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

Endvortrag zur Diplomarbeit
von Dominik Schindler
am 19.12.2006

Übersicht

1. Einleitung

- Motivation
- Aufgabenstellung
- Beispiel

1. Einleitung

- Der UML-Simulator soll im Zusammenhang mit dem Projekt UnITeD verwendet werden
- Dieses Projekt versucht, den Test hochzuverlässiger und sicherheitskritischer Software zu automatisieren, um die Testkosten zu reduzieren und die Erkennung von Restfehlern in komplexer Software zu optimieren
- Die Testdaten werden durch evolutionäre Verfahren automatisch aus UML-Diagrammen ermittelt
- Die ermittelten Testdaten werden anschließend als Eingabe für den UML-Simulator verwendet
- Der Simulator führt das zugrunde liegende UML-Modell mit den Daten aus und liefert die überdeckten Elemente zurück.
- Anschließend wird die Fitness der Testdaten anhand der Simulationsergebnisse bestimmt
- Je nach Güte der Testdaten werden diese weiterverwendet oder verworfen

1.1. Motivation

- **Weiterer Vorteil:** Durch die Simulation eines Modells können frühzeitig Fehler im Modell 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

1.1. Motivation

Nachteil einer automatischen Simulation

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

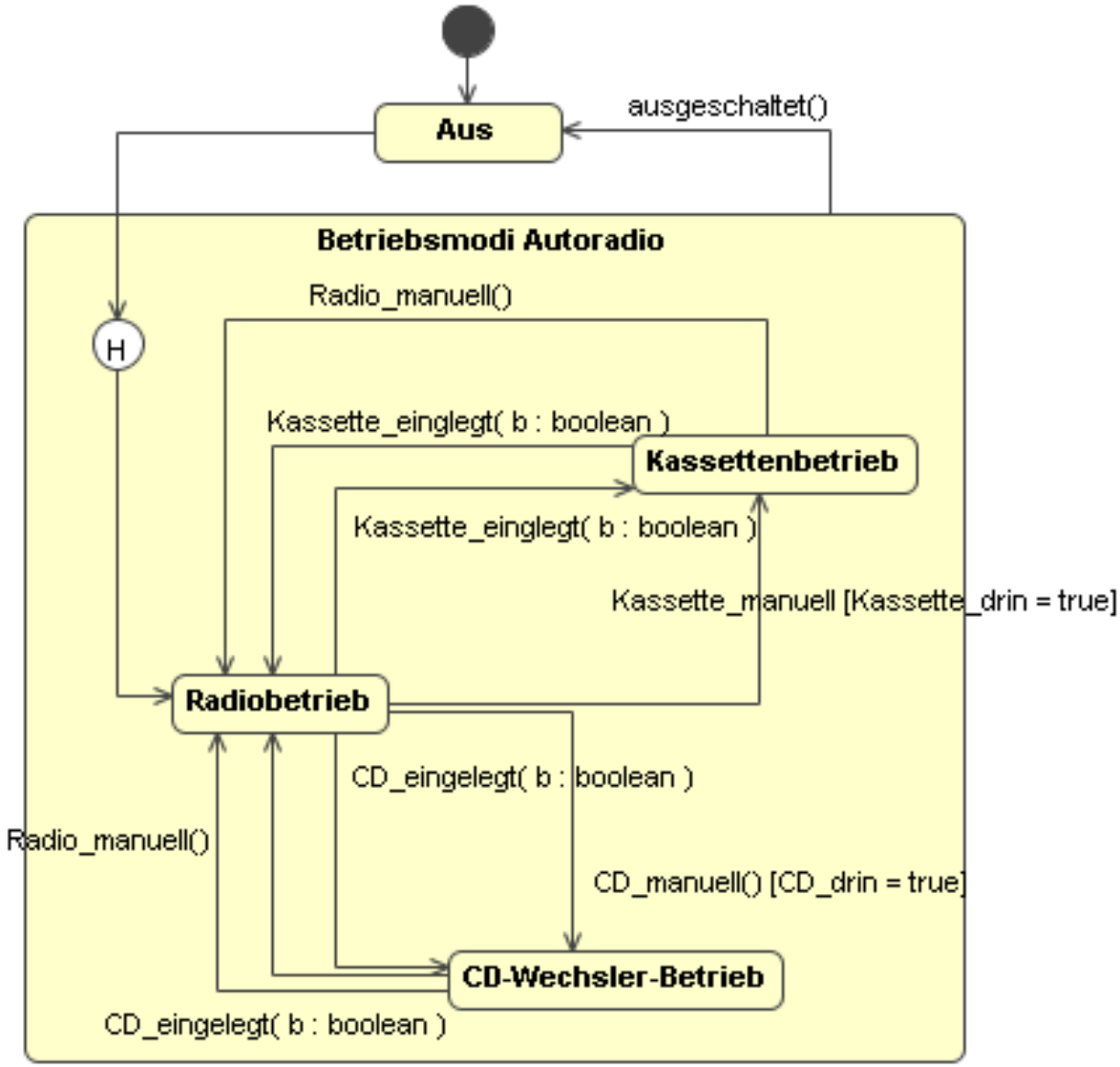
1.2. Aufgabenstellung

- 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.

1.3. Beispiel


Aus
ausgeschaltet()
Betriebsmodi Autoradio
Radio_manuell()
H
Kassette_einglegt(b : boolean)
Kassettenbetrieb
Kassette_einglegt(b : boolean)
Kassette_manuell [Kassette_drin = true]
Radiobetrieb
CD_eingelegt(b : boolean)
Radio_manuell()
CD_manuell() [CD_drin = true]
CD-Wechsler-Betrieb
CD_eingelegt(b : boolean)

2. Anforderungskatalog

- Pflichtanforderungen
- Optionale Anforderungen
- Simulatoren

2. Anforderungskatalog

Pflichtanforderungen	Optionale Anforderungen
Automatische Simulation des UML Modells	Sollte mehrere dynamische UML Modelle unterstützen
Automatische Auswertung der Ausdrücke von Guards und automatisches Ausführen des Verhaltens von Operationen	Sollte Deadlocks und Starvation erkennen können
Muss standardisierte Formate verwenden	Sollte wenig Kosten und nicht zu kompliziert sein
Schnittstelle zu bestehendem Projekt muss implementierbar sein	Sollte plattformunabhängig sein
Muss die UML 2.0 unterstützen (keine älteren Versionen)	Sollte effizient sein
Alle gängigen Elemente eines Diagramms müssen unterstützt werden	

2.1. Pflichtanforderungen

- Automatische Simulation des UML Modells
  - Eingabe
    - Zu simulierendes Modell
    - Liste der Ereignisse
    - Evtl. die anfängliche Belegung der verwendeten Variablen
    - Abbruchbedingung
  - ➢ Ausgabe: Liste der überdeckten Elemente

- Automatische Auswertung der Ausdrücke der Guards
  - Verwendete Ausdrücke müssen berechenbar sein
  - Z.B. durch einen Expression Parser

- Automatisches Ausführen des Verhaltens der Operationen
  - Das Verhalten von Operationen muss modelliert werden können
  - Operationen können die in den Guard-Ausdrücken verwendeten Variablen ändern

2.1. Pflichtanforderungen

- Muss standardisierte Formate verwenden
  - Wichtig für die Implementierung der Schnittstelle
  - Eingabe: z.B. XMI (XML Metadata Interchange)
  - Ausgabe: z.B. XML
- Schnittstelle zum bestehendem Projekt muss implementierbar sein
  - Existierender Simulator: Definition einer Schnittstelle
    - API
    - Skriptsprache
  - Neuentwicklung: Entwicklung unter Berücksichtigung des verwendeten Datenmodells
- Muss die UML 2.0 unterstützen
  - Die UML 2.0 wurde gegenüber der UML 1.3 verbessert und erweitert, z.B.
    - Ein- und Austrittspunkte sowie Terminator wurde eingeführt
    - Tiefe History-Zustände können auch Ziel einer Transition innerhalb des enthaltenen Zustands sein (also nicht nur von außen)
    - ...
  - Semantik hat sich bei einigen Elementen geändert (z.B. Aktivitätsdiagramme)
  - UML 2.0 ist mittlerweile Standard

2.1. Pflichtanforderungen

- Alle gängigen Elemente eines Diagramms müssen unterstützt werden
  - Bsp. Zustandsautomat
    - Einfache und zusammengesetzte Zustände
    - Gängige Pseudozustände (Start- und Endzustand, Entscheidung, Gabelung/Vereinigung)
    - Transitionen
      - Guards
      - Gängige Trigger (SignalTrigger und CallTrigger)
    - Operationen
  - Bsp. Aktivitätsdiagramm
    - Aktions- und Objektknoten
    - Kanten (bedingt, gewichtet)
    - Start-, Endknoten
    - Verzweigungs-, Verbindungsknoten
    - Synchronisations- und Parallelisierungsknoten

2.2. Optionale Anforderungen

- Sollte mehrere dynamische UML Modelle unterstützen
  - Bzw. eine einfache Übertragung auf andere dynamische UML Modelle sollte möglich sein
  - Z.B. auf Aktivitätsdiagramme, Sequenzdiagramme
- Sollte Deadlocks bzw. Starvation erkennen können
  - Zwei Transitionen warten auf ein Signal, das der jeweils andere erzeugt
  - Es sind keine ausgehenden Transitionen mehr vorhanden, die schalten können und der aktuelle Zustand ist keine Endzustand
- Sollte wenig Kosten und nicht zu kompliziert sein
  - Außerdem keine weiteren externen Programme benötigen
- Sollte plattformunabhängig sein
- Sollte effizient sein und mit den Ressourcen schonend umgehen
  - Im Rahmen des Projekt wird der Simulator sehr häufig und parallel verwendet

2.3. Simulatoren

Artisan Studio von Artisan

- Verwendbar für das Software- und Systems Engineering (SysML)
- Unterstützt Zustandsautomaten und Sequenzdiagramme
- Sehr gute Unterstützung der Teamarbeit (durch gemeinsames Repository)
- Unterstützt das simultane Round Trip Engineering
- Sehr gute Integration in Visual Studio

TAU G2 von Telelogic

- Häufig verwendet für das Protocol Engineering und Systems Engineering
- Unterstützt Protokoll-Zustandsautomaten und Sequenzdiagramme
- Kann automatisch Java und C++/C#-Code aus dem Modell erzeugen
- Automatische Generierung, Ausführung und Verwaltung von Testfällen
- Unterstützt die Microsoft Visual Studio .NET 2003 und Eclipse

2.3. Simulatoren

	Artisan Studio	Telelogic TAU G2
Automatische Simulation des Modells	✓	✓
Automatische Auswertung der Ausdrücke von Guards und Ausführen des Verhaltens von Operationen	✓	✓
Verwendet standardisierte Formate	✓ (XMI, Rational Rose)	✓/ × (Rational Rose)
Schnittstelle zu bestehendem Projekt durch	✓ (VBScript)	×
UML 2.0 Unterstützung	✓	✓
Unterstützung aller gängigen Elemente eines Diagramms	✓	× (Parallelität, Hierarchie)
Anzahl unterstützter dynamischer UML Modelle	2	2
Erkennen von Deadlocks bzw. Starvation	?	✓
Sollte wenig Kosten und nicht zu kompliziert sein	×	×
Plattformunabhängigkeit	× (nur Windows)	✓
Effizienz und Ressourcenbedarf	× (schlecht)	× (schlecht)

2.3. Simulatoren

JavaFSM der Uni Hamburg

- Entwurf des zugrunde liegenden Automaten mit Hilfe des Editors
- Simulation von Mealy- und Moore Zustandsautomaten
- Simulation durch Verändern der Eingangswerte und Taktgebung (schrittweise Simulation)
- Überprüfen des Automaten auf Korrektheit
- Definition der logischen Übergangsbedingungen (Und, Oder und Nicht)
- Quelloffen

2.3. 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
    - Simulation eines Use-Case Diagramms selbst nicht sinnvoll, da dieses nur die Funktionalität des Systems beschreibt
    - Simulation des dynamischen Diagramms, das das Verhalten eines Use-Cases beschreibt
  - Aktivitätsdiagramm
    - Petri-Netz-Semantik (Token)
  - Zustandsautomat
    - Semantik endlicher Automaten

3. Was kann simuliert werden?

- Sequenzdiagramm
  - Semantik ähnlich Message Sequence Charts aus der Telekommunikation
- Kommunikationsdiagramm
  - Form des Sequenzdiagramms, wobei der Fokus auf den interagierenden Partnern liegt
- Interaktionsübersichtsdiagramm
  - Aktivitätsdiagramm mit Interaktionen bzw. Interaktionsreferenzen statt Aktionen und Objektknoten

- Diese Modelle stellen die Verhaltensspezifikation aus jeweils unterschiedlichen Blickwinkeln dar

3. Was kann simuliert werden?

Zustandsautomat

- 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 einem oder mehreren Zuständen

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

3. Was kann simuliert werden?

- **Elemente eines Zustandsautomaten**
  - Start- und Endzustand
  - 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?

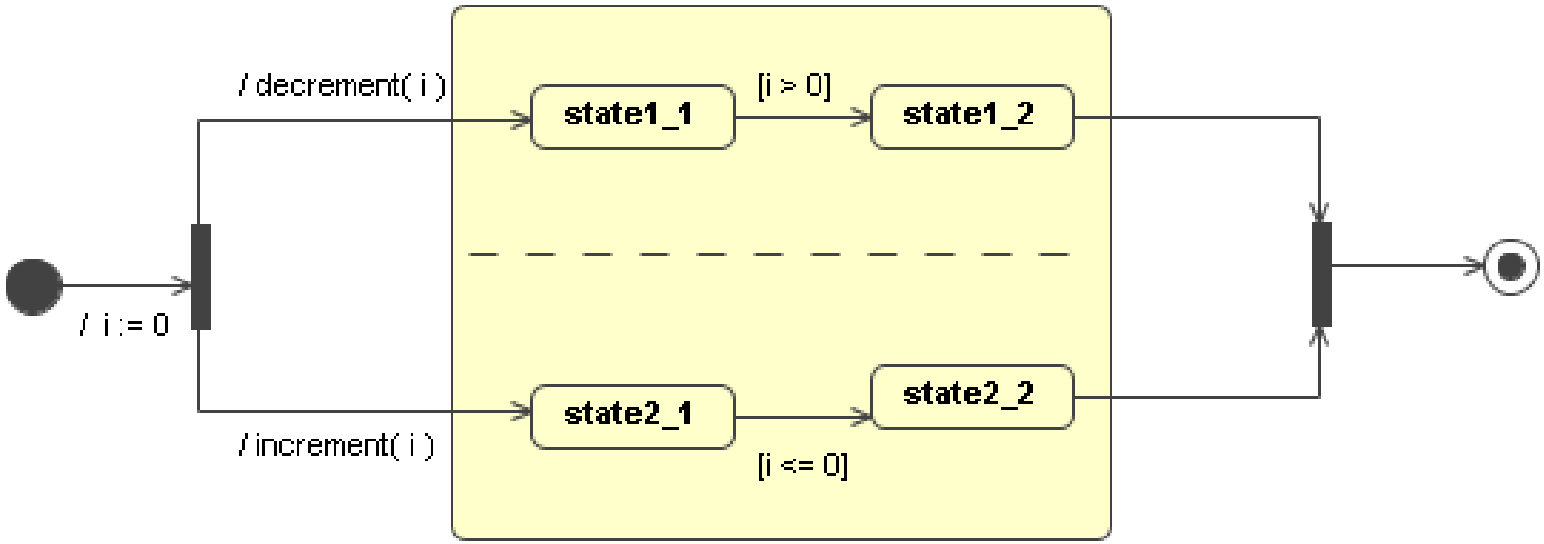
- **Startzustand:**
  - Darf nur einmal pro Zustandautomat/Region vorhanden sein
  - Darf keine eingehenden Transitionen besitzen
  - Bei abgehende Transitionen dürfen keine Trigger vorhanden sein
  - Der Guard der Transition muss leer sein
  - Jeder Startzustand darf höchstens eine ausgehende Transition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

3. Was kann simuliert werden?

- **Zustände:**
  - Eintritts- / Austritts- / Zustandverhalten muss rechnerlesbar spezifiziert werden
    - Operation (in einer formalen Sprache)
    - Aktivitätsdiagramm
    - Zustandsautomat
  - Zustandsverhalten wird umgehend nach dem Eintrittsverhalten und vor dem Austrittsverhalten ausgeführt

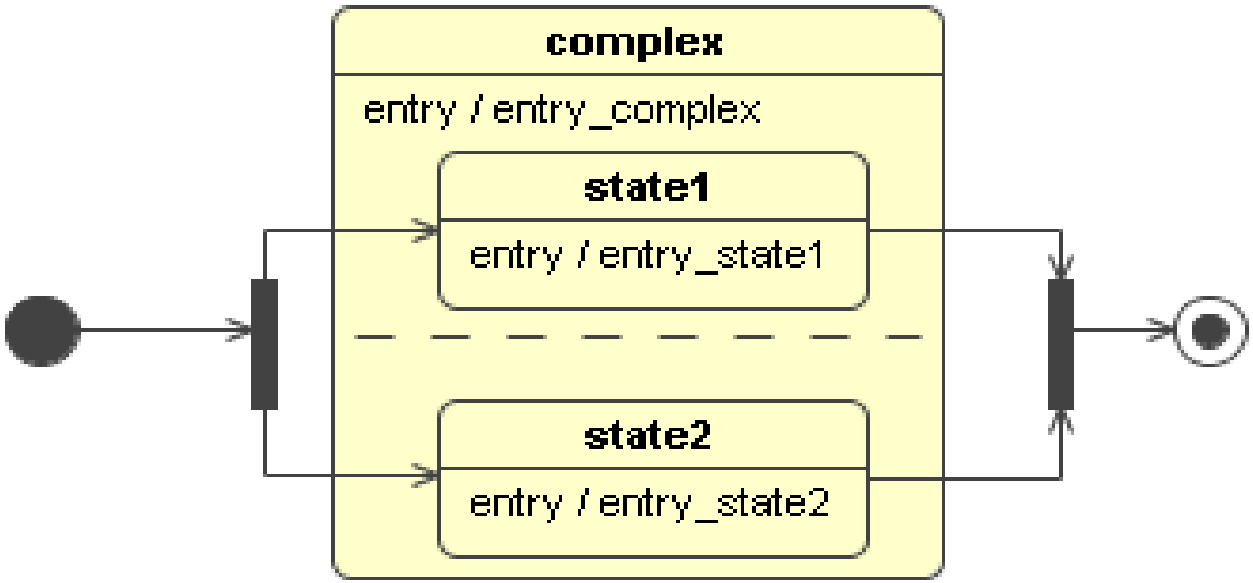
3. Was kann simuliert werden?

- Zusammengesetzte orthogonale Zustände:
  - Bei orthogonalen Regionen muss der Zugriff auf Variablen koordiniert bzw. synchronisiert werden
  - Zugriff nicht spezifiziert in der UML
  - Lösungen
    - Duplizieren der verwendeten Variable für jeden Zweig
    - Serialisieren des Zugriffs auf die Variablen
    - Vermeiden einer solche Situation



3. Was kann simuliert werden?

- Zusammengesetzte orthogonale Zustände:
  - Bei mehrfachem Eintritts- bzw. Austrittsverhalten muss eine Reihenfolge festgelegt werden
  - In der UML Spezifikation ist dabei keine konkrete Reihenfolge vorgegeben
  - Reihenfolge hängt von der Dauer des jeweiligen Verhaltens ab
  - Auswahl ob sequenziell oder parallel



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
  - Guard-Ausdrücke müssen rechnerlesbar sein
    - Z.B. in einer formalen Sprache

3. Was kann simuliert werden?

Transitionen (2)

- Das Verhalten einer Transition muss rechnerlesbar spezifiziert sein
  - Operation (in einer formalen Sprache)
  - Aktivitäten-Diagramm
  - Zustandsautomat
- Trigger:
  - CallTrigger: Verhalten der aufgerufenen Operation muss vom Rechner interpretierbar sein
  - TimeTrigger:
    - Benötigt eine (virtuelle) Zeit
    - Muss rechnerlesbar sein, z.B. wenn in einer formalen Sprache definiert (z.B. *if (time > 23:00)*)

3. Was kann simuliert werden?

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

5. Simulator

- Genereller Ablauf
- Unterstützte Elemente
- Einschränkungen
- Ansteuerung des Simulators

5. Simulator

- **Allgemeiner Ablauf:**
  - Der Simulator arbeitet rundenbasiert
  - Zu Beginn jeder Runde wird ein Zustand aus der Liste der aktiven Zustände entnommen
  - Die Liste der aktiven Zustände beinhaltet alle Zustände, in denen sich der Zustandsautomat gerade befindet (*activeStates*)
  - Wenn eine Transition geschaltet hat, wird der Zielzustand in die Liste der in der nächsten Runde aktiven Zustände aufgenommen (*nextActiveStates*)
  - 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

Anmerkung:

- Im Gegensatz zur UML Spezifikation ist der Aufenthalt in Pseudozuständen erlaubt
- Aufenthalt bedeutet in diesem Fall aber nicht, dass der Zustandsautomat sich dauerhaft in diesem Zustand aufhält, sondern maximal bis zur nächsten Runde
- Ausnahme: Vereinigung
  - Der Simulator hält sich in einer Vereinigung so lange auf, bis alle eingehenden Transitionen durchlaufen wurden

5.1. Simulationsarten

Schrittweise manuelle Simulation

- Erfolgt durch die parameterlose Methode *doStep()*
- Diese Methode führt eine oben beschriebene Runde der Simulation aus
- Vor dem Aufruf der Methode kann der Aufrufer neue aktive Ereignisse für die Trigger hinzufügen, Variablen auslesen und ändern, oder die bereits besuchten Zustände und durchlaufenen Transitionen untersuchen
- Die Kontrolle des Simulators liegt vollständig beim Aufrufer
- Kann beispielsweise dazu verwendet werden, um das Modell zu verifizieren und Fehlverhalten bereits vor dem nächsten Schritt festzustellen
- Weiterhin kann dieser Modus zur Überprüfung des Simulators auf Korrektheit und Vollständigkeit verwendet werden
- Nützlich vor allem bei neu implementierter Funktionalität.

5.1. Simulationsarten

Automatische Simulation

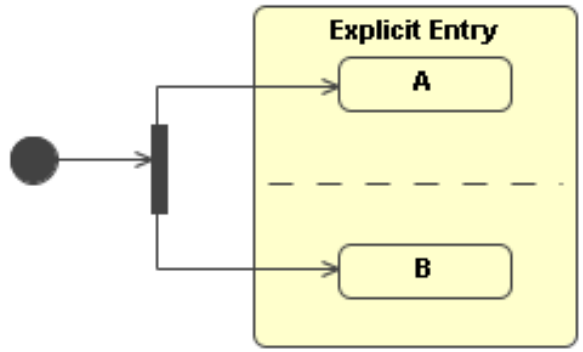
- Relevant für das UnITeD Projekt
- In diesem Modus wird das Modell solange simuliert, bis während der Simulation ein vorher festgelegtes Ereignis aufgetreten ist
- Dazu wird die parameterlose Methode *simulate()* verwendet, die bei der Rückkehr den Grund für das Ende der Simulation angibt
- Anhand dieses Rückgabewerts kann entschieden werden, ob die Simulation mit eventuell neuen Eingabedaten wieder aufgenommen, oder vollständig abgebrochen werden soll

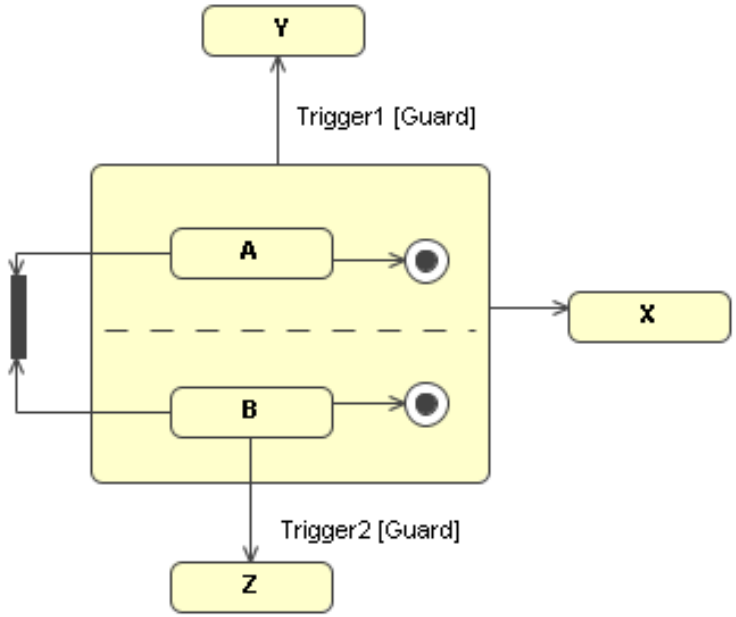
5.2. Unterstützte Elemente

- Einfacher Zustand
  - Eintritts-, Zustands und Austrittsverhalten modelliert durch eine Operation
- Start- und Endzustand
  - Auch Default Entry/Exit zusammengesetzter Zustände
- Terminator-Zustand
- Entscheidung
  - Als einfacher Zustand mit Guards
- Kreuzung (noch) nicht unterstützt

5.2. Unterstützte Elemente

- Zusammengesetzte Zustände
  - Mehrere Regionen
  - Betreten
    - Default Entry
    - Explicit Entry
    - Gabelung (bei orthogonalen Regionen)
  - Verlassen:
    - Erreichen der Endzustände
    - Trigger für den zusammengesetzten Zustand
    - Trigger für einen Unterzustand
    - Über eine Vereignung





5.2. Unterstützte Elemente

- Gabelung und Vereinigung
  - Sonderfall: Zusammengesetzte orthogonale Zustände
- Eintritts- und Austrittspunkt
  - Werden (noch) wie einfache Zustände vom Simulator behandelt
  - → Keine Connection Point References
- Unterzustandsautomatenzustand
  - Durch eine weitere Instanz des Simulators
- Historie
  - Flache Historie
  - Tiefe Historie

5.2. Unterstützte Elemente

- Transition
  - Trigger
    - CallTrigger, SignalTrigger und AnyTrigger
    - Unterschiedliche Verhalten der Ereignis-Warteschlange wenn benötigtes Ereignis nicht vorhanden
      - Ereignis am Kopf verwerfen
      - Ereignis am Kopf hinten wieder anhängen
      - Warteschlange nach Ereignis durchsuchen
  - Guard
    - Als bool‘scher Ausdruck, else Guard
- Operation
  - Verhalten modelliert durch Wertänderung von Variablen
  - Parameterübergabe

5.5. Einschränkungen

- 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
  - Disjunkte Guards und Trigger
- Nur bool‘sche Guard-Ausdrücke mit definierten Variablen
- Das Verhalten von Operationen muss durch einen oder mehrere Ausdrücke als Constraint erfolgen
- Die Wertzuweisung an Operationsparameter muss über das Mapping erfolgen

5.5. Einschränkungen

- Einschränkungen der Elemente (1)

Element	Einschränkung
Einfacher Zustand	- Eintritts-, Zustand- und Austrittsverhalten nur als Operation modellierbar - Zustandsverhalten nicht unterbrechbar
Transition	
Guard	Nur bool‘sche Ausdrücke mit definierten Variablen
Trigger	-Nur CallTrigger, SignalTrigger und AnyTrigger möglich -Keine Verzögerung durch /defer
Effekt	Nur durch eine Operation modellierbar
Entscheidung	Nur bool‘sche Ausdrücke mit definierten Variablen
Kreuzung	Noch nicht implementiert

5.5. Einschränkungen

- Einschränkungen der Elemente (2)

Element	Einschränkung
Zusammengesetzter orthogonaler Zustand	Keine echte Parallelität
Gabelung und Vereinigung	Keine echte Parallelität
Eintritts- Austrittspunkt	Werden noch nicht besonders behandelt → noch keine Connection Point References
Unterzustandsautomat	Keine Connection Point References
Spezialisierung	Nicht implementiert

5.6. Ansteuerung des Simulators

- Vorbereiten des Models: *getSimulateableStateMachine(Model, StateMachine)*
- Initialisierung des Simulators: *simulator.setupSimulation()*
- Simulation manuell schrittweise: *StopReason simulator.doStep()*
- Simulation automatisch: *StopReason simulator.simulate()*
- Einstellungen: *setEventQueueBehavior (EventsQueueBehavior), setMultipleConcurrentBehavior(MultipleConcurrentBeavior*
- Ereignisse hinzuzufügen und löschen: *addEvent(), addEvents(), removeEvent()*
- Variablen setzen und auszulesen: *setVariable(), getVariable()*
- Besuchte Zustände und durchlaufenen Transitionen abfragen *getVisitedVertices(), getPassedTransitions()*
- Parameter der Operationen setzen: *setCallEventParameter()*

6. Ausblick

-Übertragung auf andere Diagramme

-Verbesserungen und Erweiterungen

6.1. Übertragung

- Bsp. Aktivitätsdiagramm (1)
  - Wieder verwendbare Teile
    - Behandlung der Zustände und Transitionen einfach für Aktionen und Kanten anpassbar
    - Der Expression Parser kann für die Auswertung von Bedingungen wieder verwendet werden
    - Entscheidung als Verzweigungsknoten verwendbar
    - Gabelung und Vereinigung grundsätzlich als Parallelisierungs- und Synchronisationsknoten verwendbar, aber einige Anpassungen nötig
    - Behandlung zusammengesetzter Zustände nach geringfügigen Änderungen verwendbar für Unterbrechungsbereich und strukturierter Knoten
    - Ereigniswarteschlange für das Senden und Empfangen der Signale

6.1. Übertragung

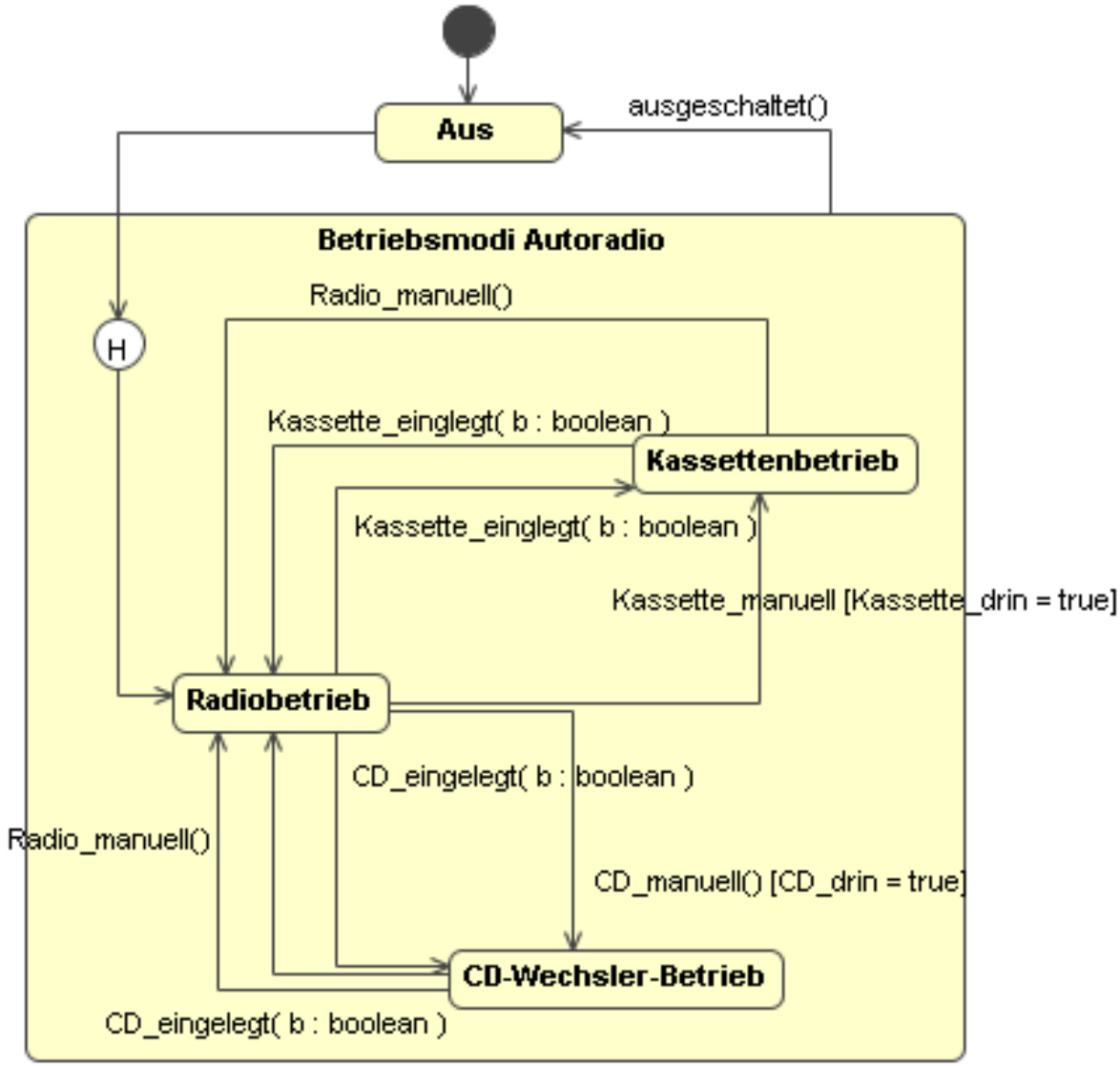
- Bsp. Aktivitätsdiagramm (2)
  - Nötige Erweiterungen
    - Behandlung von Objektknoten und Objektfluss
      - Eingabe- und Ausgabeparameter
      - Parametersätze
    - Unterstützung für gewichtete Kanten und Knoten (Tokenkonzept)
    - Sprungmarken und Verbindungsknoten
    - Schleifen- und Entscheidungsknoten
      - Auswertung der Bedingungen kann aber durch den Expression Parser erfolgen

6.2. Verbesserungen und Erweiterungen

- Effizienz
  - Austausch der vielen Listen und Hashtabellen durch performantere Datenstrukturen
- Fehlende Elemente
  - TimeTrigger: Einführen einer virtuellen Zeit (z.B. die Rundenanzahl), ähnlich wie bei der DES (Diskreten Event Simulation)
  - ChangeTrigger: Erkennbar z.B. durch Überwachung der Laufzeitumgebung (verwendet für das Lesen und Speichern der Variablen)
  - Eintritts- und Austrittspunkt sowie Connection Point References
  - Spezialisierung
- Threads für echte parallele Ausführung
- Implementierung der Deadlockerkennung
  - Erkennung von Starvation bereits implementiert

6. Demonstration

6. Demonstration



Frohe Weihnachten und ein
gesundes und erfolgreiches Jahr 2007

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
- [SFM] http://tech-www.informatik.uni-hamburg.de/applets/java-fsm/, Stand 19.12.2006