ModSoft

Modellbasierte Software-Entwicklung mit UML 2
im WS 2014/15

Teil III: Zustandsautomaten

Prof. Dr. Joachim Fischer
Dr. Markus Scheidgen
Dipl.-Inf. Andreas Blunk

fischer@informatik.hu-berlin.de

ModSoft (UML): Inhalt

Teil I- Einführung

Teil II-Struktur

a)
- Klassen, Assoziationen, Abhängigkeiten
- Interface, Datentypen, Signale, Port,
- Strukturierter Classifier
- Aktive Klassen

b)
- Präzisierungen
- Klassendiagramm, vertiefende Betrachtung
- Operationen

Teil III-Verhalten

a)
- Einfacher Zustandsautomat
- Beispiel: DemonGame
- UML-Modellierungsdefizite

b)
- Aktivitäten

c)
- Erweiterte Zustandsautomaten

Teil IV- OCL

Zustandsautomaten

1. Einführung
2. Einfache Zustandsautomaten
3. Trigger-Arten
4. Semantik der Ereignisbehandlung
5. Allgemeine Charakterisierung (Wdh.)
6. Zusammengesetzte Zustände
7. Run-to-Completion-Semantik-Präzisierung
8. Pseudozustände, Unterautomaten, History-Zustände
9. Zustandstypen, Vererbung
10. Protokollautomaten

UML-Zustandsdiagramme

- basieren auf Automatenkonzept von Harel (Moore, Mealy)
- Zwei Varianten in UML
  - Verhaltensautomat (beschreibt Verhalten einzelner Objekte)
    - Zustand eines Objektes (Attributbelegung)
    - Zustand eines Automaten (grobkörniger: ~ Situation, in der er auf bestimmte Art und Weise einheitlich reagiert
  - Protokollautomat (Spezifikation zur Benutzung von Systemteilen)

UML-Ereignisse, die Zustandsübergänge triggern

1. **Signal** (-Ankunft) mit optionaler Wächterbedingung
2. **Op-Call-Request** mit optionaler Wächterbedingung

   Es muss unterscheidbar sein, ob der Call von synchroner oder asynchroner Art ist

3. **Zeitereignis** (absolute, relative Zeitangabe)

   Zeiteinheit ist nicht vorgegeben

4. **Zustandsereignis** (mit Bedingung)
5. **implizites Ereignis** (Beendigung einer Aktivität) mit optionaler Wächterbedingung

   Completion ist möglicherweise einziges implizites Ereignis

- Signal-Parameter werden als Attribute eines Signal-Classifiers repräsentiert
- Senden von Signalen/Op-Call-Requests sind Vorgänge, an denen mindestens zwei Objekte (Sender, Empfänger) beteiligt sind

  Sonderfälle:
  - Broadcast (Empfangsobjekt ist ein Aggregationsobjekt (System))
  - Multicast (Empfänger ist eine Kollektion von Objekten)

Synchrone – Asynchrone Operationsrufe

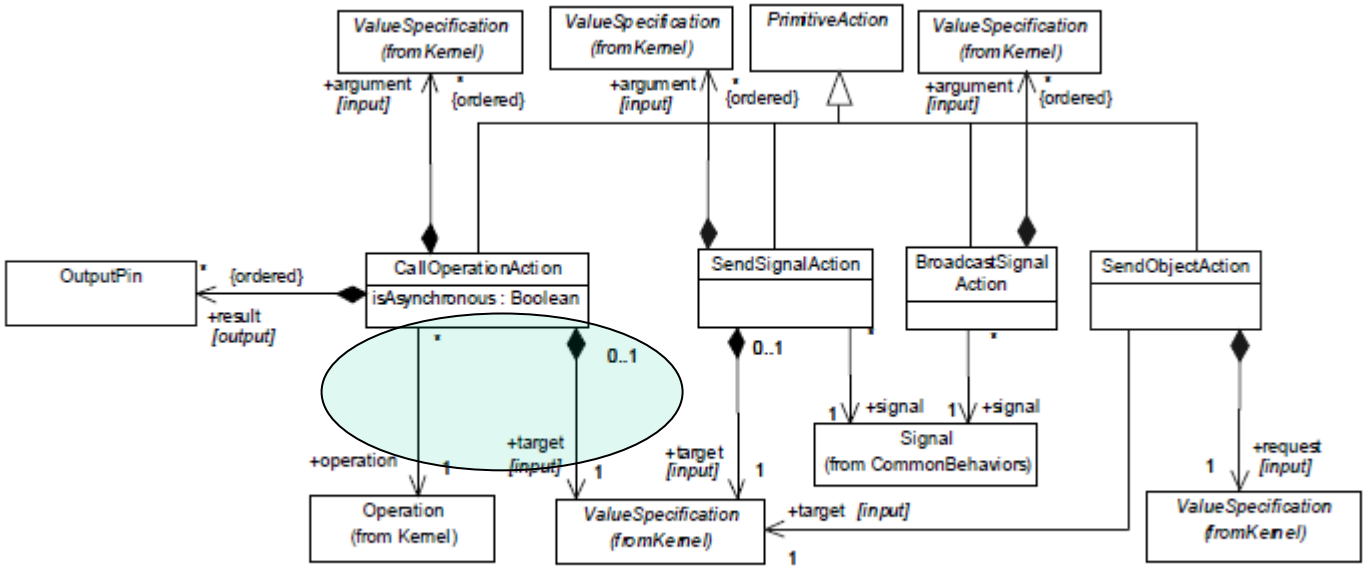
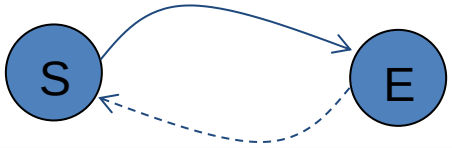


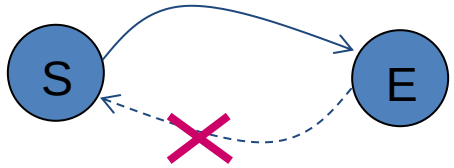
Figure 4-60. The messaging actions.

synchron
(pseudosynchron)



S blockiert solange bis zum Reply
(d.h. Op-Aufruf in E ist abgeschlossen und Resultat wurde übermittelt)
oder Exception von E liegt vor oder Timeout

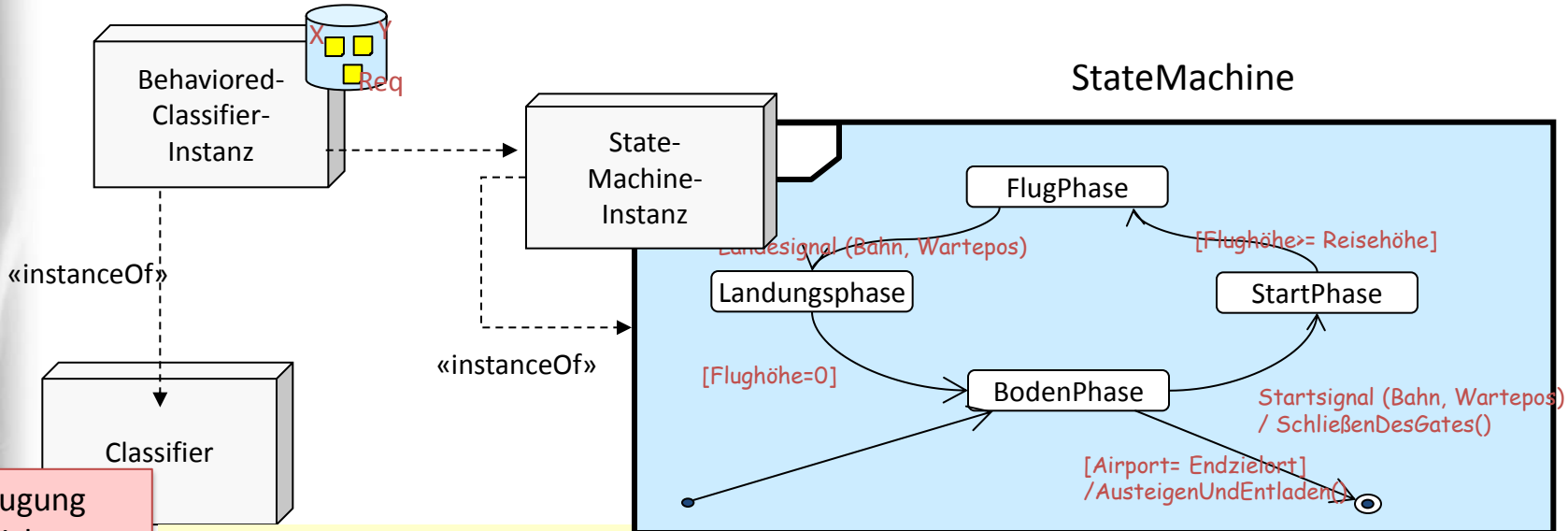
asynchron



S setzt Arbeit nach callOperationAction fort
Reply wird von E nicht ausgeführt

Semantik der Ereignisbehandlung

Ereignispool (Empfangsereignisse (ankommende Signale, eingehende Call-Requests), Zustandsereignisse, Zeitereignisse, Beendigungsereignisse)



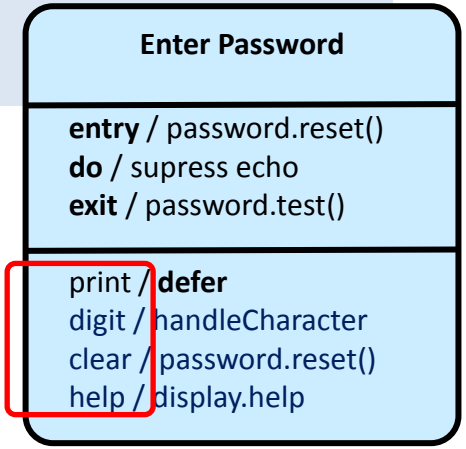
Objekterzeugung bedeutet nicht automatisch Zustandsautomat-Aktivierung

OFFEN:
Wie können beide Aktionen verbunden werden?

Run-to-completion

- die Zustandsmaschine bearbeitet (zu einem beliebigen Zeitpunkt) kein oder genau ein Ereignis
- es wird immer erst dann ein weiteres Ereignis aus dem Pool bearbeitet, wenn das vorhergehende Ereignis **vollständig** abgearbeitet wurde
- Reihenfolge der Abarbeitung von Ereignissen ist in UML **nicht** bestimmt (aus der Menge feuerbarer Transitionen wird nichtdeterminiert gewählt)
  → erlaubt den Einsatz verschiedener Priorisierungsverfahren bei UML-Profilen

UML-Zustandsdefinition



Trigger-Signale

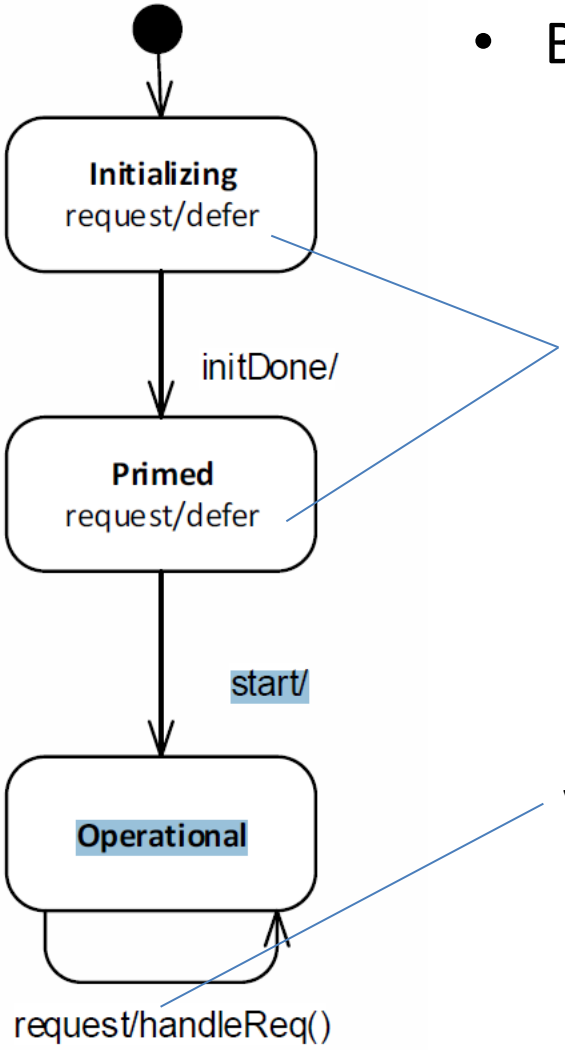
Bestandteile/Charakteristik

- Name
  (optional), kein Name, dann anonymer Zustand
- Entry-/ Exit- Aktionen bei Eintritt bzw. Austritt
- Do-Aktivität
  durch externes Ereignis unterbrechbar,
  d.h. per Signal-, Call-, Time-, State-Event

ohne Parameter

- interne Transitionen (ohne Wechsel des Zustandes)
  durch externes Ereignis unterbrechbar, d.h. per Signal-, Call-, Time-, State-Event
  Achtung: semantischen Unterschied zu externen reflexiven Transitionen klarmachen
- Verzögerung (defer) von Ereignissen
  spezielle interne Transition
  Liste von Signal- und Call-Request-Events,
  die in diesem Zustand nicht behandelt werden aber
  für die möglichen direkten Folgezustände nicht verworfen, sondern nur zurückgestellt werden (= SDL-save-Trigger)

Defer (Wdh.)



- Beispiel aus dem UML-Standard

Zurückstellung des Signals request

Verarbeitung des Signals request

do-Aktivität (Wdh.)

- ein Zustand kann (höchstens) **eine** interne do-Aktivität enthalten
- wird der Zustand betreten, beginnt die do-Aktivität parallel mit der Entry-Aktivität
- die do-Aktivität kann dabei endlich oder unendlich sein (Zyklus)
  - terminiert die (endliche) do-Aktivität,
    so komplettiert sich der Zustand,
    wobei eine Komplettierungstransition zum Verlassen des Zustandes führt (ausgelöst durch ein implizites Komplettierungsereignis)
  - eine unendliche oder endliche do-Aktiviät kann aber auch durch ein explizites Ereignis (z.B. Zustandsereignis) beendet werden
    (**Problem**: Unterbrechung einer Aktionsfolge)

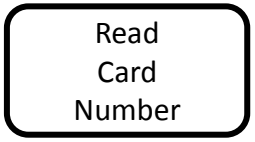
Zustandsautomaten

1. Einführung
2. Einfache Zustandsautomaten
3. Trigger-Arten
4. Semantik der Ereignisbehandlung
5. Allgemeine Charakterisierung (Wdh.)
6. Zusammengesetzte Zustände
7. Run-to-Completion-Semantik-Präzisierung
8. Pseudozustände, Unterautomaten, History-Zustände
9. Zustandstypen, Vererbung
10. Protokollautomaten

Zustand und Zustandskompositionen

Komposition gilt sowohl für Zustände als auch für Automaten (**später**)

Zustandsarten

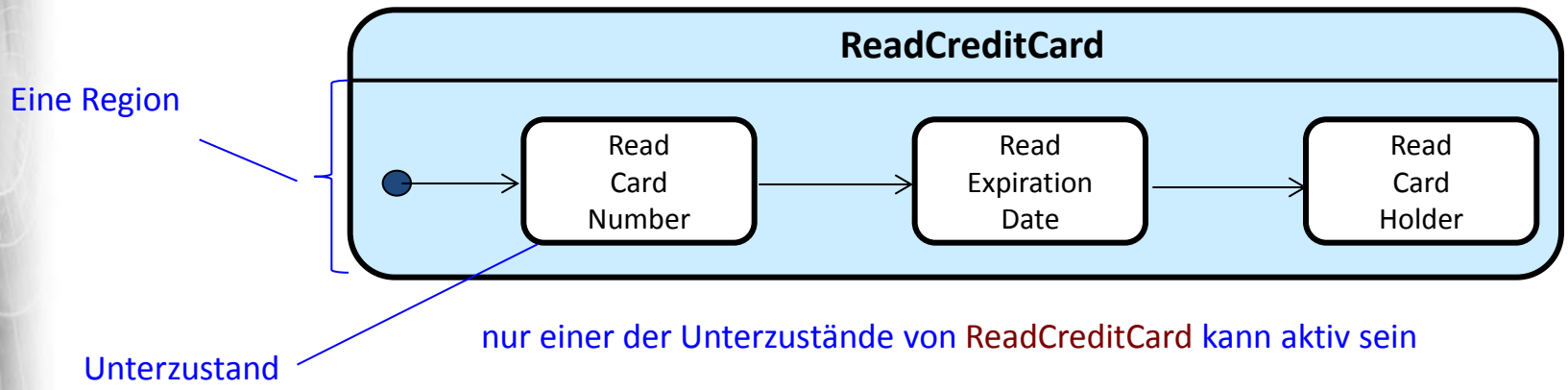


(1) einfacher Zustand (simple State)

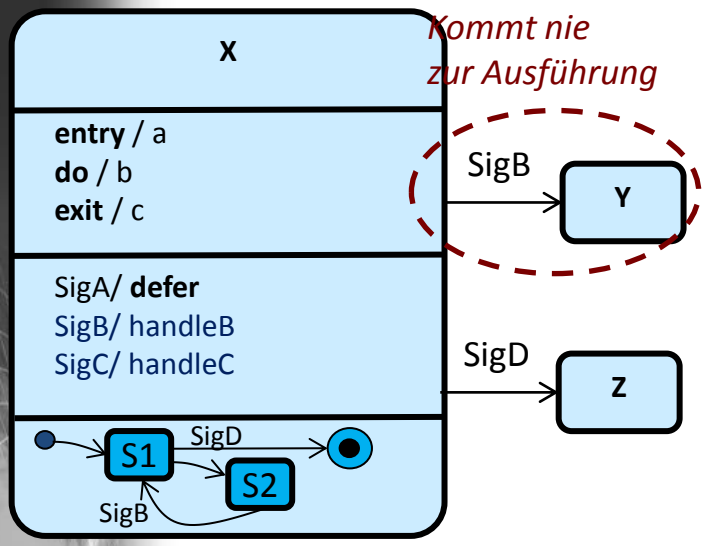
- Zustand ohne Unterzustände
- können zu Zustandskompositionen (composite States) zusammengesetzt werden

(2) zusammengesetzter Zustand (composite State)

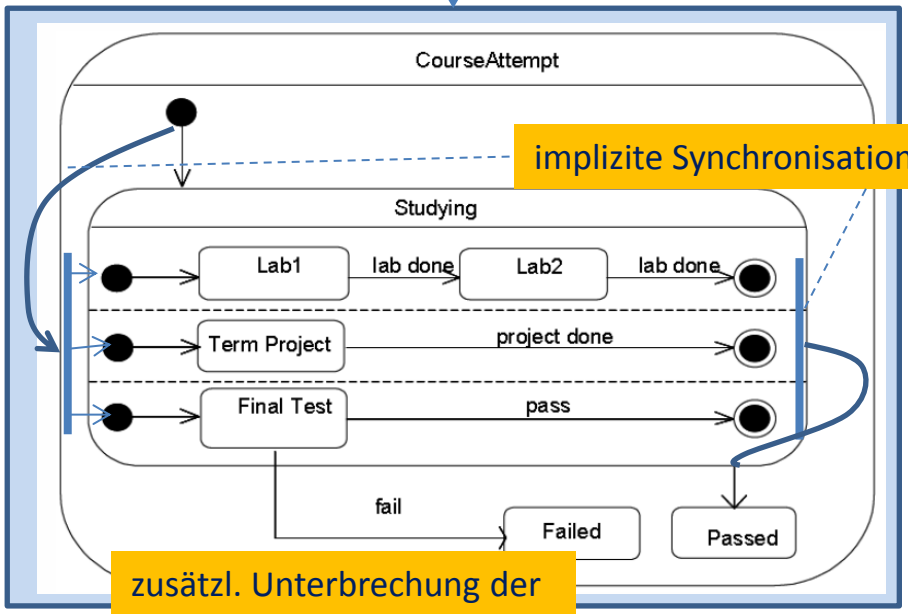
- Zustand, der **mindestens eine** sog. **Region** für **Unterzustände** beinhaltet
- mehrere Regionen: parallele Ausführungszweige



UML-Zustandsdefinition



- Substates/Unterzustände: verschachtelte Struktur
  - ODER-verfeinerter Zustand (sequentielle Verschachtelung)
  - ODER- plus UND-verfeinerter Zustand (parallele Verschachtelung)



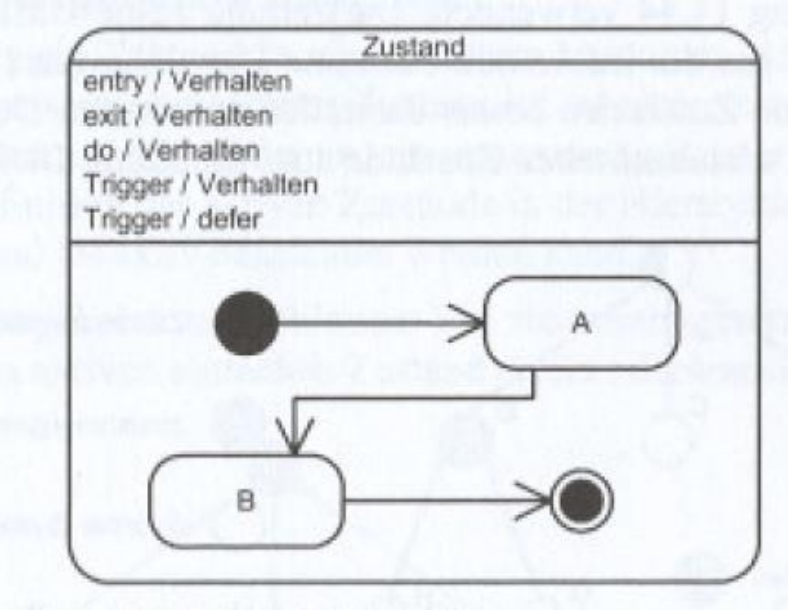
Maskierung äußerer Transitionen durch innere Transitionen (bei identischen Ereignissen)
Hier: SigB, SigD

SigB:
X/S1 → X/S1 und handleB
X/S2 → X/S1
X → X

SigD:
X/S1 → X
X/S2 →Z und c
X → Z und c

Zusammengesetzter Zustand

zusammengesetzter Zustand
mit angegebener Komposition



zusammengesetzter Zustand
mit versteckter Komposition

Baum-Hierarchie bei Zustandskomposition

- Blätter sind einfache Zustände
- innere Zustände sind immer zusammengesetzte Zustände
- Wurzel in einem vollständig definierten Modell ist Zustandsautomat

Ausführungsreihenfolge (Zustandsübergang)

Reihenfolge ist kanonisch (folgt der Richtung der Transition):

1. alle exit-Aktionen des aktuellen Zustandes

   wurde Transition durch Call-Event ausgelöst, wird die entsprechende Methode ausgeführt
   (Magie! **denn** synchrone und asynchrone Rufe haben unterschiedliche Effekte und Exception-Handling bleibt offen)

2. Ausführung der Transitionsaktionen
   (Aktionen, die zur Transition gehören)
   Reihenfolge durch ihre Anordnung im Text bestimmt

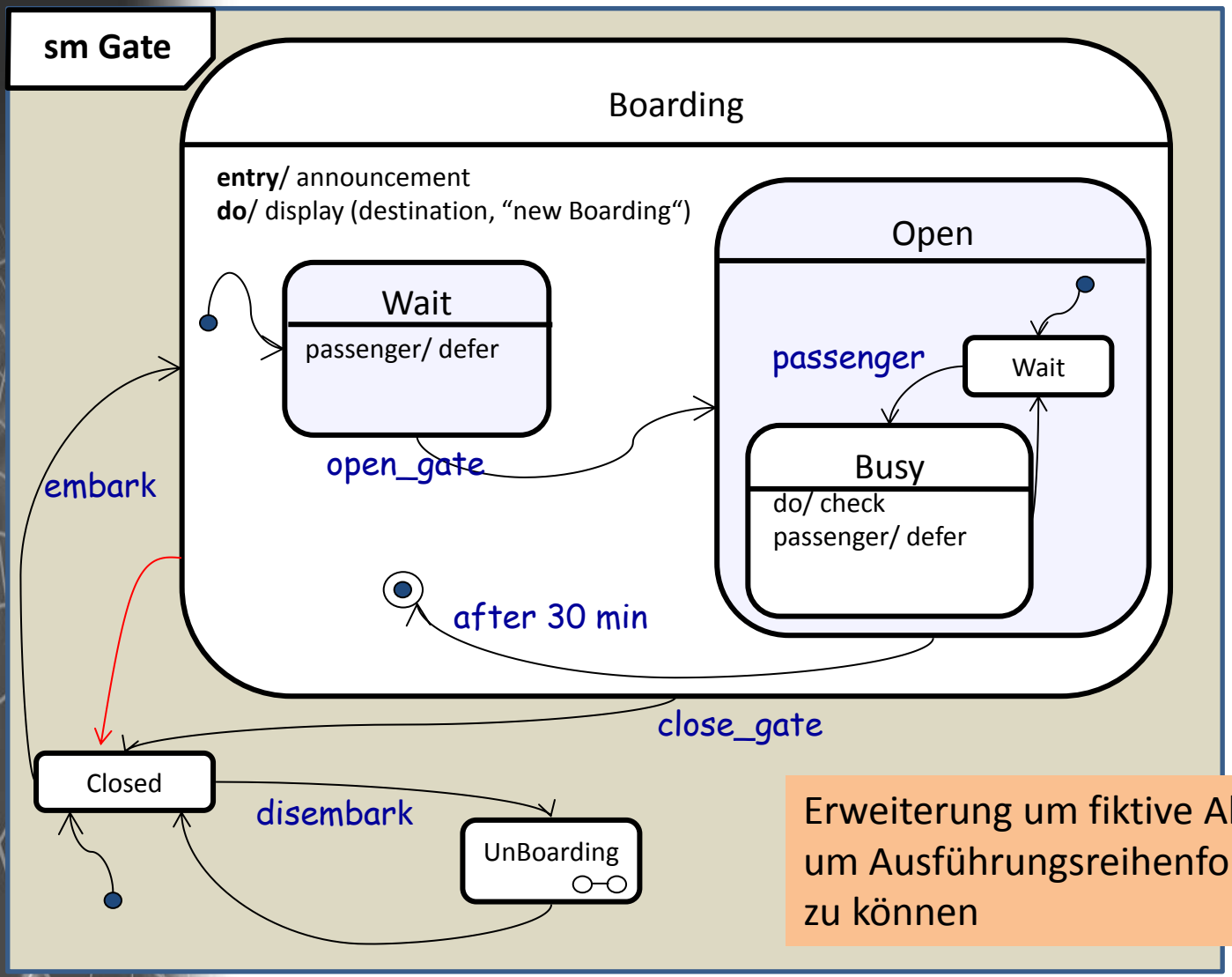
3. Abarbeitung der entry-Aktionen des Zielzustandes

4. Start der do-Aktivität

UML-Standard:
3 und 4 laufen parallel

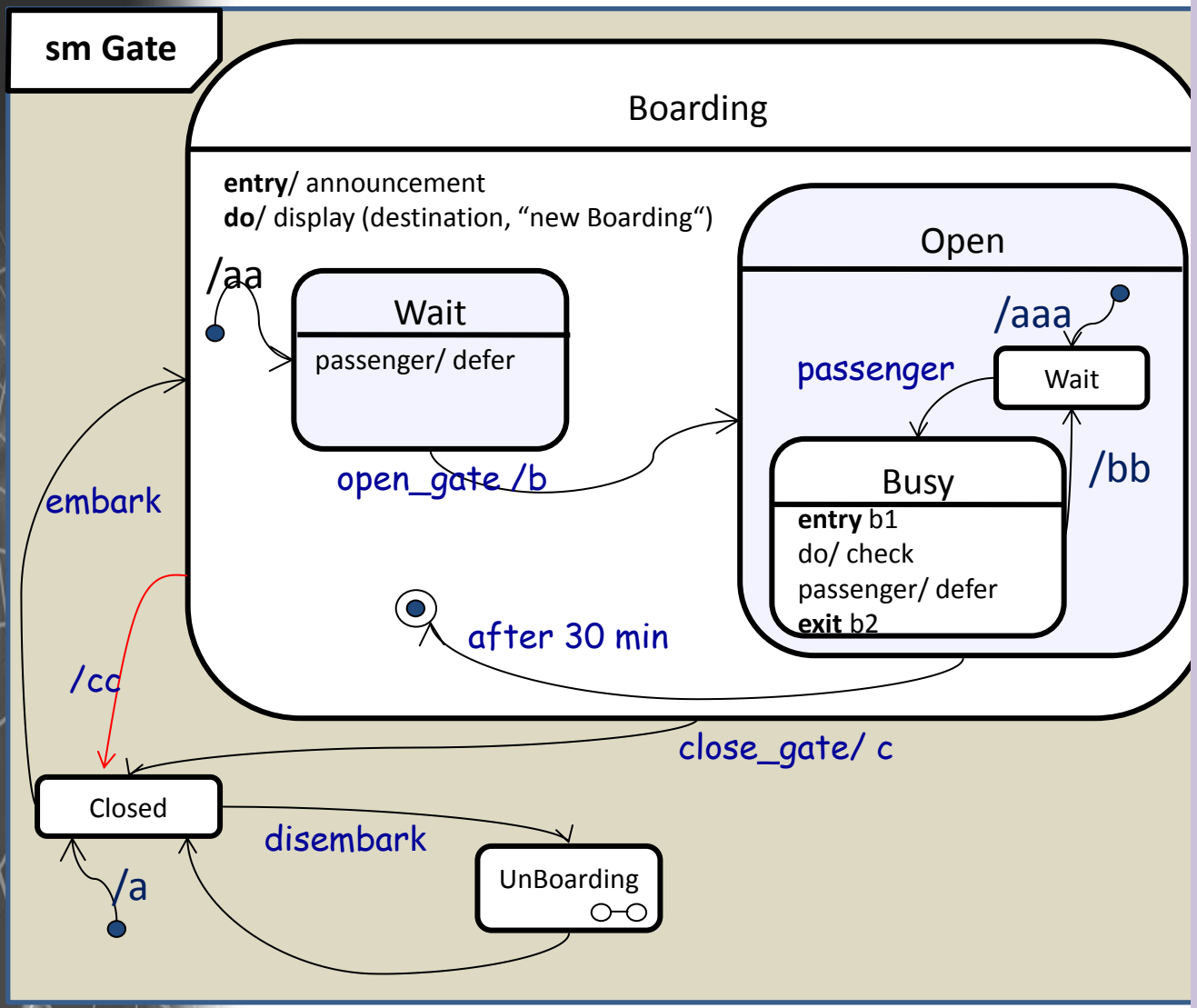
noch zu präzisieren für Zustände mit:
- internen Aktionen
- Unterzuständen
- Regionen

Beispiel-1: Automat mit verschachtelten Zuständen



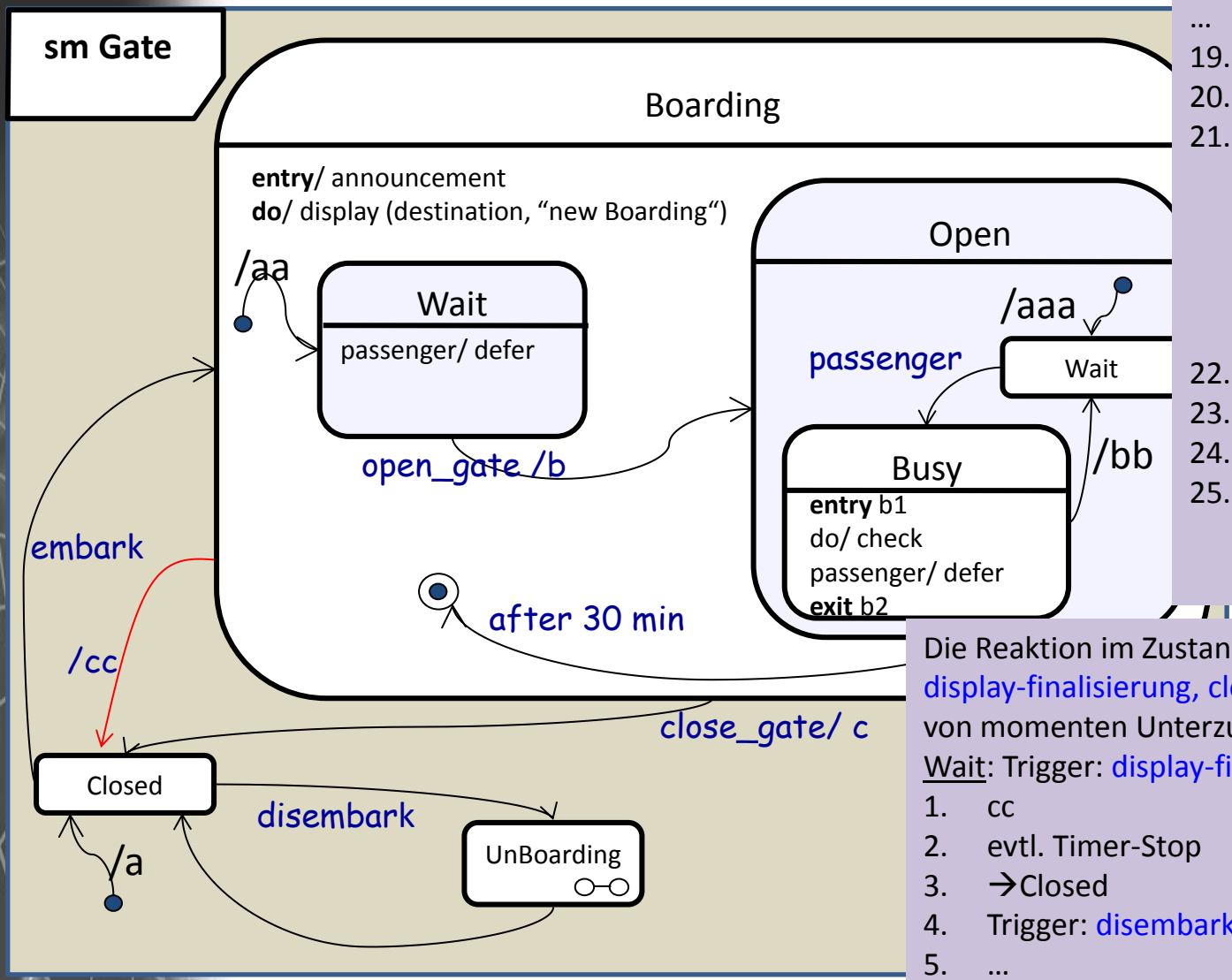
Erweiterung um fiktive Aktionen
um Ausführungsreihenfolgen besser diskutieren zu können

Beispiel-1: Automat mit verschachtelten Zuständen



1. a
2. →Closed
3. Trigger: embark,disembark
   alle anderen werden verworfen
4. announcement
5. → Boarding
6. display … parallel
7. aa
8. → Boarding/Wait
9. Trigger: passenger werden gerette
   auf open_gate, close_gate und
   display-finalisierung wird reagiert,
   alle anderen werden verworfen
10. Trigger: open_gate
11. b
12. → Boarding/Open
13. Start eines Timers
14. aaa
15. →Boarding/Open/Wait
16. auf passenger, close_gate und
    display-finalisierung wird
    reagiert, andere werden verworfe
17. Trigger passenger
18. b1
19. →Boarding/Open/Busy
20. check

Beispiel-1: Automat mit verschachtelten Zuständen



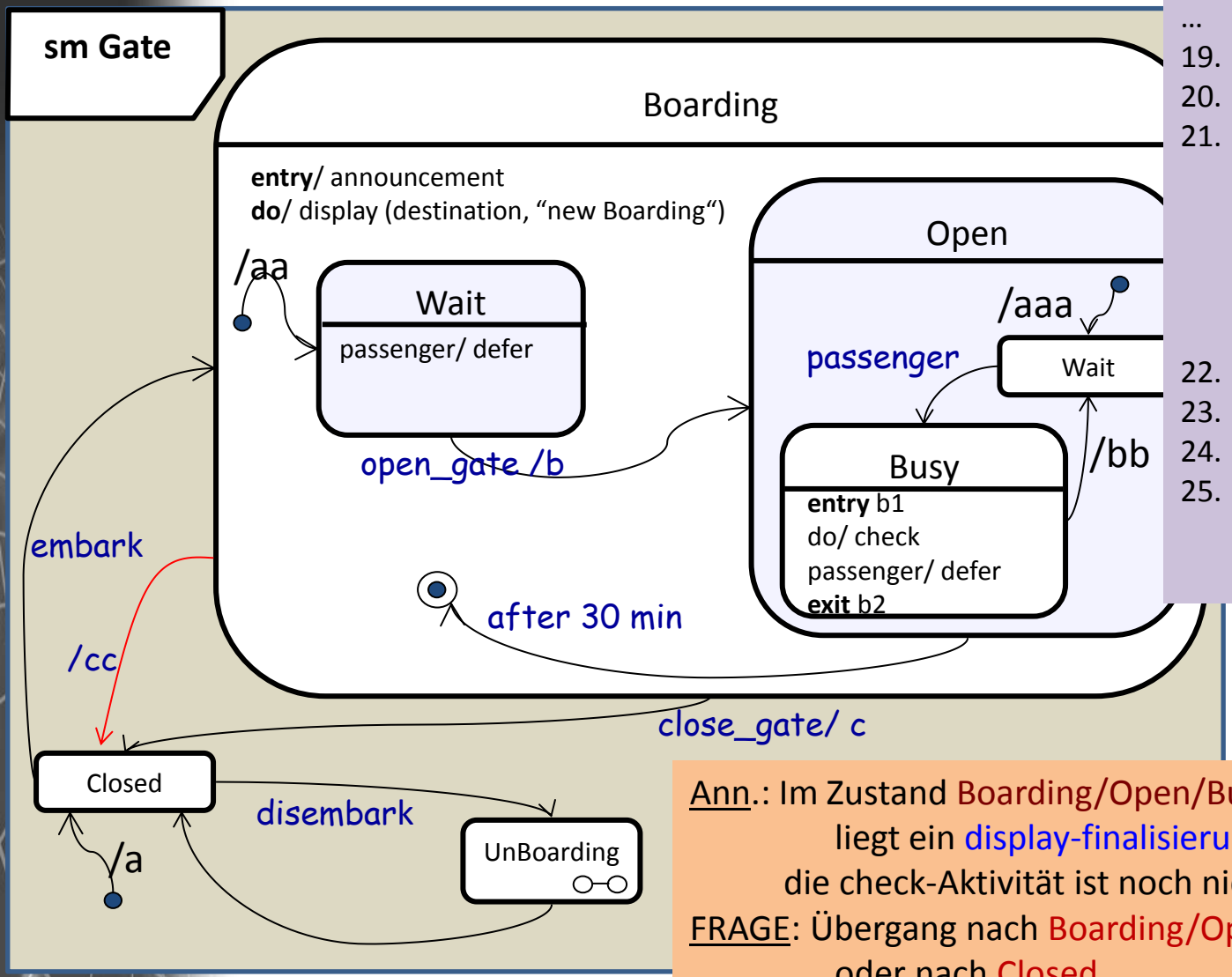
...
19. →Boarding/Open/Busy
20. check
21. auf close_gate, display-finalisierung, check-finalisierung und timeout wird reagiert, passenger wird gerettet, andere werden verworfen
22. Trigger: check-Finalisierung
23. b2
24. bb
25. →Boarding/Open/Wait (→15.)

Die Reaktion im Zustand Boarding/Open bei Trigger display-finalisierung, close_gate und timeout hängt von momenten Unterzustand ab:

Wait: Trigger: display-finalisierung oder timeout
1. cc
2. evtl. Timer-Stop
3. →Closed
4. Trigger: disembark
5. ...

bei Trigger: close_gate
1. Timer-Stop
2. c
3. → Closed
4. Trigger: disembark

Beispiel-1: Automat mit verschachtelten Zuständen

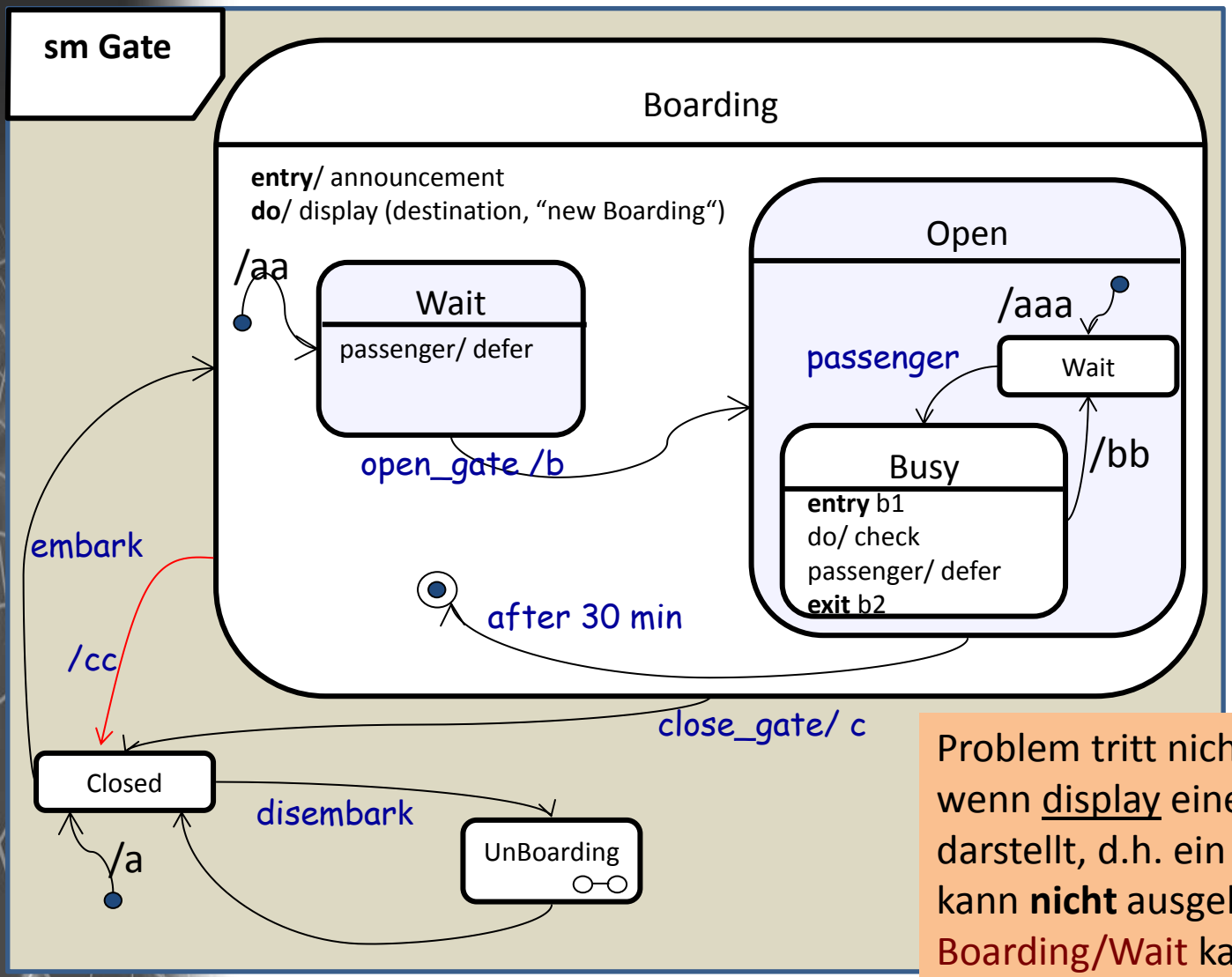


...
19. →Boarding/Open/Busy
20. check
21. auf close_gate, display-finalisierung, check-finalisierung und timeout wird reagiert, passenger wird gerettet, andere werden verworfen
22. Trigger: check-Finalisierung
23. b2
24. bb
25. →Boarding/Open/Wait (→15.)

problematisch?

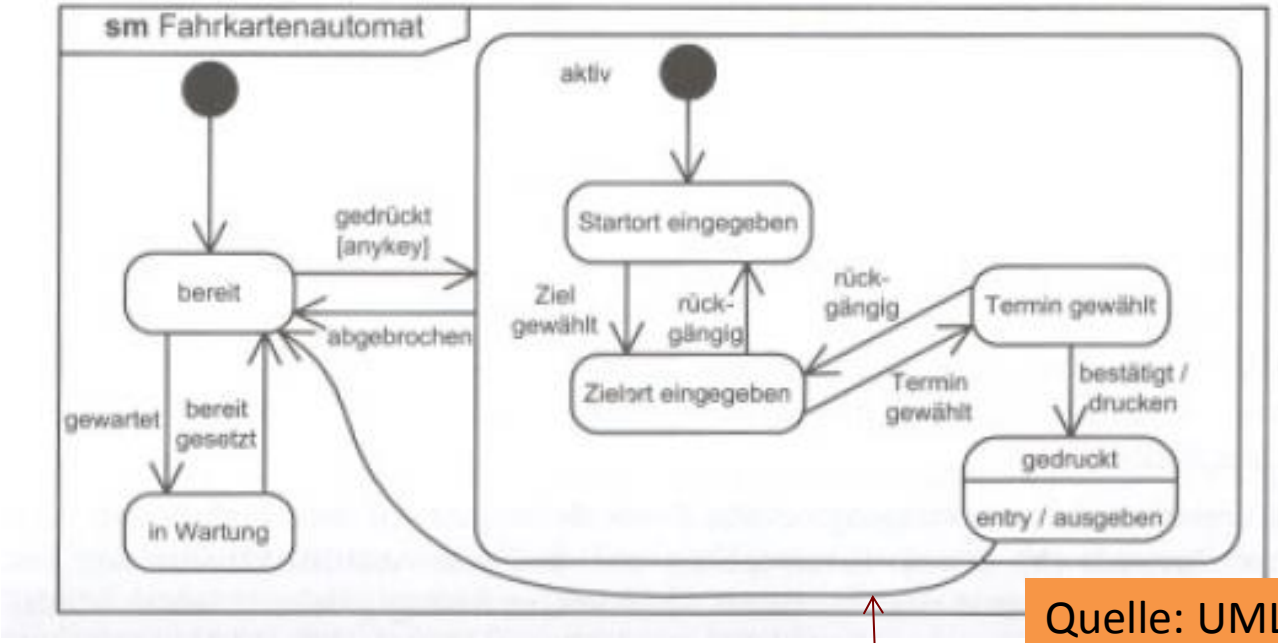
Ann.: Im Zustand Boarding/Open/Busy
liegt ein display-finalisierungs-ereignis an,
die check-Aktivität ist noch nicht beendet
FRAGE: Übergang nach Boarding/Open/Wait
oder nach Closed
bei Unterbrechnung von display ?

Beispiel-1: Automat mit verschachtelten Zuständen



Problem tritt nicht auf,
wenn display eine Daueraktivität darstellt, d.h. ein Finalisierungsereignis kann **nicht** ausgelöst werden
Boarding/Wait kann nur mit open_gate oder close_gate unterbrochen werden

Beispiel-2: Fahrkartenautomat



Quelle: UML-Glasklar

- Drücken einer Taste: bereit → aktiv
- Standort, Zielort, Termin werden bestimmt
- Bestätigungsknopf: termin gewählt → gedruckt
- implizites Beendigungsereignis (impliziter Trigger) → bereit

Wann/wodurch entstehen Komplettierungsereignisse?

Komplettierungsereignisse

Fälle

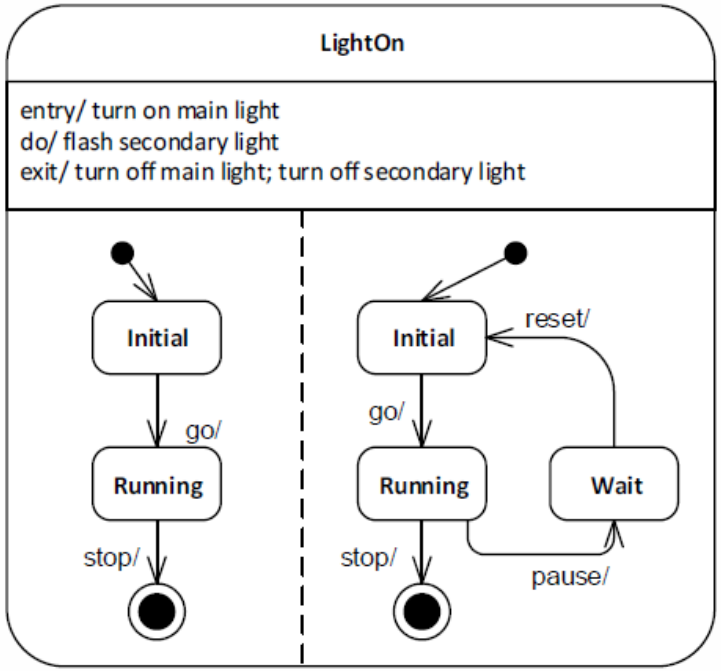
a) Zustand hat keine **entry-** und keine **do-Aktivität**
➔ Komplettierungsereignis wird mit Eintritt in den Zustand ausgelöst

b) Zustand verfügt sowohl über eine **entry**- als auch eine **do-Aktivität**
➔ Komplettierungsereignis wird nach Beendigung von Entry- und Do-Aktivität ausgelöst

c) Zustand verfügt über eine entry- oder do-Aktivität
➔ Komplettierungsereignis wird nach Beendigung dieser Aktivität ausgelöst

Beispiel: Automat mit orthogonalen Regionen

Quelle: UML-Standard



Semantik
ist sehr ungewöhnlich

SDL fordert hier auch
eine Orthogonalität
der zulässigen
regionalen Ereignismengen

Due to the presence of orthogonal Regions, it is possible that multiple Transitions (in different Regions) can be triggered by the same Event occurrence.

The order in which these Transitions are executed is left undefined.
… When all orthogonal Regions have finished executing the Transition, the current Event occurrence is fully consumed, and the run-to-completion step is completed.

Orthogonale Regionen versus Aktive Objekte

alternative Modellierungsmöglichkeiten von Nebenläufigkeiten

a) Zustandsautomat eines aktiven Objektes in zwei oder mehreren nebenläufigen Regionen

b) mehrere aktive Objekte, deren individuelle Zustandsverhalten genau einer Region entsprechen

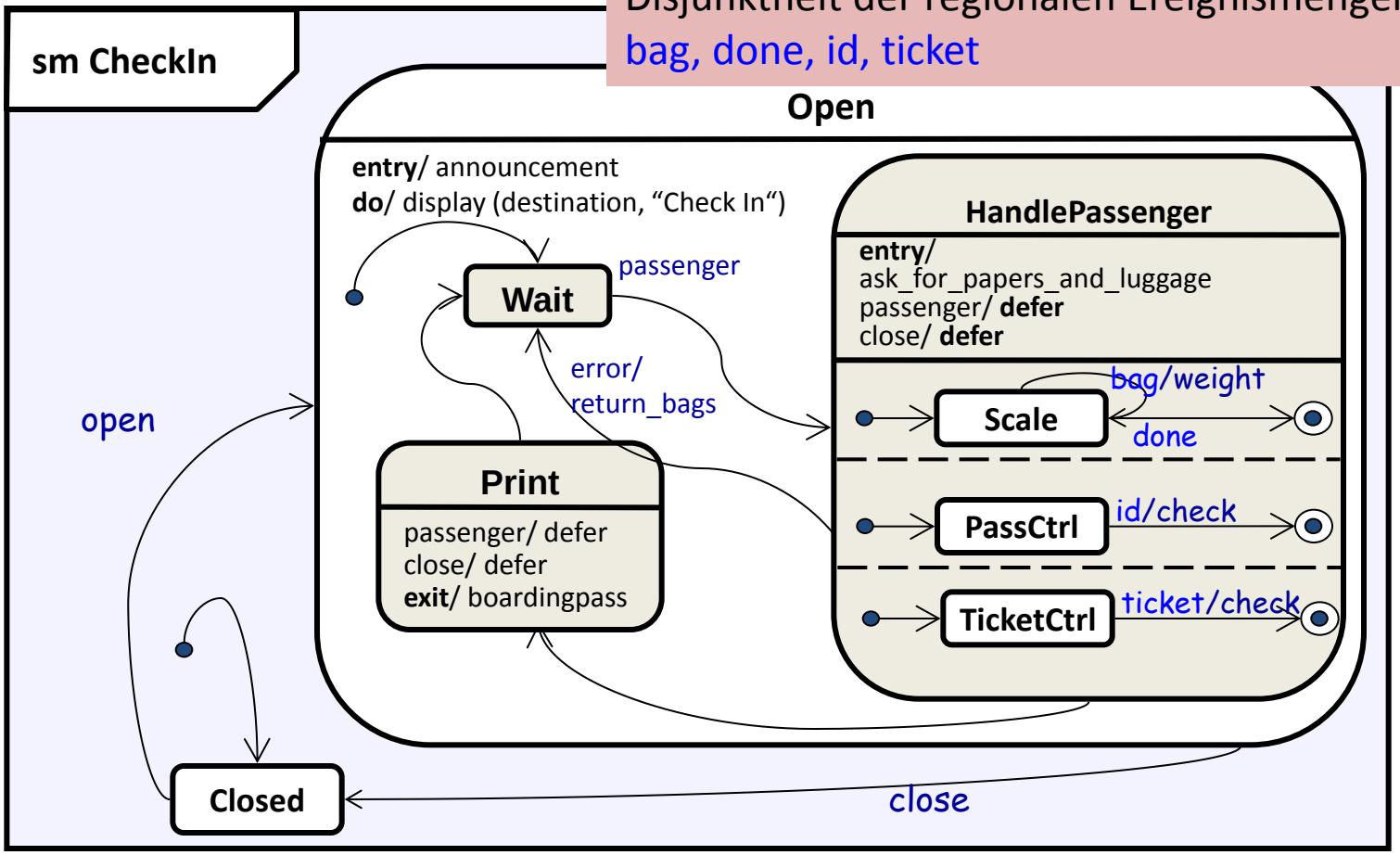
Wann, welche Methode?

- wird Verhalten eines der nebenläufigen Flüsse vom Zustand eines anderen Flusses beeinflusst → Region-Dekomposion

- Ist eine Kommunikation der Flüsse untereinander notwendig (z.B. Nachrichtenaustausch → mehrere aktive Objekte

Beispiel: Check-In

Disjunktheit der regionalen Ereignismengen
bag, done, id, ticket



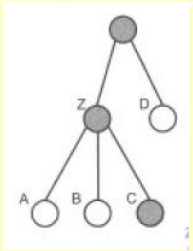
- Reihenfolge der Passagierabfertigung ist beliebig
- done-Ereignis: sobald alle Gepäckstücke gewogen sind
- Abfertigung kann nicht durch neuen Passagier *passenger* oder *close* beendet werden
- bei einem Fehler in einem der Abfertigungsschritte wird Abfertigung abgebrochen und Gepäck zurückgegeben

Präzisierung: Zustandskonfiguration

aktiver Zustand: Zustandskonfiguration
in dem sich ein Objekt zu einem Zeitpunkt befindet

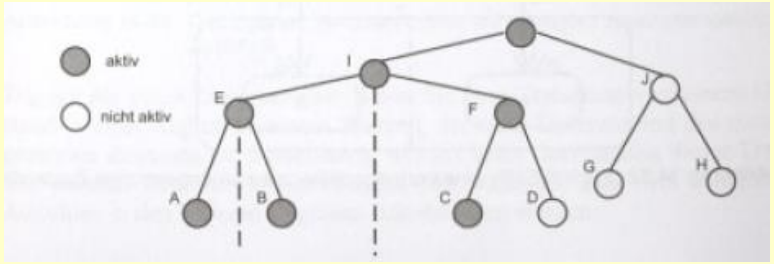
nicht-orthogonale Zerlegung

- zwei (oder mehr) Zustände auf der gleichen Ebene, dann ist das Objekt genau in einem der Zustände



orthogonale Zerlegung

- zwei (oder mehr) orthogonale Bereiche, dann ist das Objekt genau in einem Zustand, aber resultierend aus allen Bereichen



- in jeder Region muss sich ein aktiver Zustand befinden
- der Übergang von einer stabilen aktiven Zustandskonfiguration zur nächsten ist jeweils ein Schritt
- Transitionen zwischen Zuständen unterschiedlicher Regionen sind nicht erlaubt

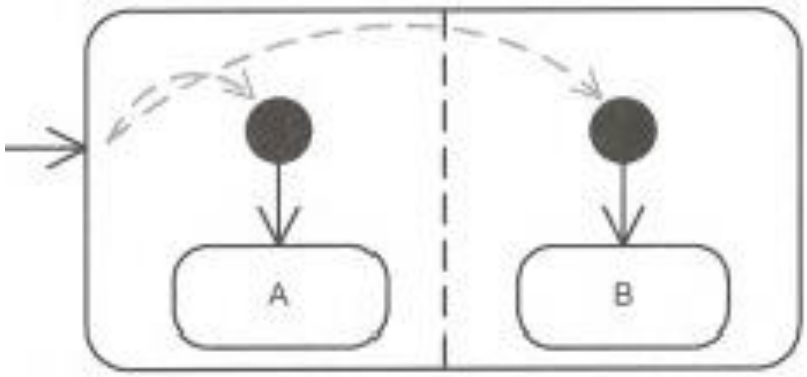
Run-to-Completion
verlangt ein zwischenzeitliches Verweilen der zuerst fertigen Regionen in ihren neuen Zuständen (bzw. alten Zuständen) bis all die anderen Regionen sich ebenfalls in einem aktiven Zustand befinden

Anpassung von Semantik-Regeln (1)

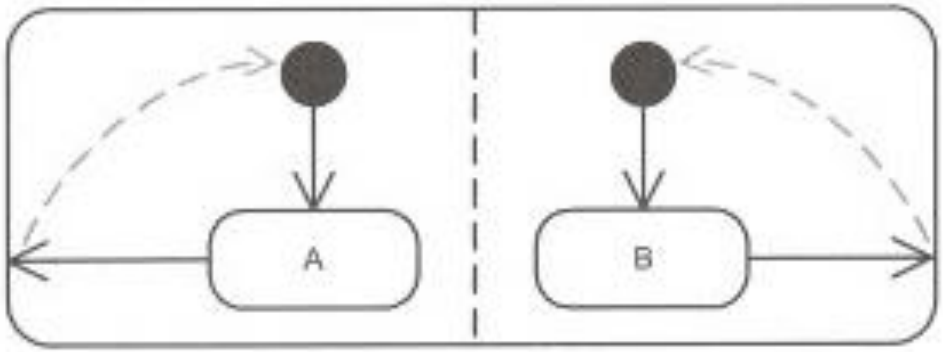
zwei Möglichkeiten des Betretens (alle Regionen aber immer gleichzeitig)
- Default Entry
- Explicit Entry

Default Entry

a)



b)

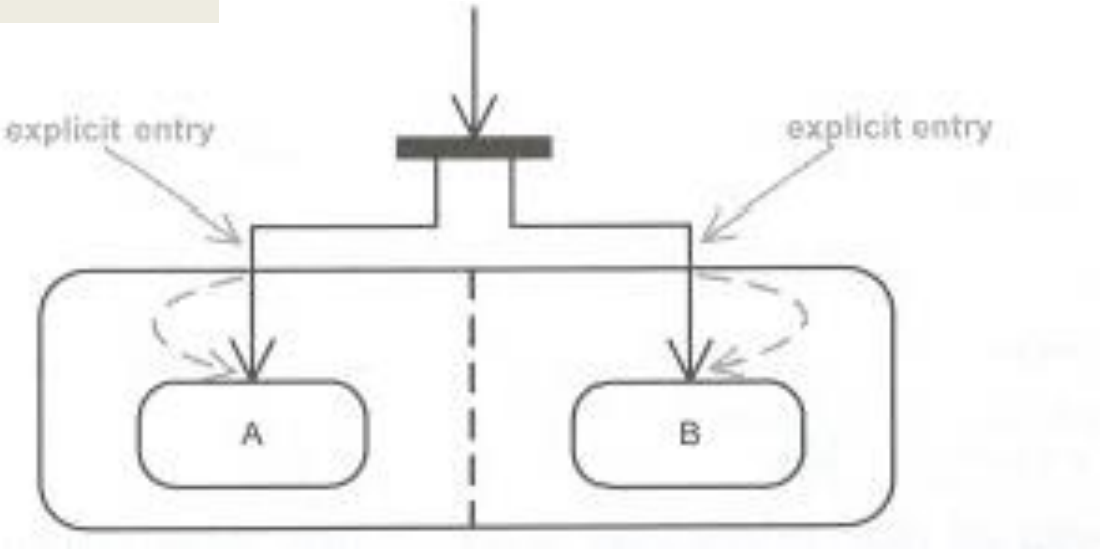


die schon beendete Region wartet dabei auf die andere:
erst wenn beide bereit sind, werden die Startzustände gleichzeitig angenommen

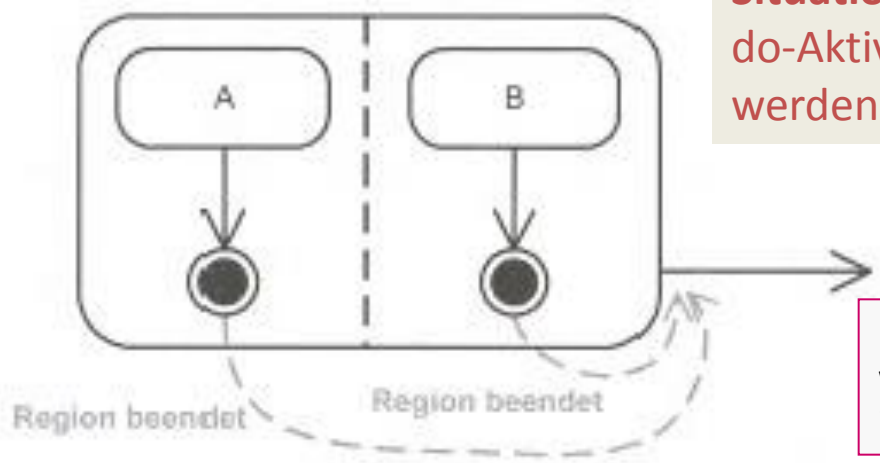
Anpassung von Semantik-Regeln (2)

Explicit-Entry

gleichzeitige Aktivierung beider Regionen, ohne deren Startzustände zu benutzen



Anpassung von Semantik-Regeln (3)



Situation:
do-Aktivitäten von A und B werden beendet

In einem Arbeitsschritt
werden die exit-Aktivitäten von innen nach außen abgearbeitet

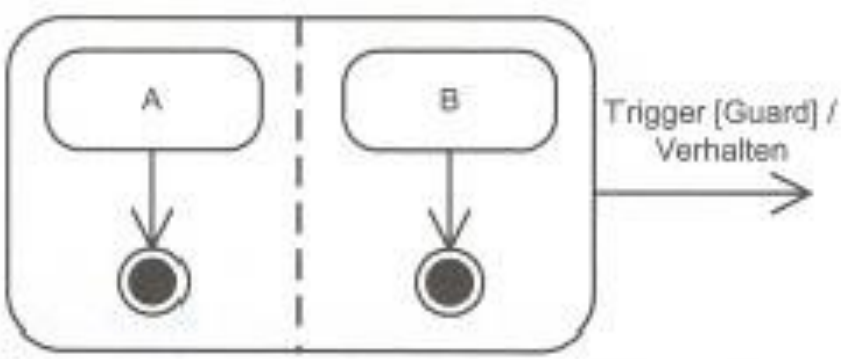
Verlassen eines orthogonal zusammengesetzten Zustandes durch das Erreichen von Endzuständen
→ Implizites Komplettierungsereignis (impliziter Trigger)

Run-to-Completion
verlangt ein zwischenzeitliches Verweilen der zuerst fertigen Region in ihrem Endzustand bis die andere Region sich ebenfalls
in einem Endzustand befindet

Natürlich muss zusätzlich auch die do-Aktivität des zusammengesetzten Zustandes beendet sein

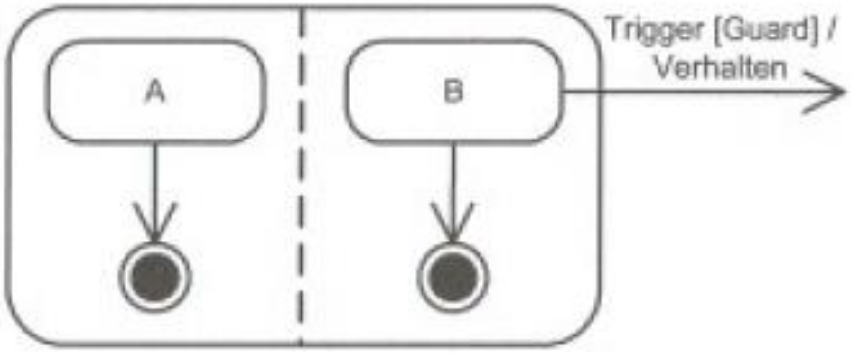
Anpassung von Semantik-Regeln (4)

getriggertes Verlassen eines orthogonal zusammengesetzten Zustandes



alle Regionen werden gleichzeitig unmittelbar mit Eintritt des Ereignisses verlassen
(implizite Synchronisation kleinster unterbrechbarer Elementar-Aktionen)

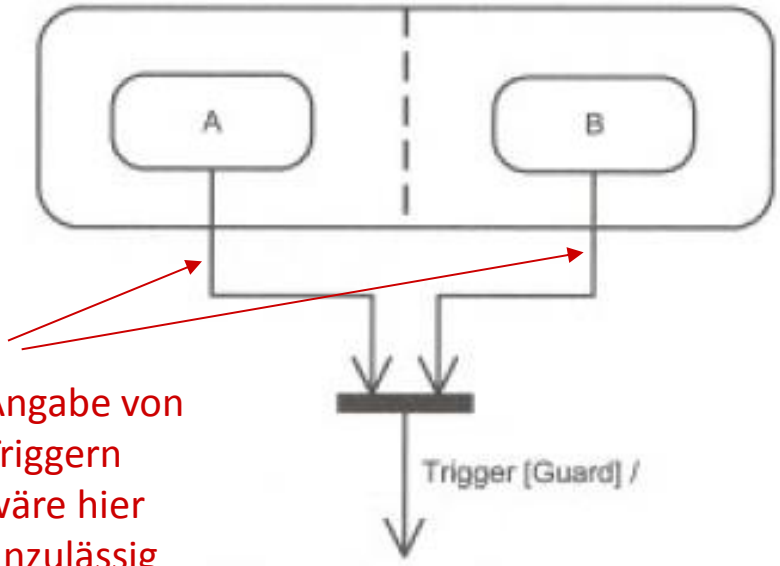
Trigger für einen Unterzustand



Ereignis verursacht synchronisierte Unterbrechungen der anderen Zustände

Anpassung von Semantik-Regeln (5)

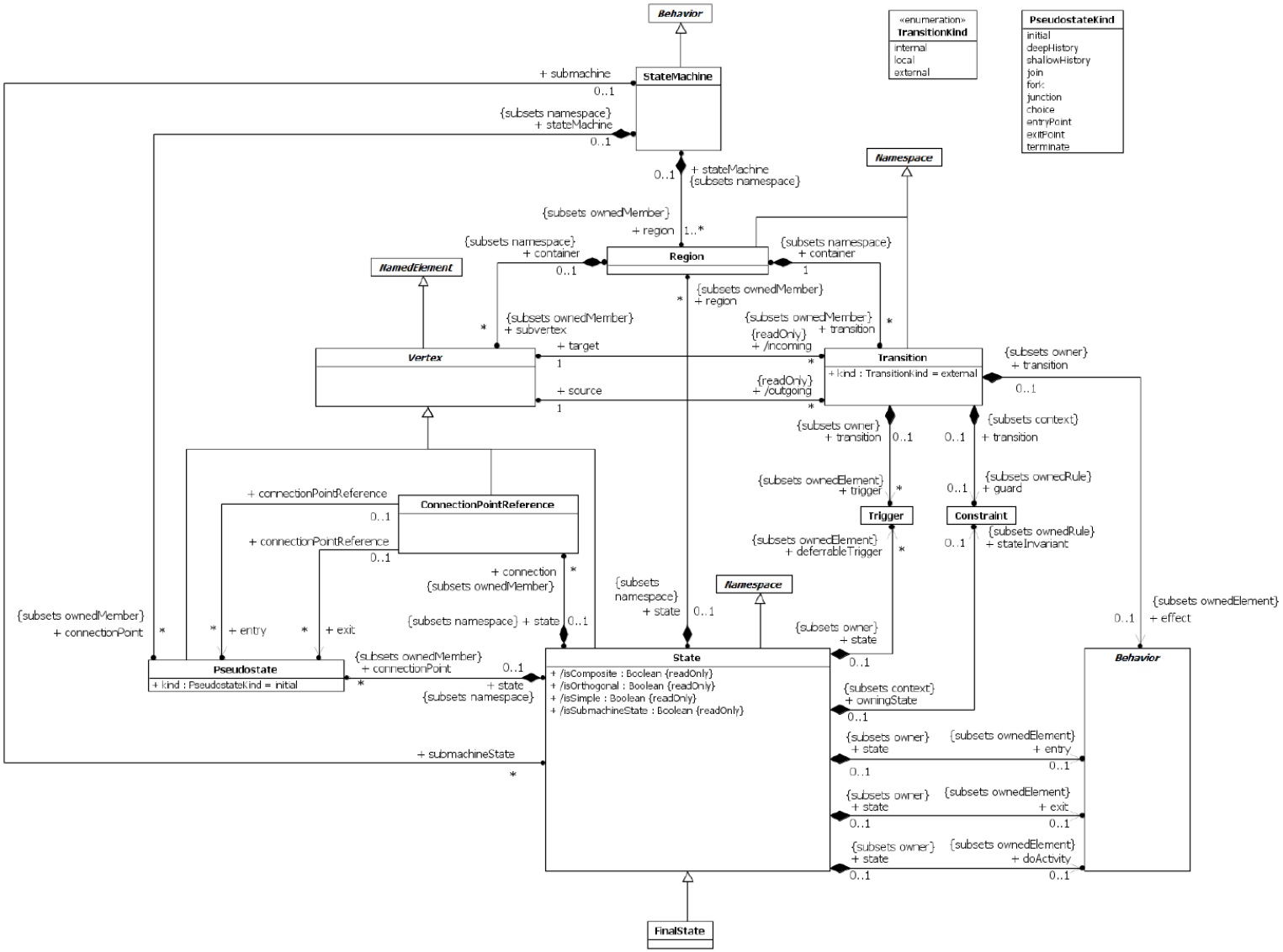
Trigger für Unterzustände jeder Region



Angabe von Triggern wäre hier unzulässig

Synchronisation sämtlicher Unterbrechungen

Metamodell (UML-Standard)



Zustandsautomaten

1. Einführung
2. Einfache Zustandsautomaten
3. Trigger-Arten
4. Semantik der Ereignisbehandlung
5. Allgemeine Charakterisierung (Wdh.)
6. Zusammengesetzte Zustände
7. Run-to-Completion-Semantik-Präzisierung
8. Pseudozustände, Unterautomaten, History-Zustände
9. Zustandstypen, Vererbung
10. Protokollautomaten

Run-to-Completion-Semantik (1)

Run-to-completion

- sichert die korrekte Ausführung der Zustandsübergangsfunktion,
  keine Concurrency-Probleme bei Ereignis-Bearbeitung

- Abarbeitung von Ereignissen resultiert in der Feuererlaubnis von keiner, einer oder mehreren Transitionen (bei parallelen Ausführungszweigen)
  - falls keine Transition feuerbar ist und es kein zu verzögerndes Ereignis vorliegt,
    wird das Ereignis gelöscht und der Run-to-completion-Schritt ist beendet

- Während einer Transition können Aktionen ausgeführt werden
  - falls eine solche Aktion der Ruf einer synchronen Operation an einem anderen Objekt darstellt, wird die Transition solange verzögert bis,
    die Zustandsmaschine des gerufenen Objektes ihrerseits den Run-to-completion-Schritt abgeschlossen hat

Run-to-Completion-Semantik (2)

Konflikte, wenn mehr als eine Transition ausgelöst werden könnte

- bei zwei oder mehr Transitionen mit dem gleichen Trigger (auf einer Zustandshierarchieebene), deren Randbedingungen alle erfüllt sind, wird nichtdeterministisch gewählt
- bei Transitionen mit dem gleichen Trigger (auf unterschiedlichen Zustandshierachieebene), deren Randbedingungen alle erfüllt sind, ist die innere/tiefere höher priorisiert (→Maskierung)
- interne Transitionen mit gleichem Trigger **müssen** orthogonale Randbedingungen besitzen

Run-to-Completion-Semantik (3)

Konflikte, bei mehr als einer Region

- haben mehrere Transitionen unterschiedlicher Regionen den **gleichen Auslöser** in ihren jeweiligen aktiven Zuständen, so werden all diese Transitionen innerhalb desselben Ausführungsschrittes ausgeführt

- enthält die aktive Konfiguration einer Region einen Zustand, in dem für das aktuelle Ereignis ein *defer* spezifiziert ist, und in einem Zustand einer anderen Region eine Transition, dann wird *defer* ignoriert
  (d.h. das aktuelle Signal wird nur von einer Region konsumiert)

Run-to-Completion-Semantik (4)

Run-to-completion-Semantik und Thread-Ausführungssemantik

- Run-to-completion wird von aktiven Objekten
  (Instanzen von Active Classes) in ihren eigenen Threads ausgeführt

- Innerhalb dieser Threads muss einfache run-to-completion implementiert werden,

  jedoch können Threads zeitweilig unterbrochen/suspendiert werden, und später fortgesetzt werden

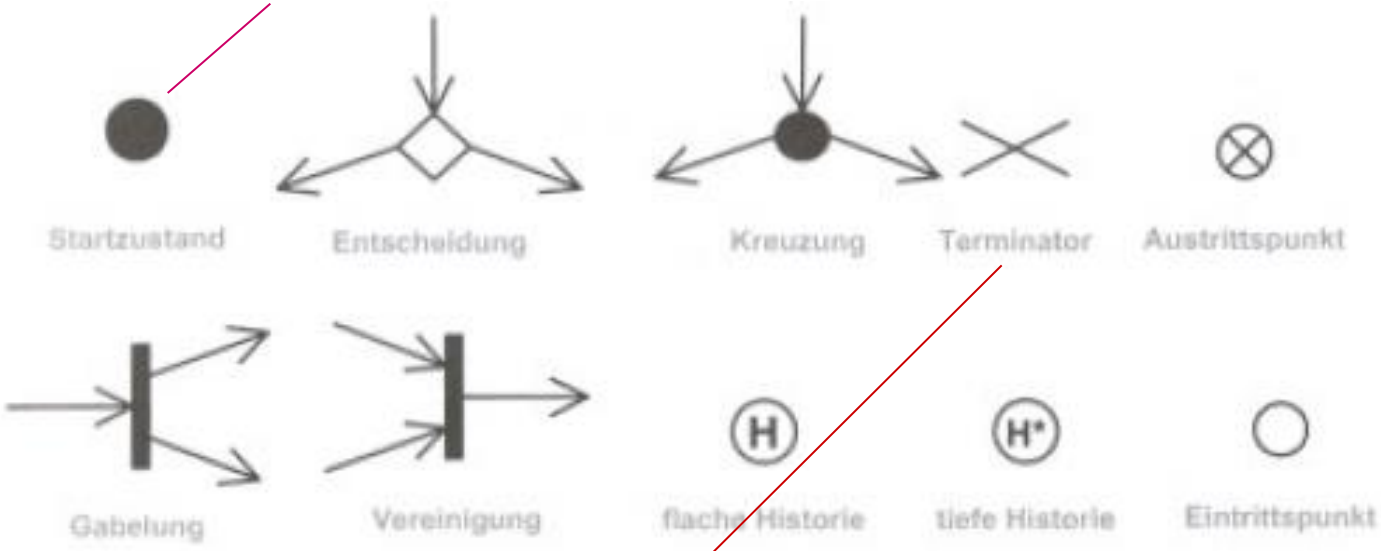
➔ Run-to-completion ist unabhängig vom Thread-Scheduling !

Zustandsautomaten

1. Einführung
2. Einfache Zustandsautomaten
3. Trigger-Arten
4. Semantik der Ereignisbehandlung
5. Allgemeine Charakterisierung (Wdh.)
6. Zusammengesetzte Zustände
7. Run-to-Completion-Semantik-Präzisierung
8. Pseudozustände, Unterautomaten, History-Zustand
9. Zustandstypen, Vererbung

Überblick vordefinierter Pseudozustände

- max. ein Startzustand mit genau einer initialen Transition (evtl. Trigger ohne Randbedingung)



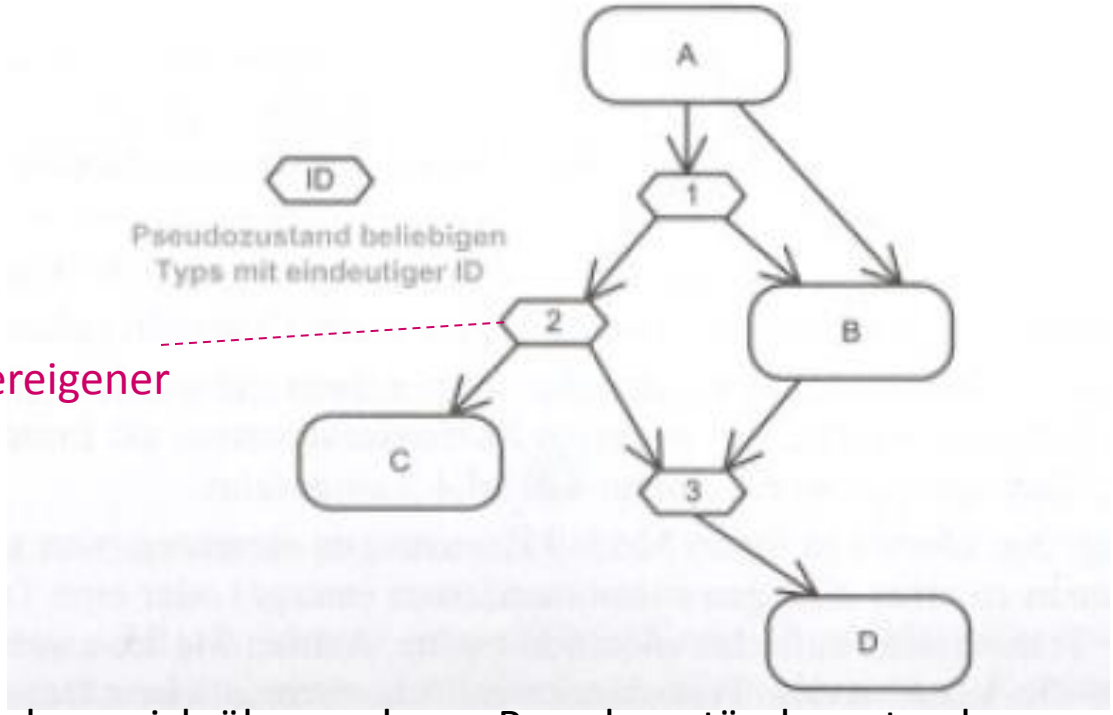
Endzustand

Unterschied:

- bei Erreichen eines **Endzustandes** bricht Automat ab
- bei Erreichen eines **Terminators** wird zusätzlich der Lebenslauf des zugeordneten Classifiers beendet

Nutzereigene Pseudozustände

keine Trigger an Ausgängen nutzereigener Pseudozustände erlaubt

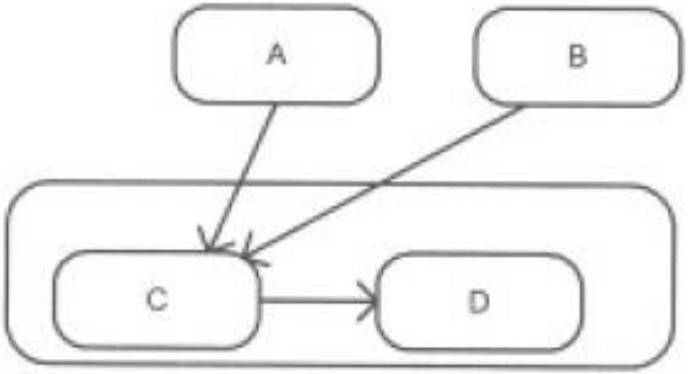


eine Transition kann sich über mehrere Pseudozustände erstrecken (UML spricht von einem Transitionsverbund)

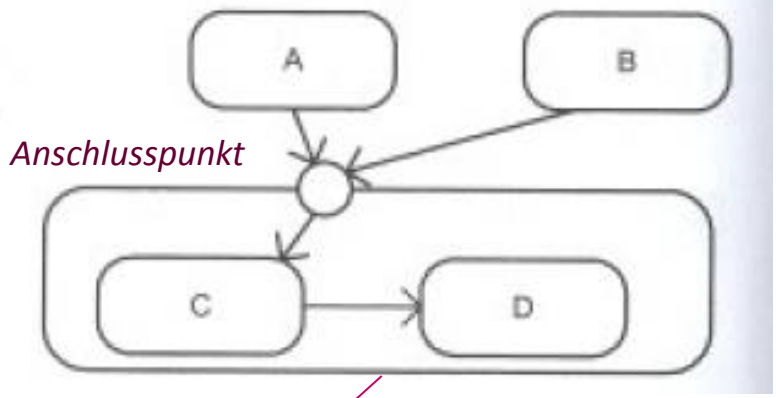
Transition A → D ist ein Verbund über 1, 2, 3

Unterautomat: Motivation

- Konzept zur weiteren Unterstützung von Zustands- bzw. Automaten-Dekomposition
- besitzen mehrere Zustände eines Automaten die gleiche oder sehr ähnliche Dekompositionen,
  dann empfehlen sich eigenständige Teilautomaten-Definitionen
- Spezifikation von Teilautomaten folgt den Regeln für komplette Automaten, benötigen zwei neue **Pseudozustände**: entry, exit

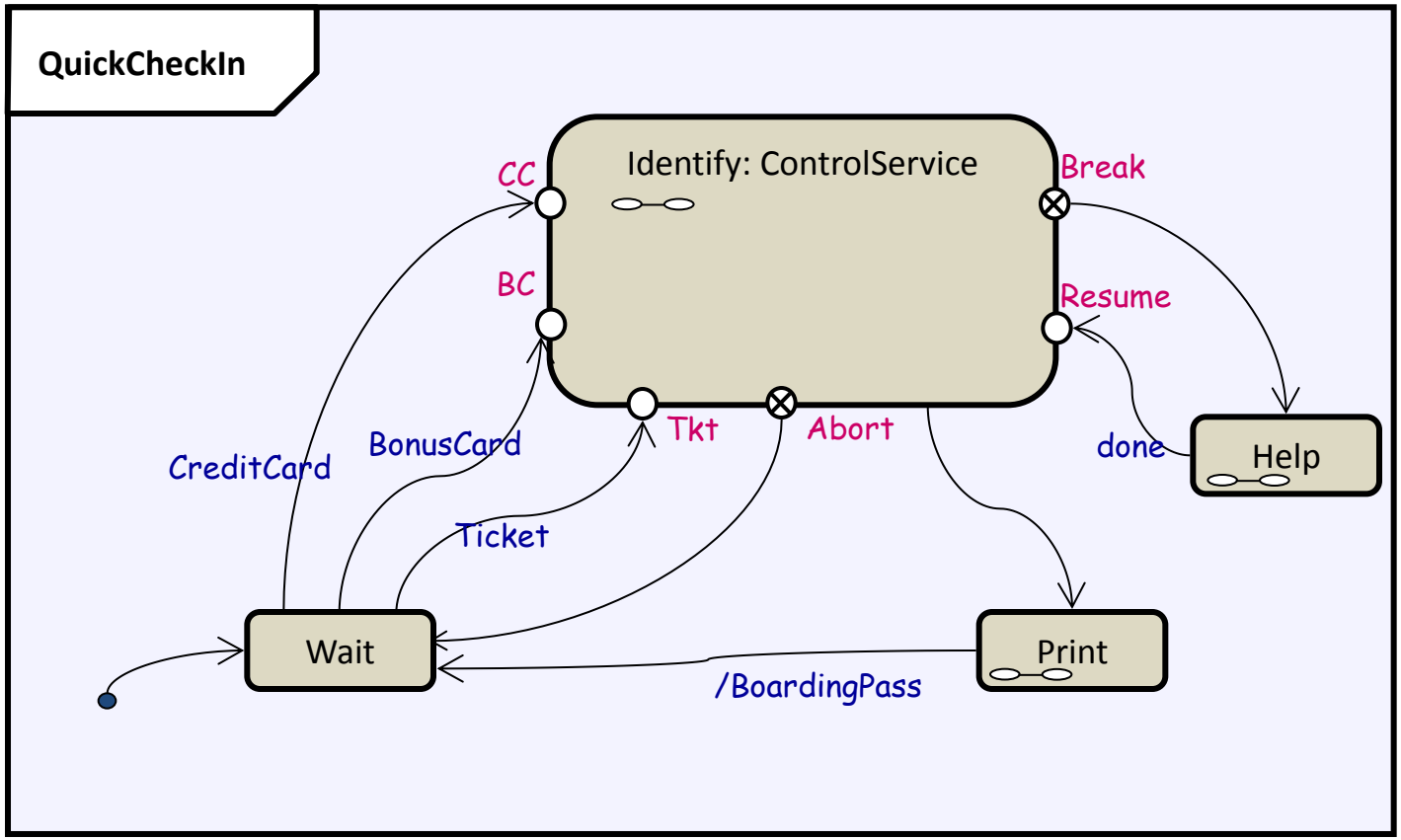


schlecht,
nicht in separate Diagramme zerlegbar



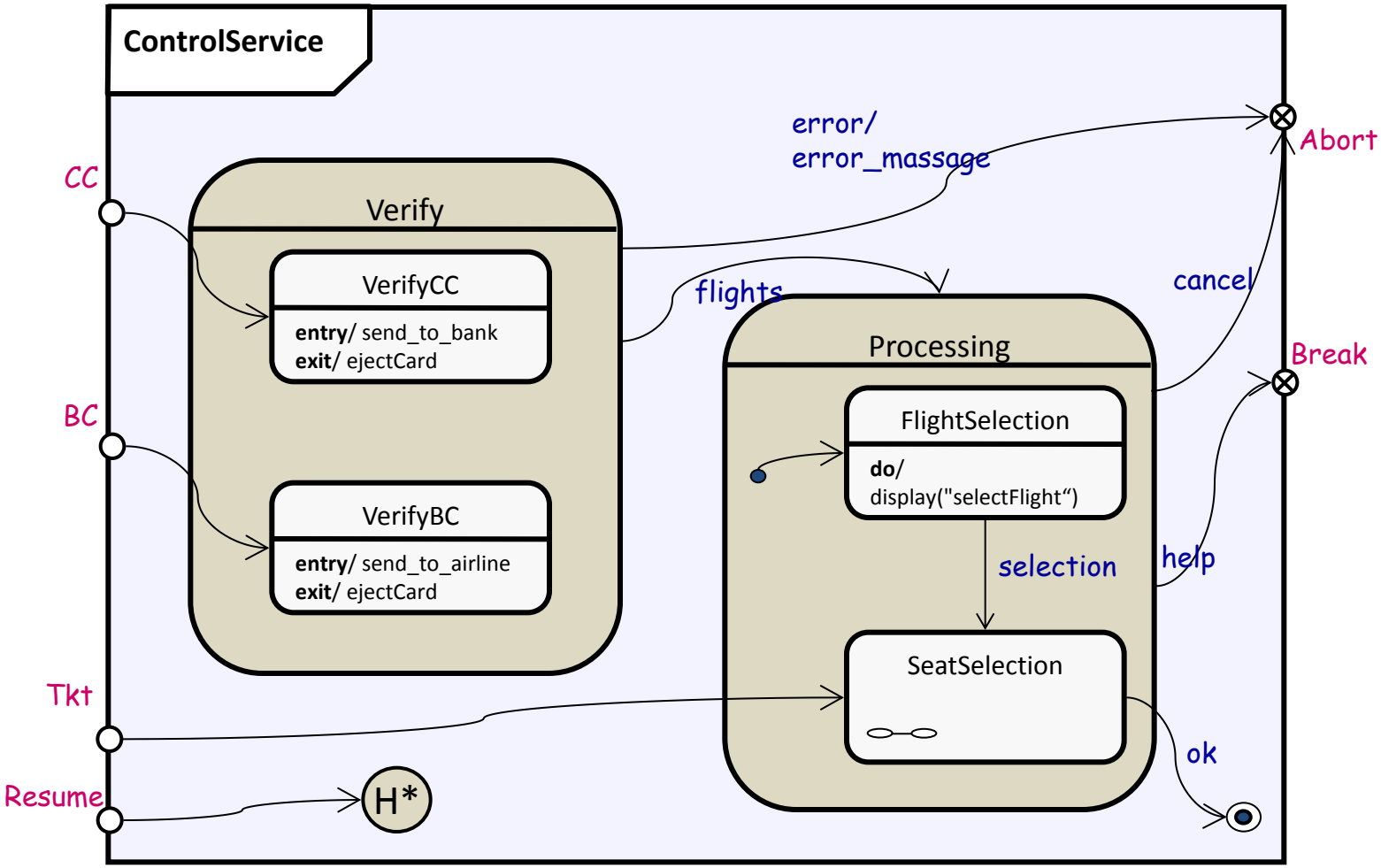
Unterautomat

Beispiel: Quick-Check-In-Automat (1)



- Anschlusspunkte liegen normalerweise auf dem Rand, aber auch innerhalb/außerhalb möglich
- unabhängig von der Existenz von Eintrittspunkten sollte ein Teilautomat stets einen Initialzustand mit einer initialen Transition besitzen

Beispiel: Quick-Check-In-Automat (2)



- Unterautomatenzustände sollten außer in Ausnahmesitationen nur über Austrittspunkte oder über Beendigungstransitionen verlassen werden

History-Zustand

- Ein mit einem History-Zustand ausgestatteter Kompositionszustand “führt Buch” über den zuletzt aktiven Unterzustand,
  der im Zuge einer Transition in den Historie-Zustand erneut angenommen wird
  tiefe History ←→ **flache Historie**
- komplette Unterzustands-konfiguration (Unterzustand der ersten Ebene)

- Bei erneuter Ausführung der Entry-Aktionen der jeweilgen History-Zustandskonfiguration

Zustandsautomaten

1. Einführung
2. Einfache Zustandsautomaten
3. Trigger-Arten
4. Semantik der Ereignisbehandlung
5. Allgemeine Charakterisierung (Wdh.)
6. Zusammengesetzte Zustände
7. Run-to-Completion-Semantik-Präzisierung
8. Pseudozustände, Unterautomaten, History-Zustände
9. Zustandstypen, Vererbung
10. Protokollautomaten

Spezialisierung von Zustandsautomaten (1)

Zustandsmaschinen

- … sind Classifier und können spezialisiert werden
  - setzt i.allg. eine Spezialisierung der zugeordneten Kontext-Classifier voraus
  - Hinzufügen von Zuständen, Regionen, Transitionen

- … können bei der Spezialisierung erweitert und modifiziert werden
  - neue Zustände und Transitionen können hinzugefügt werden (Vor. Geschlossener Graph)
  - die Ereignismenge kann eweitert werden
  - Zustände und Transitionen können re-definiert werden

Notation

- Schlüsselwort extended
- originale Elemente der Basis-Zustandsmaschine werden in der Spezialisierung gestrichelt dargestellt

Zulässige Spezialisierungen und Re-Definitionen

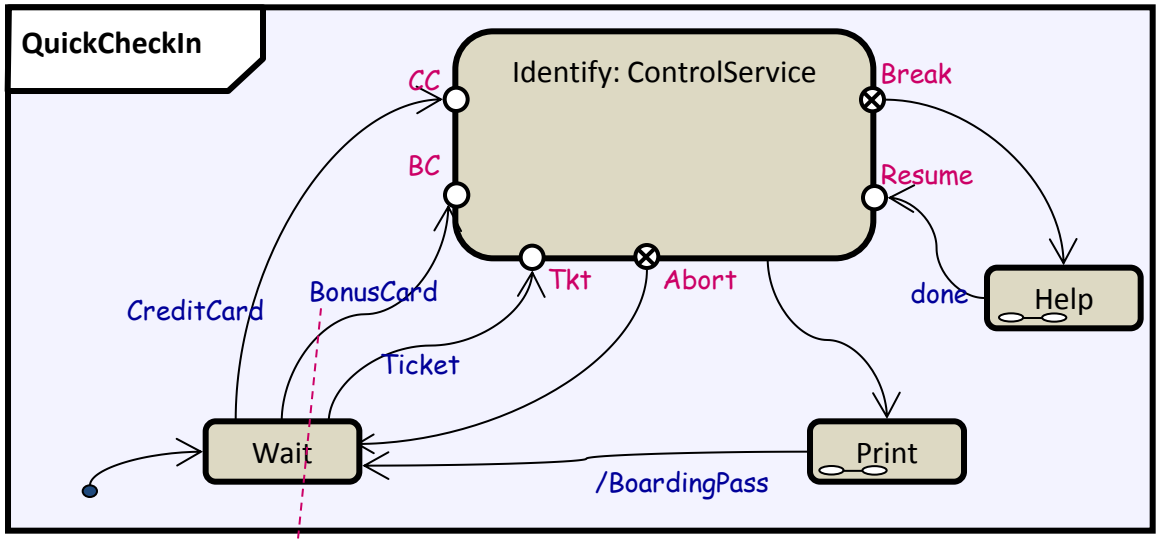
- **Re-definition von Zuständen**
  - einfache Zustände können zu zusammengesetzten oder orthogonalen Zuständen erweitert werden
  - zusammengesetzte oder orthogonaler Zustände können durch Erweiterung um Regionen re-definiert werden
  - zusammengesetzte oder orthogonale Zustände können um neue Unterzustände und Transitionen erweitert werden
    → **neue** entry-, exit- und do-Aktivitäten,
    → **weitere** Transitionen

- **Re-Definition des Automatendiagramms**
  - in einem Automatenzustand kann der verwendete Automat(typ) ausgetauscht werden (eine Art **virtueller Typ**);
    → Bedingung: der neue Automat(typ) muss mindestens alle Ein- und Ausgangspunkte des alten besitzen

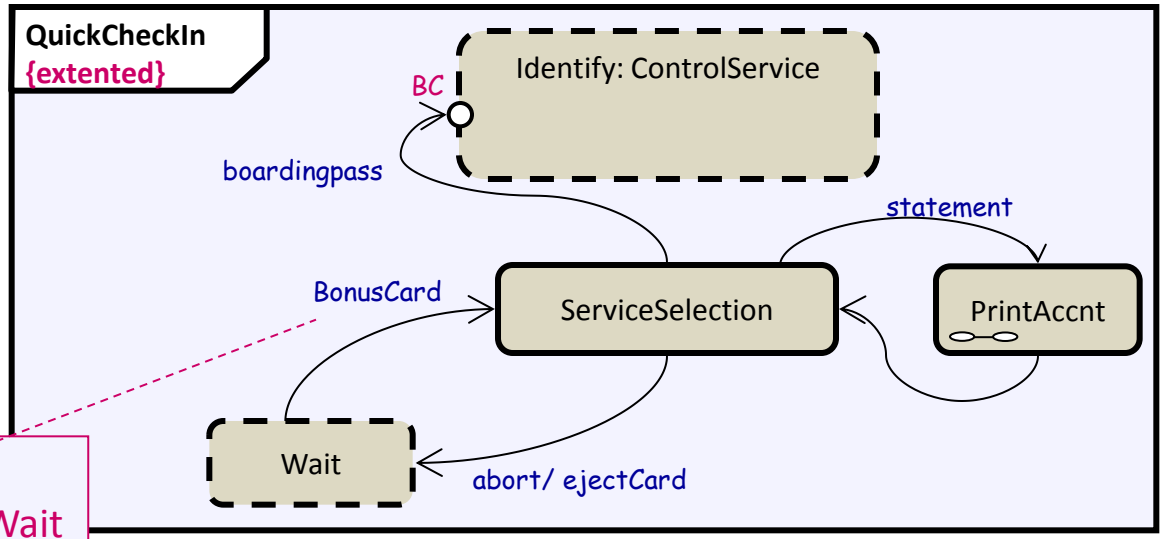
- **Re-Definition von Transitionen**
  - die neue Transition besitzt gleichen Ausgangszustand und gleichen Trigger
  - Zielzustand, Randbedingung und Aktion dürfen redefiniert werden

- **Finalität einer Re-Definition**
  - geerbte Zustände und Transitionen, die mit **<<final>>** **Schlüsselwort** gekennzeichnet sind, können nicht mehr re-definiert werden

Beispiel: Quick-Check-In-Automat (3)



Start,
Wait,
Help,
Print
und deren Transitionen werden ebenfalls geerbt



Redefinition von Transition BonusCard für Wait

Mehrfachvererbung

- wird ein Kontext-Classifier durch Mehrfachvererbung aus mehreren Classifiern mit individuellen Zustandsmaschinen gebildet,

  so erhält er einen **orthogonalen Zustandsautomaten**, der für jeden ererbten Automaten eine eigene Region erhält

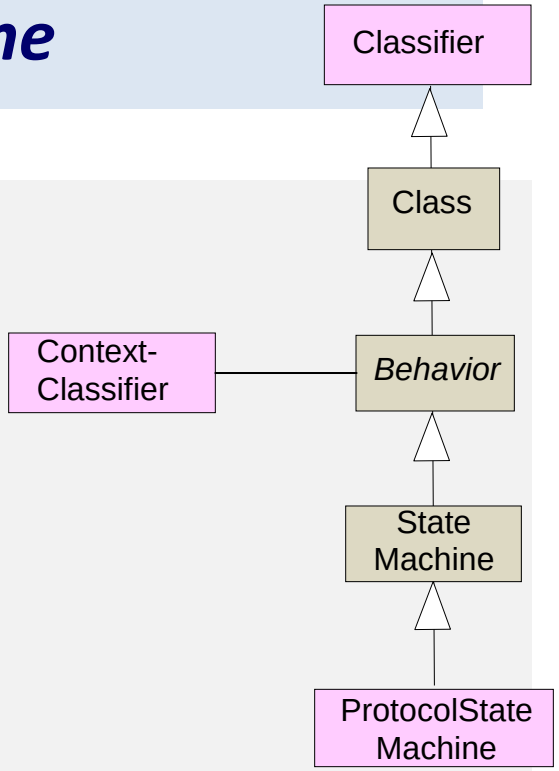
- **evtl. Problem**: mehrfache Einerbung von einer Struktur, einer Zustandsmaschine

- SDL (UML-Profil) erlaubt **keine** Mehrfachvererbung von Agenten

Zustandsautomaten

1. Einführung
2. Einfache Zustandsautomaten
3. Trigger-Arten
4. Semantik der Ereignisbehandlung
5. Allgemeine Charakterisierung (Wdh.)
6. Zusammengesetzte Zustände
7. Run-to-Completion-Semantik-Präzisierung
8. Pseudozustände, Unterautomaten, History-Zustand
9. Zustandstypen, Vererbung
10. Protokollautomaten

Protokoll-Zustandsmaschine



- **Im Metamodell**: Spezialisierung von *State Machine*
- **Zweck**:
  eine Protokollzustandsmaschine spezifiziert,
  - welche Operationen/Signale eines Context-Classifiers
  - in welchem Zustand und
  - unter welchen Bedingungen

  gerufen/akzeptiert werden können

  ➔ Spezifikation von Aufrufreihenfolgen (Protokoll)
  Ähnlichkeit zum OSI-Modell: *Service-Spezifikation*

  ➔ Festlegung des Nutzungsmodus von Operationen
  - In welcher Konfiguration kann eine Operation oder ein Signalempfang genutzt werden? (Angabe von Zuständen, Pre-Condition)
  - Welches Resultat kann von einer Operationsnutzung erwartet werden? (Post-Condition)
- **Einsatz**:
  - nur im Kontext eines *Classifiers* erlaubt und **nicht** aber für *Behavioral Feature* selbst

typische Anwendung:
- Interface-Spezifikation

Protokoll- Zustandsmaschine

- Notation – Schlüsselwort {protocol}
- Besonderheit

Protokoll-Zustand

kein Verhalten im Zustand erlaubt
optionale Invariantenangabe zusätzlich zu einer implizit geltenden Invariante

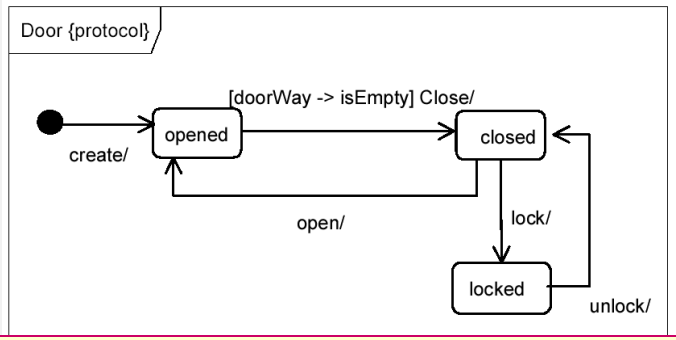
Protokoll-Transition

spezifiziert legale Transition für eine Operation
Syntax: [pre condition] trigger / [post condition]
keine Aktionen!

muss bei Trigger-Auftreten erfüllt sein

muss nach Übergang gelten

Beispiel:
elektrisches Tor

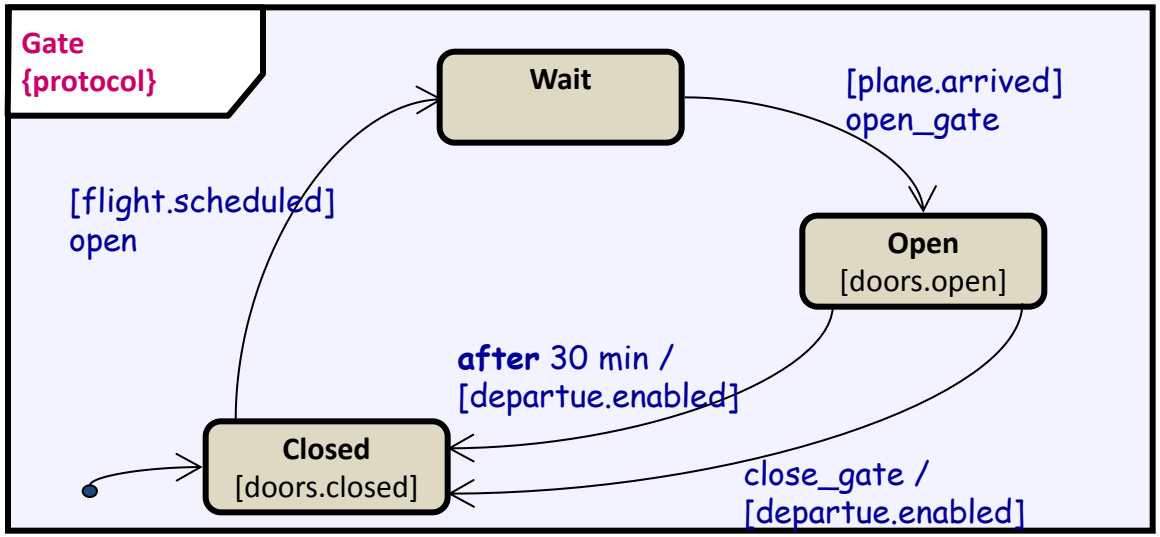


Ausführung erfolgt, sobald
Call-Trigger anliegt

Bem.: Vor- und Nachbedingung einer Transition werden durch Bedingungen des Quell- und Zielzustandes ergänzt
(meist sind diese aber bereits durch implizite Invariante(=Zustand) erfüllt)

Nachbedingung ist meist Bedingung des Zielzustandes und diese ist meist implizite Invariante

Beispiel: Flughafen-Gate



Fortsetzung: semantische Präzisierung

- Randbedingungen schränken das Schalten von Transitionen ein
- mit einer Protokoll-Transition sind niemals Aktionen verbunden

  entry-, exit-, do-Aktivitäten sind für Protokollzustände **nicht** zulässig
- unter Einhaltung dieser Restriktionen sind sämtliche Dekompositionsmöglichkeiten von gewöhnlichen Zustandsautomaten anwendbar

Protokoll-Zustandsmaschine

- bieten – im Gegensatz zu lokalen Pre- und Post-Conditions –
  eine **globale Übersicht** über die protokollarische Nutzung einer Classifier-Instanz

- deklarative Protokoll-Zustandsmaschine:
  - Spezifikation **sämtlicher legaler** Transitionen für jede Operation/Signal eines Classifiers
    exakte Trigger für die Transitionen sind nicht notwendigerweise definiert
  - erlaubt die Definition eines Vertrages zur Nutzung eines Classifiers (z.B. für Interfaces oder Ports)

- ausführbare Protokoll-Zustandsmaschine:
  - Spezifikationen **aller** Ereignisse (**aller Trigger** für Transitionen)
  - kann vollständig ausgeführt werden