich zu einem Zeitpunkt in genau einer Zustandskonfiguration, d.h., in ein oder mehreren Zustände

  - Der Übergang von einem Zustand zu einem anderen erfolgt ohne Verzögerung.

3. Was kann simuliert werden?

- **Elemente eines Zustandsautomaten**
  - Start- und Endzustand
  - „Echte“ Zustände: Einfache und zusammengesetzte Zustände
  - Unterzustandsautomatenzustände
  - Transitionen
  - Pseudozustände mit charakteristischen Eigenschaften
  - Regionen
- Damit die Elemente simuliert werden können, müssen bei einigen Elementen Einschränkungen gemacht werden

3. Was kann simuliert werden?

- **Startzustand:**
  - Darf nur einmal pro Zustandautomat vorhanden sein
  - Darf keine eingehenden Transitionen besitzen
  - Bei abgehende Transitionen dürfen keine Trigger vorhanden sein
  - Der Guard der Transition muss leer sein
  - Jeder Startzustand darf höchstens eine ausgehende Transition besitzen
- **Endzustand:**
  - Darf mehrmals pro Zustandautomat vorhanden sein
  - Darf keine ausgehenden Transitionen besitzen
  - In einem Endzustand wird kein weiters Verhalten ausgeführt

3. Was kann simuliert werden?

- **„Echte“ Zustände:**
  - Eintritts- / Austritts- / Zustandverhalten muss wie folgt spezifiziert werden:
    - Operation (in einer formalen Sprache)
    - Aktivitäten-Diagramm
    - Zustandsautomat
  - Komplexe Zustände:
    - Bei orthogonalen Regionen muss der Zugriff auf Variablen koordiniert werden

3. Was kann simuliert werden?

Unterzustandsautomatenzustände:

- Es gelten die gleichen Einschränkungen wie bei den Zustandsautomaten

Transitionen (1)

- Determinismus:
  - Mehrere ausgehende Transitionen mit gleichem Guard müssen disjunkte Trigger besitzen
  - Mehrere ausgehende Transitionen mit gleichen Triggern müssen disjunkte Guards besitzen
- Guards müssen in einer formalen Sprache spezifiziert worden sein

3. Was kann simuliert werden?

Transitionen (2)

- Das Verhalten einer Transition muss wie folgt spezifiziert worden sein
  - Operation (in einer formalen Sprache)
  - Aktivitäten-Diagramm
  - Zustandsautomat
- Trigger:
  - CallTrigger: Operation (s.o.)
  - TimeTrigger:
    - Benötigt eine (virtuelle) Zeit
    - Muss in einer formalen Sprache definiert worden sein, z.B. `if ( time > 23:00 )`
  - AnyTrigger: Darf nur schalten, wenn kein anderer Trigger geschaltet hat

3. Was kann simuliert werden?

- **Pseudozustände**
  - Entscheidung/Kreuzung:
    - Für Guards gelten die gleichen Einschränkungen wie bereits bei den Transitionen beschrieben
  - Gabelung/Vereinigung:
    - Ausgehende bzw. eingehende Transitionen dürfen keine Trigger und keinen Guard besitzen
  - History-Zustand:
    - Darf nur einmal in einer Region vorkommen → Eindeutige Zuordnung eines History-Zustands zu einer Region
    - Wichtig, da beim Verlassen einer Region die zuletzt aktiven Zustände gespeichert werden müssen

4. Eclipse UML2 Modeling Framework

4. Eclipse UML2 Modeling Framework

- Das Eclipse UML2 Modeling Framework ist eine EMF-basierte Implementierung des UML™ 2.x Metamodels
- Das EMF (Eclipse Modeling Framework) ist ein Java-Framework zum Erzeugen, Abfragen, Manipulieren, Serialisieren (als XMI) und Validieren strukturierter Modelle
- EMF kann aus einem Modell automatisiert Code erzeugen
- Ziel des Projekts ist eine verwendbare Implementierung des Metamodells
  - Zur Unterstützung bei der Entwicklung von Modellierungs-Werkzeugen
  - Um den Austausch von Modellen zu erleichtern (XMI Schema)

5. Simulator

Genereller Ablauf
Transitionen, Trigger,
Guards, Operationen
Einschränkungen an das UML-Modell

5. Simulator

Allgemeiner Ablauf:

- Der Simulator arbeitet rundenbasiert
- Zu Beginn jeder Runde wird ein Zustand aus der Liste der aktiven Zustände entnommen
- Wenn eine Transition geschaltet hat, wird der Zielzustand in die Liste der aktiven Zustände aufgenommen
- Eventuell vorhandenes Verhalten (Eingangs-/Ausgangsverhalten, Effekt der Transition) wird entsprechend ausgeführt
- Hat **keine einzige** ausgehende Transition geschaltet, wird der alte Zustand wieder zur Liste hinzugefügt

5. Simulator

Zustand

- Hat ein Zustand keine ausgehende Transition, die schalten kann, wird in dieser Runde das Zustands-Verhalten ausgeführt
- Wird ein Zustand verlassen, wird das Ausgangs-Verhalten ausgeführt und das Zustandsverhalten beendet
- Sobald der neue Zustand zur Liste der aktiven Zustände hinzugefügt wird, wird dessen Eingangs-Verhalten ausgeführt
- In der nächsten Runde wird das Zustands-Verhalten des neuen Zustands das erste mal ausgeführt

5. Simulator

Transition/Trigger (1)

- Es werden alle ausgehenden Transitionen des aktuellen Zustands überprüft
- Besitzt eine Transition keinen Trigger und keinen Guard, wird der Zielzustand zur Liste hinzugefügt und der Effekt ausgeführt
- Hat eine Transition keinen Trigger, aber einen Guard, wird der Guard überprüft
  - Ist der Guard wahr, wird der Zielzustand zur Liste hinzugefügt
  - Ist der Guard falsch, wird nichts unternommen
  - Falls kein Guard wahr, aber ein „else“ Guard vorhanden ist, wird der Zielzustand dieser Transition in die Liste aufgenommen

5. Simulator

Transition/Trigger (2)

- Besitzt eine Transition einen oder mehrere Trigger, wird in der Liste der aktiven Ereignisse nach diesen Triggern gesucht
  - Ist das Ereignis in der Liste vorhanden und ein eventuell vorhandener Guard wahr, wird das Ereignis „konsumiert“ und der Zielzustand zur Liste der aktiven Zustände hinzugefügt
  - Ist das Ereignis nicht in der Liste vorhanden, wird der alte Zustand zur Liste hinzugefügt, sofern nicht schon vorhanden
  - Ereignisse können sein: CallTrigger, SignalTrigger, AnyTrigger
- Wenn keine ausgehende Transition geschaltet hat und auch keine Transition mit einem AnyTrigger vorhanden ist, wird der alte Zustand zur Liste wieder hinzugefügt

5. Simulator

Transition/Trigger (3)

- Der Effekt einer Transition kann ein Aufruf einer Operation oder das Senden eines Signals sein
  - Beim Aufruf einer Operation wird das entsprechende Ereignis (CallEvent, SignalEvent) zur Lise der aktiven Ereignisse hinzugefügt
  - Bei einem CallEvent wird außerdem das Verhalten der Operation ausgeführt
  - Dieses Verhalten wird durch das Ändern von Variabelwerten modelliert
  - Die Wertänderungen erfolgen durch (mathematische) Ausdrücke, die als Constraint an das Modell angehängt werden

5. Simulator

Guard/Operation:

- Die in den Guards und den Operationen verwendeten Variablen müssen in der Klasse als Attribut deklariert worden sein
- Initialisierung der Variablen:
  - Bei der Deklaration kann ein Defaultwert angegeben werden, z.B. `i: Integer = 0`
  - Die Initialisierung kann auch durch eine Operation erfolgen, z.B. `init( i )` mit `i := 0` als Constraint
- Als Guard können bool'sche Ausdrücke mit diesen Variablen verwendet werden, z.B. `( i < 100 ) && ( i > -100 )`
- Das Verhalten von Operationen wird durch das Ändern dieser Variablen über Ausdrücke modelliert
- Operationen können auf diese Variablen lesend und schreibend zugreifen, z.B. `i := i + 1`

5. Simulator

- Einschränkungen an das zu simulierende UML Modell
  - Der Zustandsautomat muss valid bzgl. der UML 2 sein, d.h. z.B.
    - Startzustände dürfen keine eingehenden Transitionen, Endzustand keine ausgehenden Transitionen besitzen
    - Transitionen, die von einem Gabelung wegzeigen, bzw. zu einer Vereinigung hinzeigen dürfen keine Guards und Trigger besitzen
    - …
  - Der Zustandautomat muss deterministisch sein, sonst ist der Verlauf der Simulation unbestimmt und die Ergebnisse falsch
  - Nur bool‘sche Guards mit definierten Variablen sind erlaubt
  - Das Verhalten von Operationen muss durch einen oder mehrere Ausdrücke als Constraint erfolgen

5. Ausblick

Was ist bereits fertig?

Was ist noch zu tun?

5. Ausblick

- Was ist bereits fertig?
  - Einfache Modelle können simuliert werden (Auswahl, Fork/Join)
  - Signal- und CallTrigger werden unterstützt
  - Guards werden ausgewertet, Effekte werden ausgeführt
  - Variablen können initialisiert und durch Operationen verändert werden
    - Durch einen Default-Wert bei der Deklaration der Variablen
    - Durch eine initiale Operation
  - Vorläufige GUI
  - Magic Draw UML2 Modelle können geladen werden
  - Momentan noch keine automatische Simulation

5. Ausblick

- Was ist noch zu tun?
  - Fehlende Elemente
    - Zusammengesetzter Zustand
    - Unterzustandsautomatenzustand
    - Parallelität
    - History-Zustand
  - TimeTrigger, ChangeTrigger
    - TimeTrigger: Einführen einer virtuellen Zeit (z.B. die Rundenanzahl), ähnlich wie bei der DES (Diskreten Event Simulation)
    - ChangeTrigger: Erkennbar durch Überwachung der „Laufzeitumgebung“ (verwendet für das Lesen und Speichern der Variablen)
  - Automatisierte Simulation
    - Durch Verwenden einer Liste von Ereignissen als Eingabe (evtl. mit Zeitangabe)
    - Speichern des Pfads, bzw. der Modellüberdeckung

6. Demonstration

Literaturverzeichnis

- [UML2] *UML 2 glasklar. Praxiswissen für die UML - Modellierung und Zertifizierung*, Chris Rupp, Jürgen Hahn, und Stefan Queins
- [EMF] *http://www.eclipse.org/emf/*, Stand 4.10.2006
- [EMF-UML2] *http://www.eclipse.org/uml2/*, Stand 4.10.2006
- [TAU] *http://www.telelogic.de*, Stand 17.10.2006
- [Artisan] *http://www.artisansw.com/*, Stand 13.10.2006