ModSoft

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

Teil III b: Verhaltensmodellierung: Aktivitäten

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

fischer@informatik.hu-berlin.de

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. Aktionen (Nachtrag)
6. Datenströme
7. Parametersätze: Gruppierung von Parametern
8. Behandlung von Ausnahmen
9. Beispiel
10. Weitere Aspekte (summarische Betrachtung)

Diskussion offener Fragen:
Referenzsemantik, Bsp. DataStore

Aktionen

- **Elementare Bausteine** für beliebiges benutzerdefiniertes Verhalten
  können Eingabewerte zu Ausgabewerten verarbeiten
- **Atomar**, können aber abgebrochen werden
- **Sprachunabhängig**
  allerdings Definition in beliebiger Programmiersprache möglich
- Spezielle **Notation** für bestimmte Aktionsarten

  Gruppierung
  - Kommunikationsbezogene Aktionen
    (z.B. Signale und Ereignisse)
  - Objektbezogene Aktionen
    (z.B. Erzeugen und Löschen von Objekten)
  - Strukturmerkmals- und variablenbezogene Aktionen
    (z.B. Setzen und Löschen einzelner Werte von Variablen)
  - Linkbezogene Aktionen
    (z.B. Erzeugen und Löschen von Links zwischen Objekten sowie Navigation)

  <Es fehlen noch:>
  - Opaque-Aktionen

~ Spezifikation in einer textuellen konkreten Syntax (verschieden von UML)
~ Platzhalter

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. **Diskussion offener Fragen**
6. Datenströme
7. Parametersätze: Gruppierung von Parametern
8. Behandlung von Ausnahmen
9. Beispiel
10. Weitere Aspekte (summarische Betrachtung)

Offene Fragen aus der letzten Vorlesung

1. FRAGE: Inhalte von **Objekt-Token** sind Objektreferenzen oder Objektwerte ?

aus dem UML-Standard:

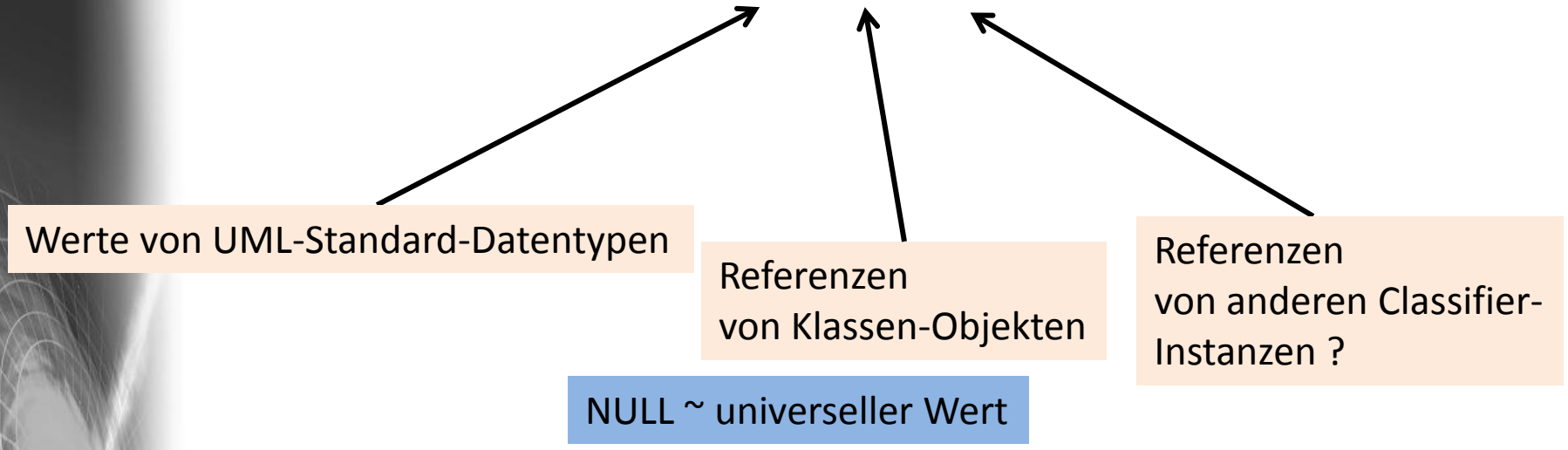
- *Tokens* are not explicitly modeled in an Activity, but are used for describing the execution of an Activity.
- An *object token* is a **container** for a value that flows over ObjectFlow edges (some object tokens can flow over ControlFlow edges, as specified by the modeler, see isControlType for ObjectNodes in sub clause 15.4).
- An object token with **no value** in it is called a *null* token.
- A *control token* affects execution of ActivityNodes, but does not carry any data, and flows only over ControlFlow edges.
- Each token is distinct from any other, even if it contains the same value as another.

Danach wäre eine Interpretation für beide Semantikvarianten denkbar

Begriffe wie *reference semantic* oder *value semantic* werden im Standard **nicht** benutzt

Inhalte von Objekt-Token

- Objekt-Token ~ Container von Werten



- Objekt-Knoten ~ Sammler von Objekt-Token
  mit Einschränkungen: Typ, Anzahl, Zustand, …

→ *damit Sammler von Daten (Werte und Objekte)*

Objekt-Knoten: DataStore

- DataStore-Knoten (lokal in einer Aktivität) halten Objekt-Token solange wie die Aktivität läuft

- Wird ein Objekt-Token vom nachfolgenden Tokenfluss akzeptiert, verlässt dieser den DataStore-Knoten. Jedoch wird eine Kopie angelegt und zurückgeschrieben.
  D.h. die einmal eingtragenen Objekte bleiben **persistent** für die Dauer der Ausführung der umfassenden Aktivität erfasst.

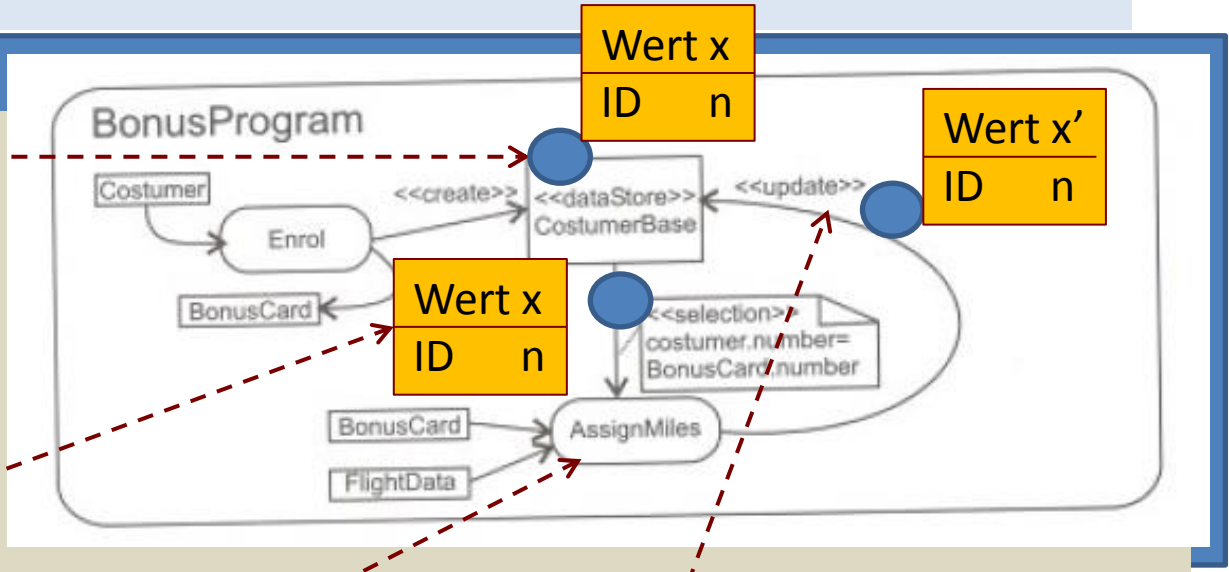
- Bei Terminierung der Aktiviät werden alle bislang abgelegten Objekt-Token vernichtet.

- Wird dem DataStore-Knoten ein Objekt-Token mit derselben Identität eines bereits erfassten Objektes abermals zugeführt, wird das Duplikat verworfen und nicht aufgenommen.

- Selektionen und Transformationen der Objekte eines Data-Store-Knotens durch den ausgehenden Objektflus durch
  a) selection and
  b) transformation

  durch ausgehenden Token-Fluss ohne Seiteneffekte (Query) möglich

Konflikt-Beispiel: Annahme einer Wertesemantik



- Erzeugung eines Objekt-Tokens mit Objekttyp CustomerBase durch Ausführung von Enrol
- Anlegen einer Kopie eines Objekt-Token (neue Container-Instanz)
- Manipulation des Objekt-Token (Ann.: customer.number=BonusCard.number) durch AssignMiles. Resultat ist geänderter Wert
- Weitere **heftige** Annahme (ohne Absicherung): update-Eintrag x' führt zu einer Aktualisierung des bisherigen Eintrages

Die Annahme einer passenden Update-Semantik ist aber **nicht zulässig**
~ update ist lediglich ein Effekt für den Ausgang von **AssignMiles**

Effect

- bei *BehavioralFeature* findet man *effect* als eine Eigenschaft von Parameter.
  Auf S. 104 steht zu Parameter:
- The *effect* property may be used to declare additional indications of the effect on values passed in or out of a Parameter.
  It is a declaration of the modeler's intent, and **does not have execution semantics**: the modeler must ensure that the Parameter has the stated effect.

➔ Damit sind das also nur Annotationen!!!
Die Datastore- Zuführung eines Objekt-Tokens unterliegt der Standardsemantik

Zurück zur Frage
reference semantic oder *value semantic*

Beantwortung von Frage 1

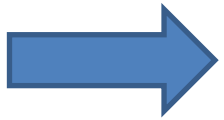
Ein starker Hinweis auf **Referenz-Semantik** findet sich bei der Definition des Join-Knoten

Bislang wurde nur die Synchronisation von eingehenden Control-Token behandelt,
Es können aber auch Objekt-Token eingehen.

Ferner kann die Semantik eines Join-Knoten variieren (MagicDraw fragt dies auch ab)

Ähnlich wie bei
- addStructuralFeatureValue-Action

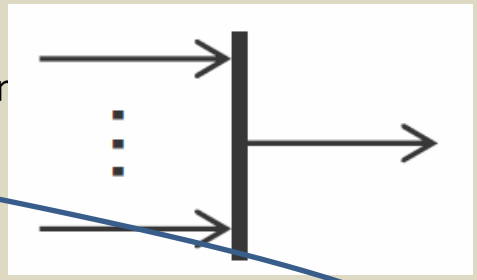
gibt es auch hier zwei Verhaltensspielarten



Join (vertiefende Betrachtung)

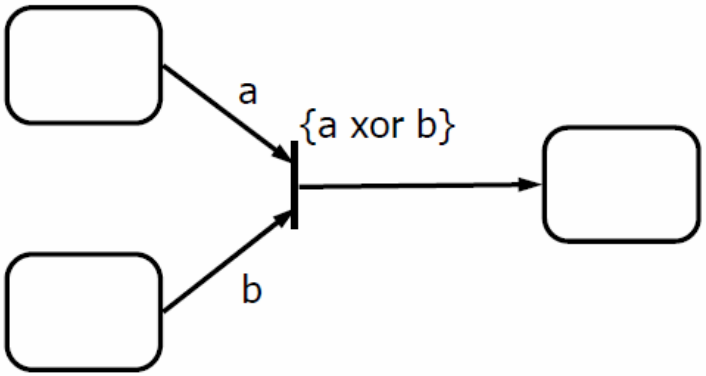
führt nebenläufige Abläufe zusammen

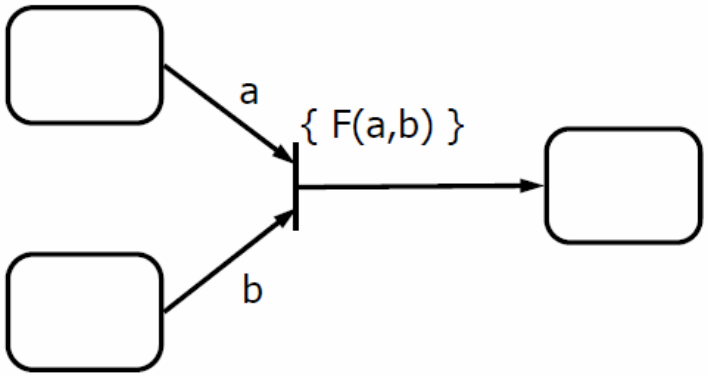
- Vereinigung der Kontroll-Token, sobald an allen Kanten vorhanden
  - an einer Kante anliegende Kontrolltoken werden vereinigt
  - Kontrolltoken verschiedener Kanten werden vereinigt und nur ein einzelnes Token weitergereicht



- Datentoken werden alle weitergereicht
  - Bei Kontroll- und Datentoken - nur Datentoken werden weitergereicht

Ausnahmen





Join-Ergänzung

Auf S. 406 im UML-Standard 2.5 findet man unter **JoinNode** folgenden Abschnitt:

- If the (implicit or explicit) joinSpec of **a JoinNode** evaluates to true, then tokens are offered on the outgoing ActivityEdge of the JoinNode according to the following rules:
  1. If all the tokens offered on the incoming edges are control tokens, then one control token is offered on the outgoing edge.
  2. If some of the tokens offered on the incoming edges are control tokens and others are object tokens, then only the object tokens are offered on the outgoing edge. Tokens are offered on the outgoing edge in the same order they were offered to the join.

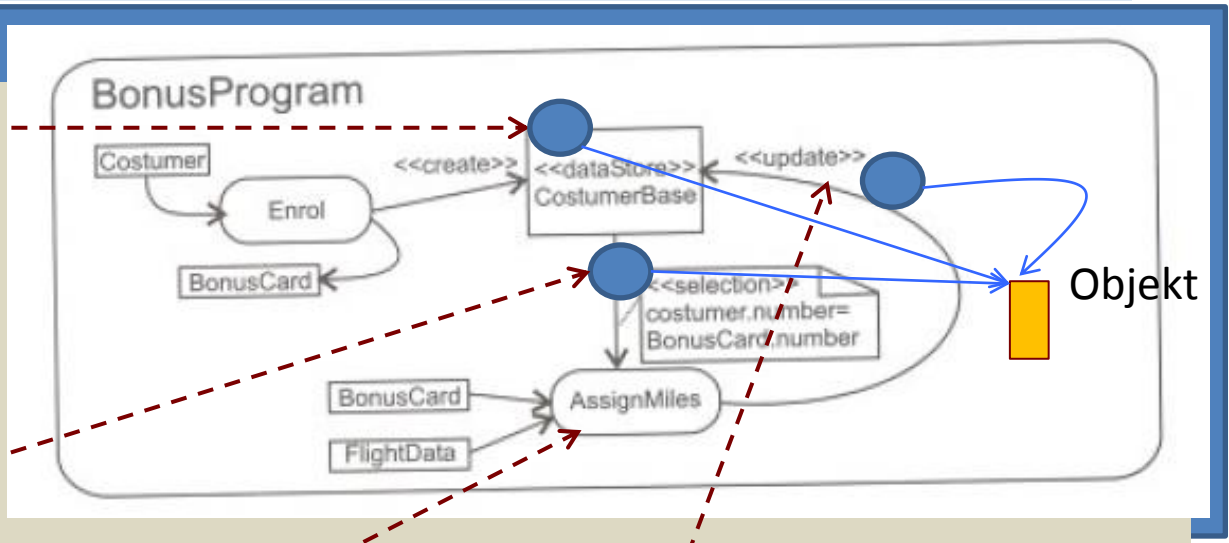
Interessant ist nun folgender Teil von Punkt 2:

- If *isCombinedDuplicate* is **true** for the **JoinNod**e, then before object tokens are offered to the outgoing edge, those containing objects with the same identity are combined into one token.

Dies deutet daraufhin,
dass es Token geben kann, die auf das gleiche Objekt verweisen.

Wie sonst könnten zwei Token ein Objekt mit der gleichen Identität enthalten?

Konflikt-Beispiel: Annahme einer Referenzsemantik

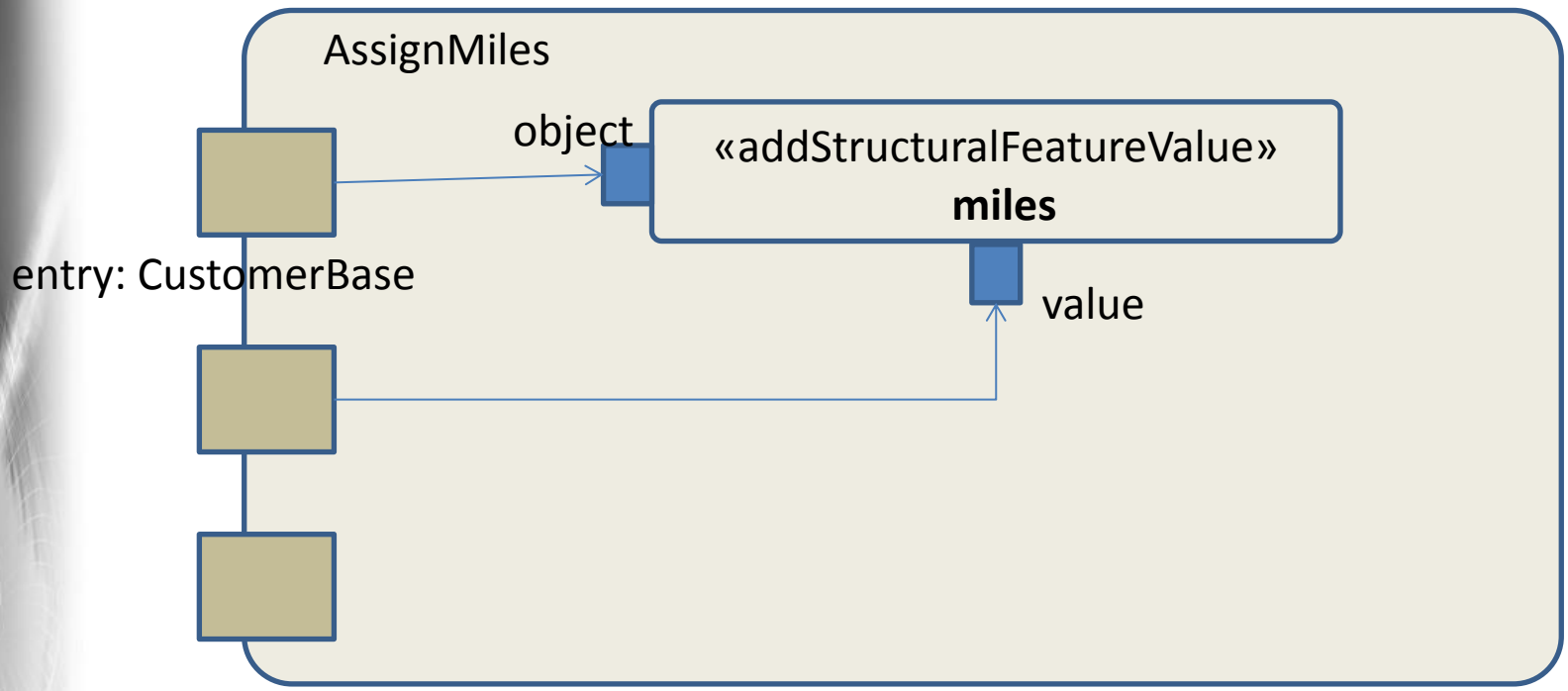


- Erzeugung eines Objekt-Tokens mit Objekttyp CustomerBase durch Ausführung von Enrol
- Anlegen einer Kopie eines Objekt-Token
- Manipulation des Objekt-Token (Ann.: customer.number=BonusCard.number) durch AssignMiles: wird referenziertes Objekt manipuiert (eigentl. **Update!!**!)
- Objekt-Token wird AssignMiles (mit übernommener Objekt-Referenz) datastore zugeführt und wegen **Verbot einer doppelten Buchführung** verworfen) (expliziter update-Übergang ist damit überflüssig)

Update-Umsetzung (1)

Annahme Klasse *CustomerBase* hat ein Attribut *miles*

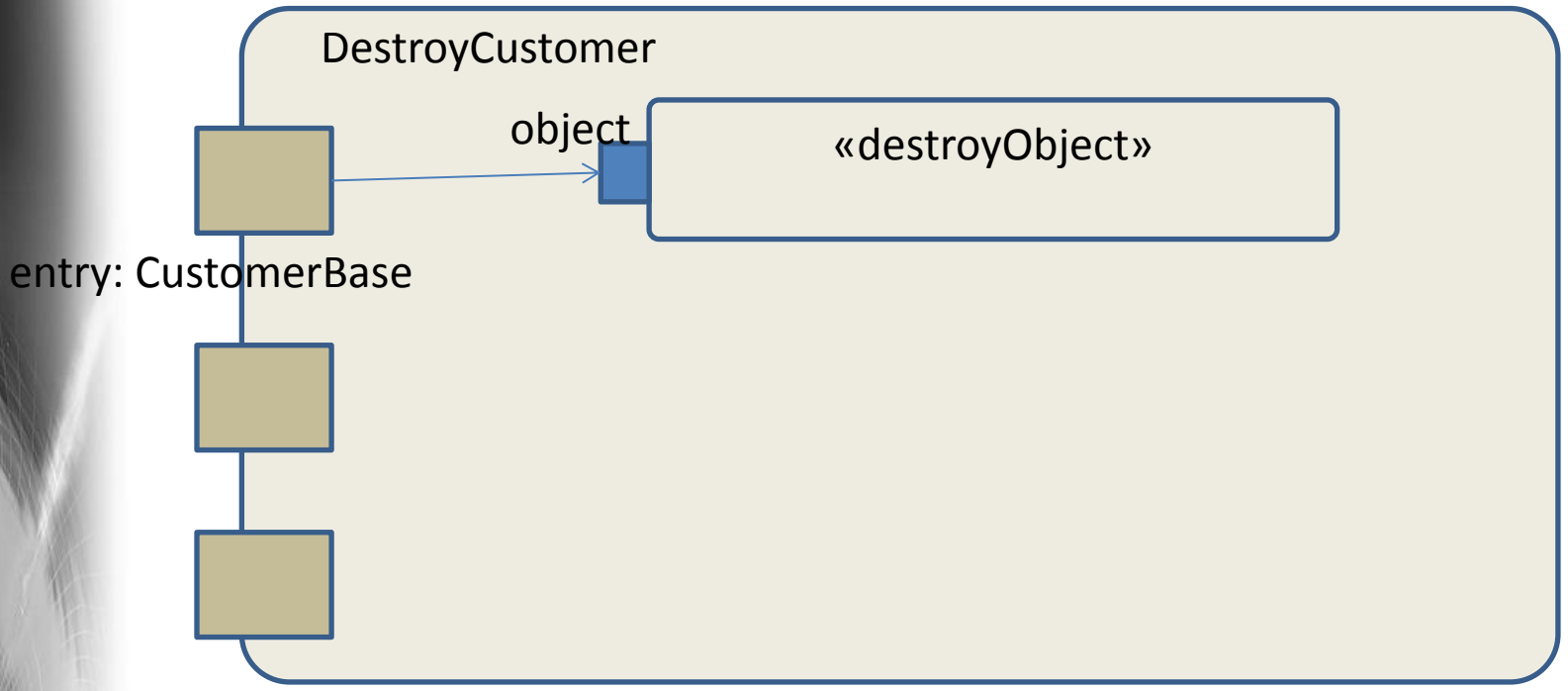
Achtung: Objektknotenname und Objekttypbezeichner sind zufällig gleich: *CustomerBase*



Wo bleibt das Resultat?

Antwort:
ObjektKnoten *CustomerBase* enthält eine Referenz zu dem aktuellen Objekt-Wert.

Löschen von Einträgen in DataStore



Ann.:
Referenz zum Objekt (enthalten in DataStore-Knoten) wird zu NULL
Was passiert mit dem Objekt-Token ?

Was sagt der Standard?

S. 480

- A DestroyObjectAction is an Action that destroys the object on its target InputPin. [..] When an object is destroyed, it is no longer classified under any Classifier. Other than the options described below for isDestroyLinks and isDestroyOwnedObjects, the Action has no other effect. In particular, no behaviors are executed, no state machine transitions are triggered, and **references to the destroyed objects are unchanged** (except see isDestroyLinks below). **The result of referencing an object that has been destroyed is not defined in this specification.**

- If **isDestroyLinks is true**, links in which the object participates are destroyed along with the object according to the semantics of DestroyLinkAction, except for link objects, which are destroyed according to the semantics of DestroyObjectAction with the same attribute values as the original DestroyObjectAction.

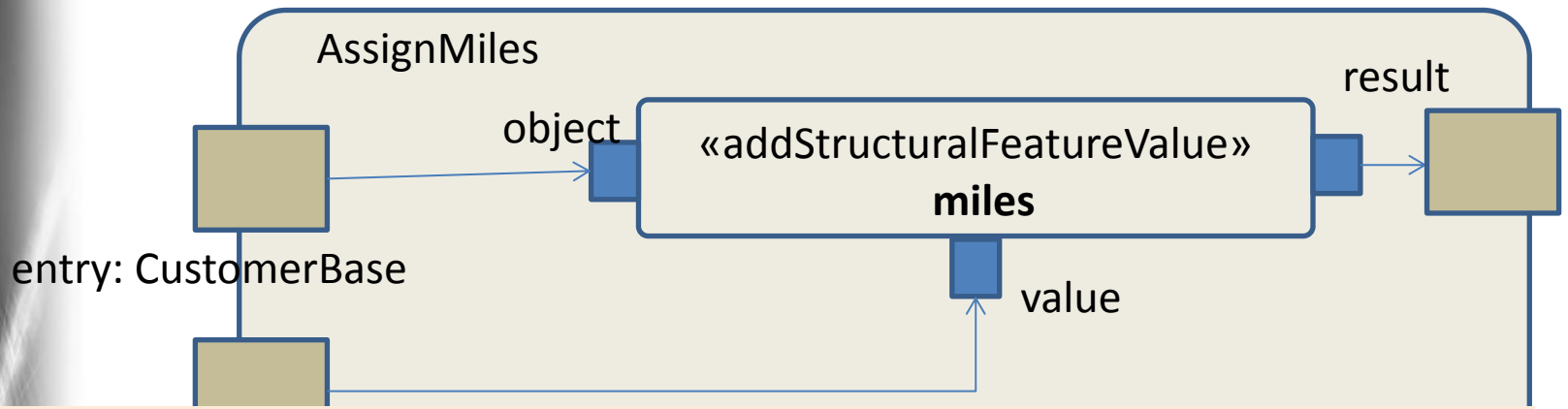
- If **isDestroyOwnedObjects is true**, objects owned by the object through composite aggregation are destroyed according to the semantics of DestroyObjectAction with the same attribute values as the original DestroyObjectAction.

- **Destroying an object that is already destroyed has no effect.**

Update-Umsetzung (2)

addStructuralFeature kann optional ein AusgabePin besitzen

Achtung: Objektknotenname und Objekttypbezeichner sind zufällig gleich: *CustomerBase*



S. 488 [..] The other kinds of StructuralFeatureActions, WriteStructuralFeatureActions (including AddStructuralFeatureValueActions
and RemoveStructuralFeatureValueActions) [..] modify the values of a StructuralFeature.

These Actions may optionally have a **result OutputPin**.
If a result OutputPin is provided, then the input object, as modified, is placed on this OutputPin. If the input object is a data value (an instance of a DataType), then a copy of the input data value is placed on the output pin, but with the
appropriate structural feature modified. As a data value does not have an independent identity, the only way to obtain the modified data value is through the use of the result OutputPin.

Offene Fragen aus der letzten Vorlesung

2. FRAGE: Spezialfall bei der Beendigung von Aktivitäten

aus dem UML-Standard:

- auf S. 393:

The execution of an Activity with no streaming Parameters **completes** when it has no nodes executing and no nodes enabled for execution, or when it is explicitly terminated using an ActivityFinalNode (see sub clause 15.3).

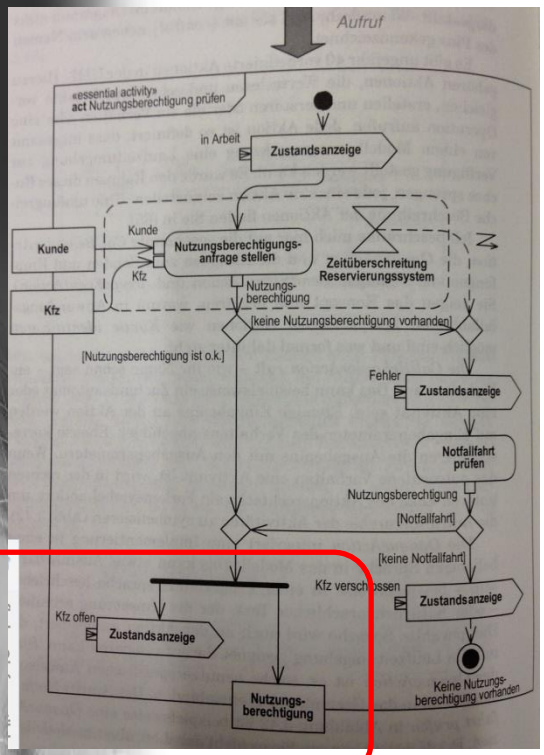
- auf S. 404-405

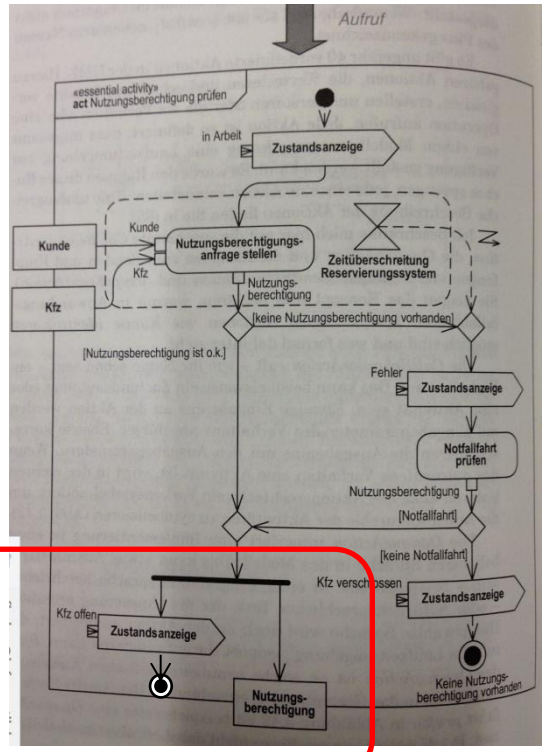
An ActivityFinalNode is a FinalNode that stops all flows in an Activity [..].

A token reaching an ActivityFinalNode owned by an Activity **terminates** the execution of that Activity. [..]

The **termination of Activity execution** shall destroy all tokens held in any ObjectNodes other than output ActivityParameterNodes and shall terminate the execution of any behaviors synchronously called from the Activity.

T. Weilkiens: "SystemsEngineering mit SysML/UML"





ist korrekt

ist ebenso korrekt

bei Beendigung steht in beiden Fällen der Ausgabe-Parameter (= Null-Token) zur Verfügung

Sowohl die Ausführung der Knotenfunktionalität als auch die Tokenflüsse sind (hier) zeitlos

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. **Diskussion offener Fragen**
6. Datenströme
7. Parametersätze: Gruppierung von Parametern
8. Behandlung von Ausnahmen
9. Bespiel
10. Weitere Aspekte (summarische Betrachtung)

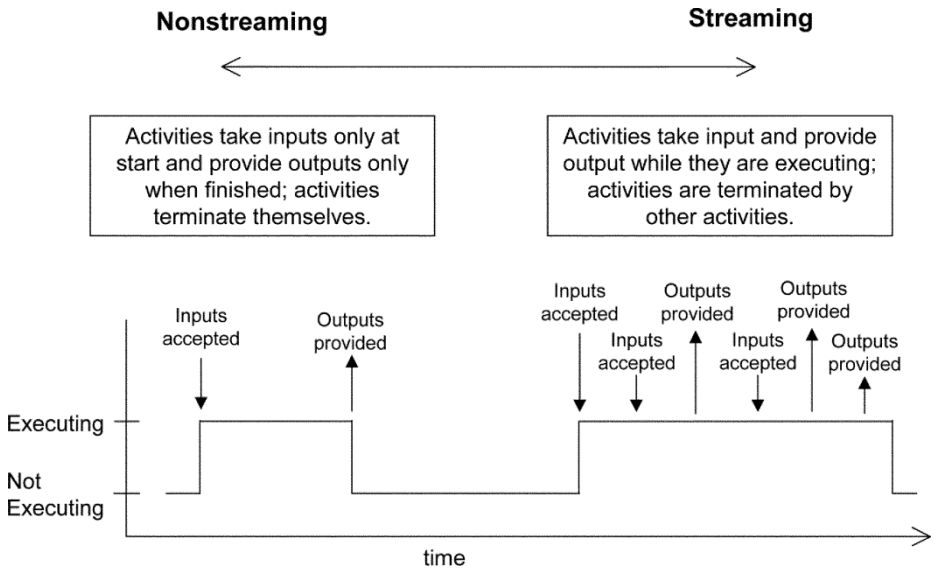
Datenströme (allg.)

Unterschied zu einer Folge von DatenToken eines Typs?

- Mit **Datenströmen** (englisch **data streams**) bezeichnet man einen kontinuierlichen Fluss von Datensätzen, dessen Ende meist nicht im Voraus abzusehen ist; die Datensätze werden fortlaufend verarbeitet, sobald jeweils ein neuer Datensatz eingetroffen ist.

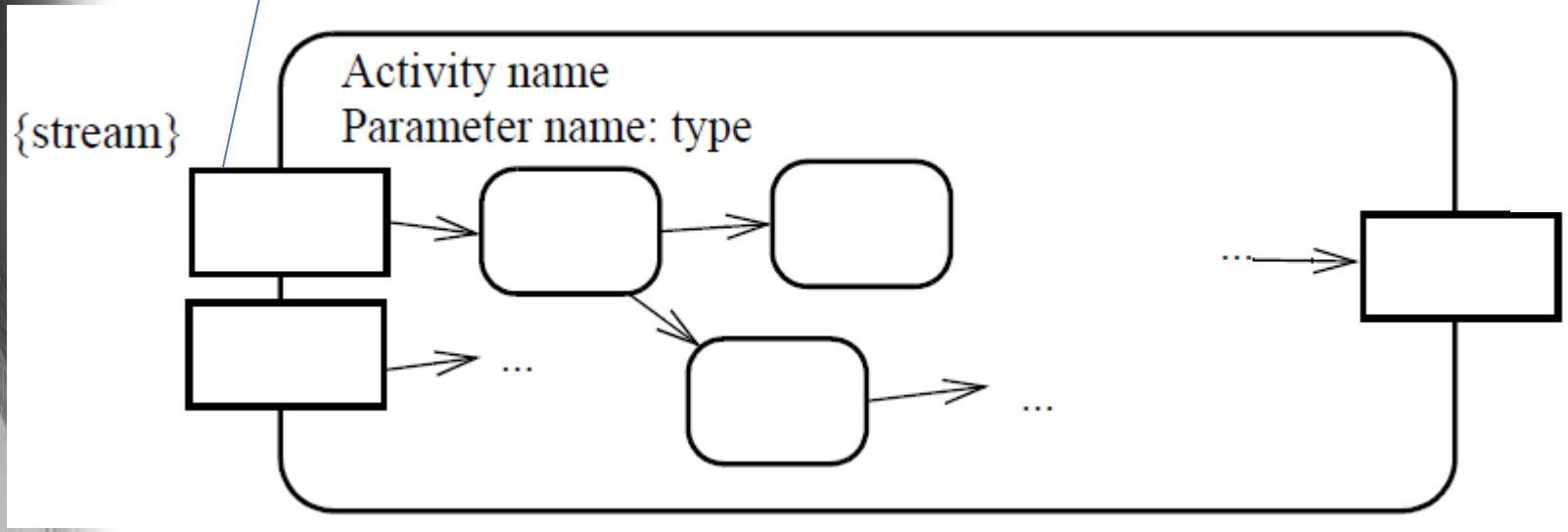
- Die einzelnen Datensätze sind dabei von beliebigem, aber festem Typ. Die Menge der Datensätze pro Zeiteinheit (Datenrate) kann variieren und evtl. so groß werden, dass die begrenzten Ressourcen zur Weiterverarbeitung nicht ausreichen und der Empfänger entsprechend reagieren muss (z. B. verwerfen von Datensätzen).

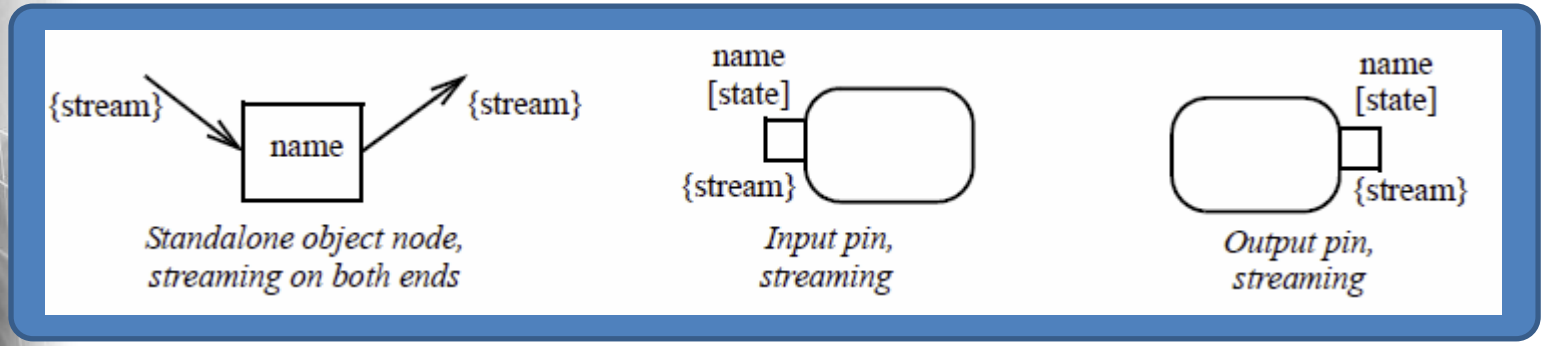
- Im Gegensatz zu anderen Datenquellen können Datenströme nur Satz um Satz fortlaufend verarbeitet werden - insbesondere ist im Gegensatz zu Datenstrukturen mit wahlfreiem Zugriff (wie z. B. Arrays) meist nur ein sequentieller Zugriff auf die einzelnen Datensätze möglich.



Allgemeine Notation

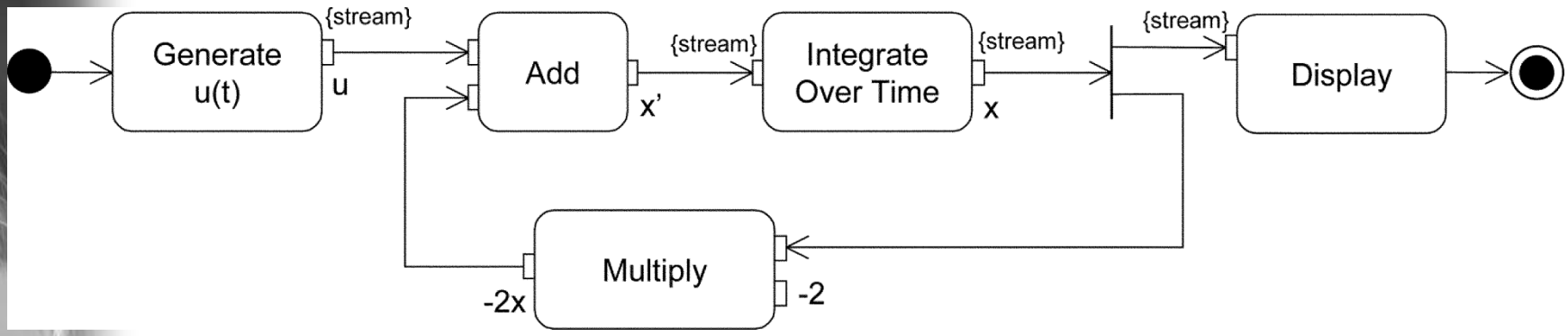
Datenstrom-Pin





Stream → Stetige Funktion ?

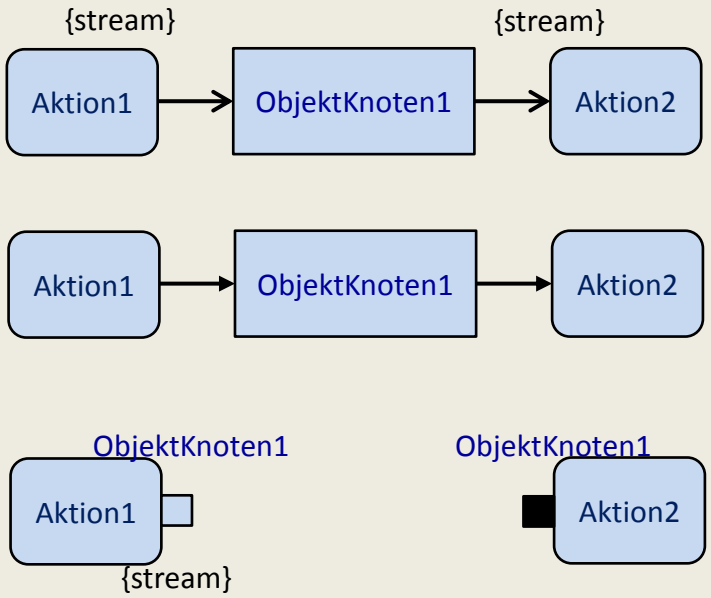
$$x'(t) = -2x(t) + u(t)$$



nicht im UML-Standard, Spezielle Erweiterung in UML ➔ SysML

Ein- und Ausgabeparameter für Datenströme

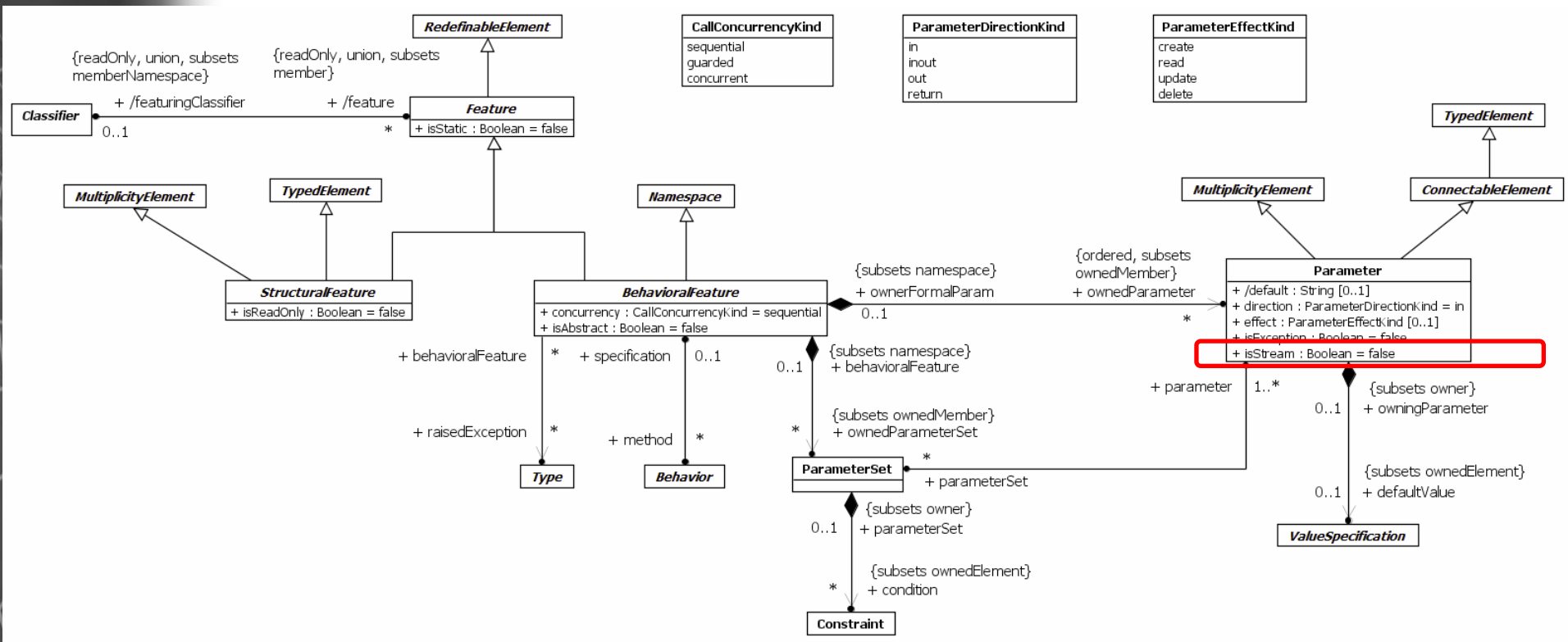
- **Gewöhnlich** nehmen Aktivitäten während sie laufen keine neuen Token an und bieten Ausgabetoken erst am Ende an, wenn sie abgeschlossen sind.
- **Datenstromeingabeparameter** erlauben eine Verarbeitung und Ausgabe von Daten-Token auch dann, wenn die Aktion noch aktiv ist.
- Bevor die Aktion ausgeführt werden kann, müssen jedoch **alle** aktuellen Parameter vorliegen, die **nicht** als Datenstromparameter gekennzeichnet sind



zulässige syntaktische Variationen

Ein- und Ausgabeparameter für Datenströme

isStream : Boolean [1..1] = false *tells whether an input parameter may accept values*



Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. **Diskussion offener Fragen**
6. Datenströme
7. Parametersätze: Gruppierung von Parametern
8. Behandlung von Ausnahmen
9. Beispiel
10. Weitere Aspekte (summarische Betrachtung)

Parametersätze

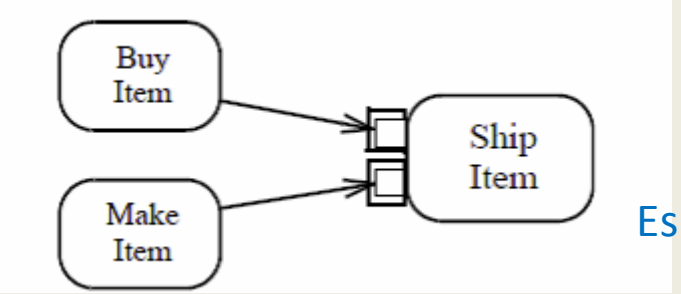
Es werden zwei Parametersätze ausgezeichnet

Gruppierung von Parametern – Parametersatz

- Nur ein ausgewählter Parametersatz ist jeweils für die Ausführung der Aktion relevant
- Spezifikation von alternativen, einander ausschließenden Gruppen von Ein- bzw. Ausgabewerten

Parametersätze: Parametergruppierungen

- Ein **Parametersatz** erlaubt die Zusammenfassung von Parametern, um alternative, einander ausschließende Gruppen von Ein- bzw. Ausgabeparametern spezifizieren zu können



Es gibt zwei Parametersätze

- Ein Parameter kann dabei **mehreren** Sätzen zugeordnet sein,
  wobei aber zwei Parametersätze aber nicht komplett gleich sein dürfen

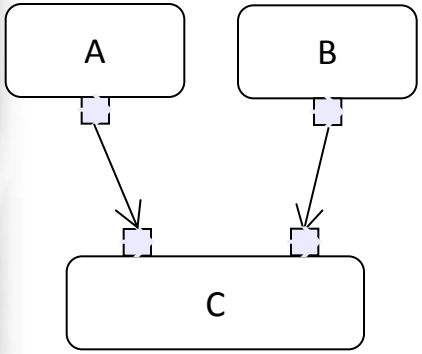
- Sobald ein Parametersatz existiert, müssen **alle einzeln** verbliebenen Parameter **Datenstromparameter** repräsentieren

- Pro Ausführung von Aktion/Aktivität werden immer nur die **Eingabeparameter eines** der alternativen Parametersätze verwendet.
  Das bedeutet, dass nicht an allen Pins Daten **anliegen müssen. Um zu Starten reicht es, wenn Datensätze an** allen Pins eines Parametersatzes anliegen

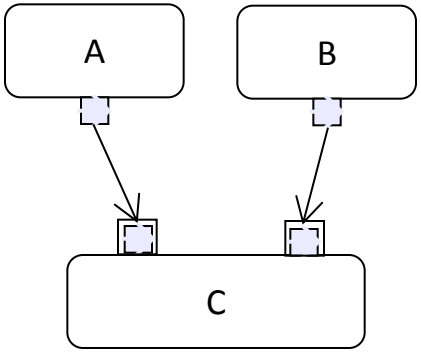
- **Ausgaben** können anlog nur für Werte für eines der alternativen Ausgabesets erzeugt werden. D.h. nach Beendigung der Aktion müssen nur an allen Pins eines der Parameterssätze Werte anliegen

Beispiel: Auszeichnung von Parametersätzen

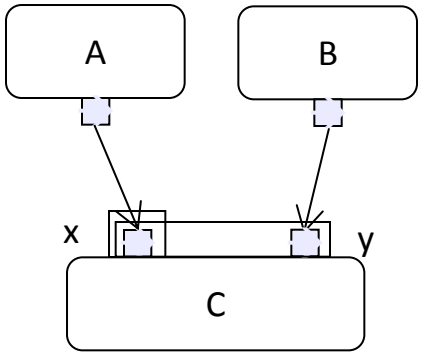
Problem: Unter welchen Bedingungen wird C ausgeführt ?



… nur dann,
wenn sowohl A als auch B Input für C liefern



… wenn entweder A oder B Input liefert

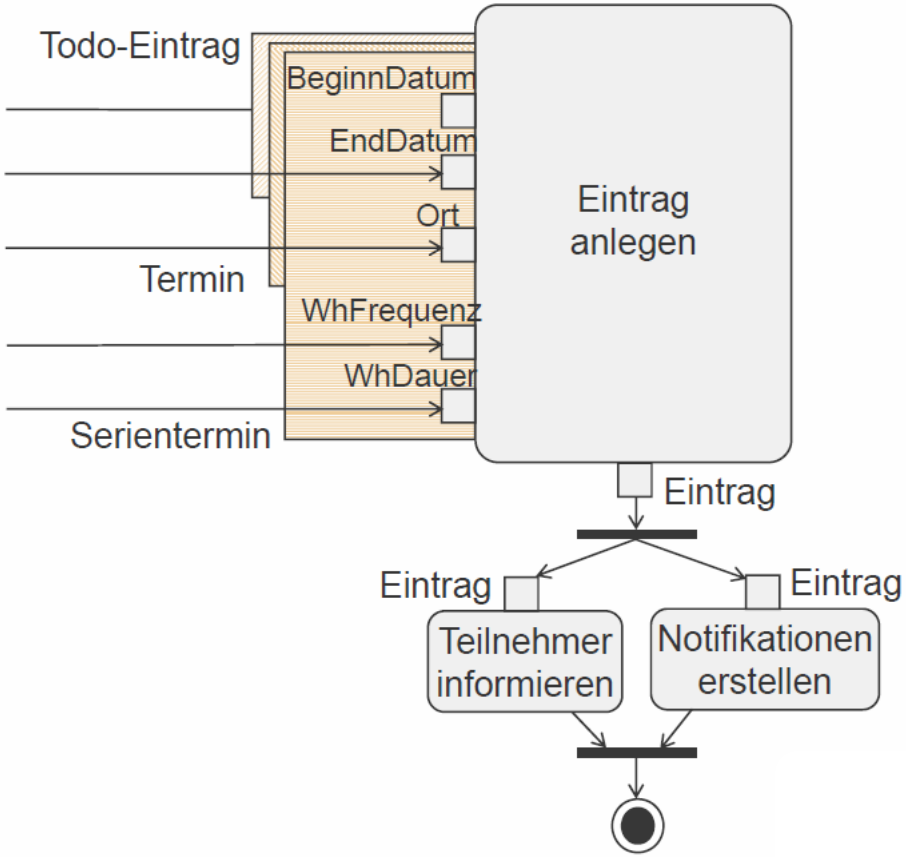


… von A mit x oder
von B mit x **und** y
Input liefert

Beispiel: Termineinträge anlegen

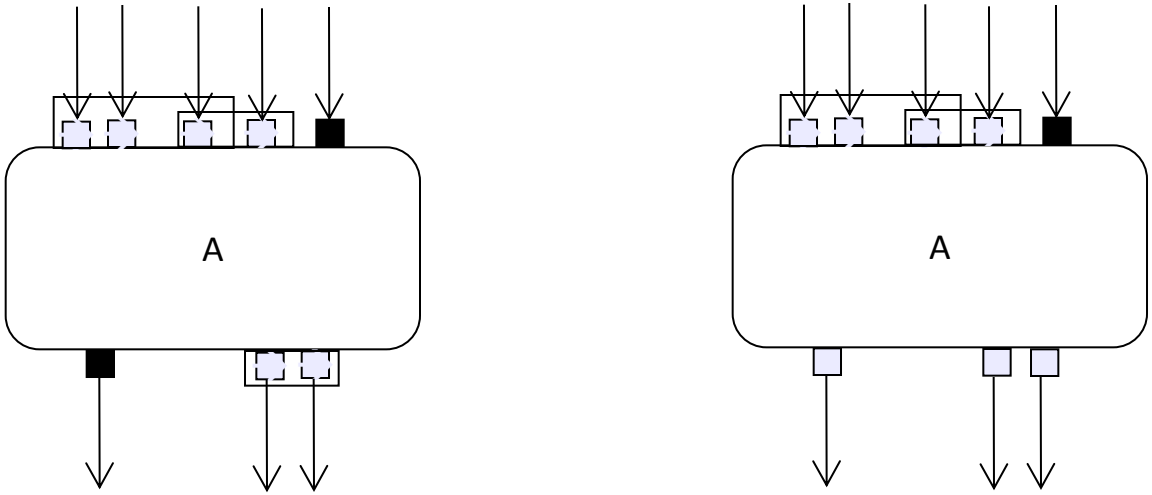
3 Terminarten:

- Todo-Eintrag
  - + BeginDatum
  - + EndDatum
- Termin
  - + BeginDatum
  - + EndDatum
  - + Ort
- Serientermin
  - + BeginDatum
  - + EndDatum
  - + Ort
  - + WhFrequenz
  - + WhDauer



Restriktionen für Parametersätze

Problem: Dürfen Parametersätze beliebig gebildet werden?



Besitzt eine Aktion Ein- bzw. Ausgangsparameter, die in einem Parametersatz enthalten sind, müssen alle weiteren Ein- bzw. Ausgangsparameter, die nicht zu einem Satz gehören, als **Stream**-Parameter definiert sein.

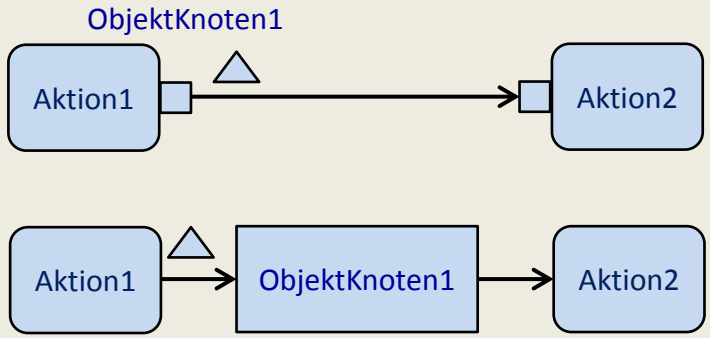
Zwei Parametersätze dürfen nicht die gleichen Parameter enthalten

Aktivitätsdiagramme

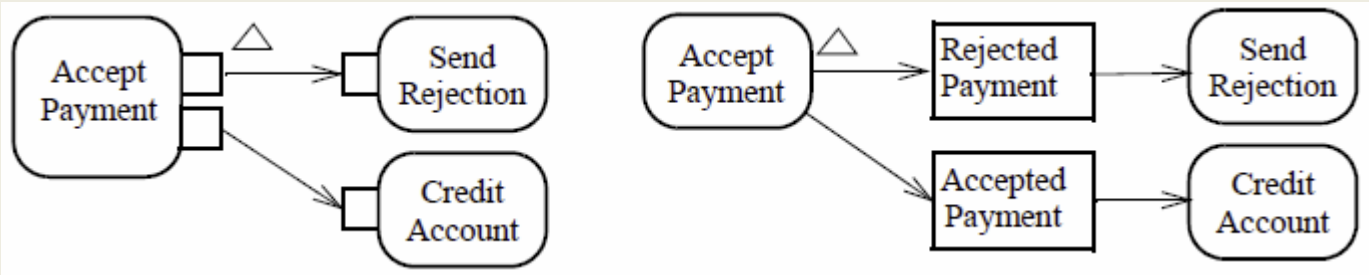
1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. **Diskussion offener Fragen**
6. Datenströme
7. Parametersätze: Gruppierung von Parametern
8. Behandlung von Ausnahmen
9. Beispiel
10. Weitere Aspekte (summarische Betrachtung)

Exceptions als Ausgabeparameter

- Ausgabeparameter könnnen als **Exceptions** gekennzeichnet werden
- Ausnahme für Aktionen: Die darauffolgende Aktion (ExceptionHandler) wird nur dann ausgeführt, falls das Auftreten einer Ausnahme durch das Vorhandensein eines solchen Ausnahmeparameters angezeigt wurde



- Ausnahmen für Aktivitäten: zusätzlich werden alle Abläufe einer Aktivität beendet



Exception-Handler

- Bearbeitet eine vordefinierte Ausnahme, die während der Ausführung einer Aktion auftritt

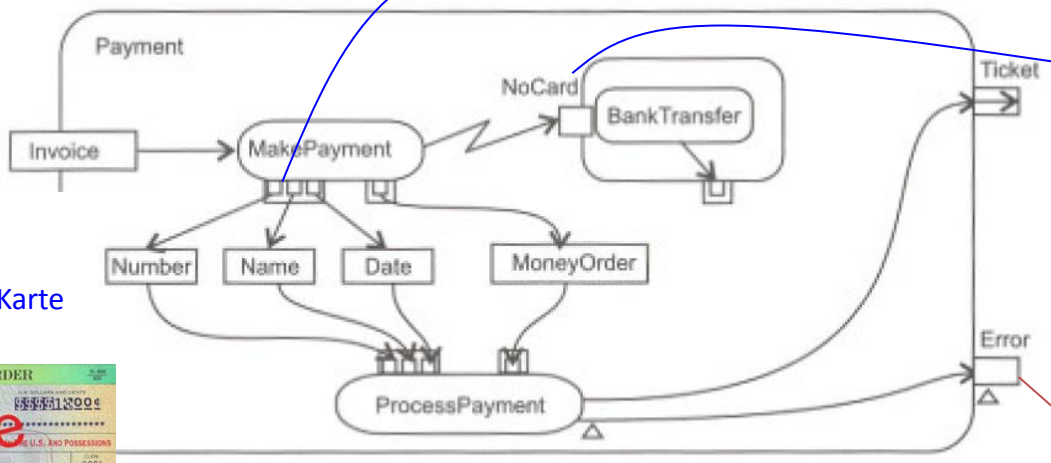
- Aktion und sein ExceptionHandler sind über “Blitz”-Kante verbunden, Exception-Typ und Eingabepin des ExceptionHandler müssen kompatibel sein

- ExceptionHandler übernimmt Ausführung.

  Ergebnis des ExceptionHandlers wird automatisch zum Ergebnis der Ursprungsaktion.

  Er **substituert** für eine gewisse Phase die Ursprungsaktion.
  Hat deshalb hat er keine eigenen Ein- und Ausgangskanten

Ausnahmebehandlung und Unterbrechung



Invoice-Attribute

NO-Card-Unterbrechung:
Behandlung ist *Banktransfer*
dieser stellt *MakePayment*
Statt der Karte die Geldmenge
zur Verfügung
FORTSETZUNG nach MakePayment

MoneyOrder
Ausstellung mit
bestimmter debit-Karte
ist Normalfall

ProcessPayment
könnte auch Ausnahme werfen

Ausnahmeparameter
(von *ProcessPayment*) zur Info
an die Umgebung

- Ausnahmen und Unterbrechungen sind Abweichungen vom normalen Kontrollfluss einer Aktivität
- Ausnahmen entstehen durch Fehler in der Verhaltensausführung oder werden explizit geworfen: **raise**-Aktion
- eine parametrisierte Aktivität kann seinen Aufrufer über das Auftreten einer Ausnahme informieren
- Unterbrechungen werden durch „**Interrupts**“ ausgelöst (Definiert für Aktivitätsbereiche)

Unterbrechungsbereich

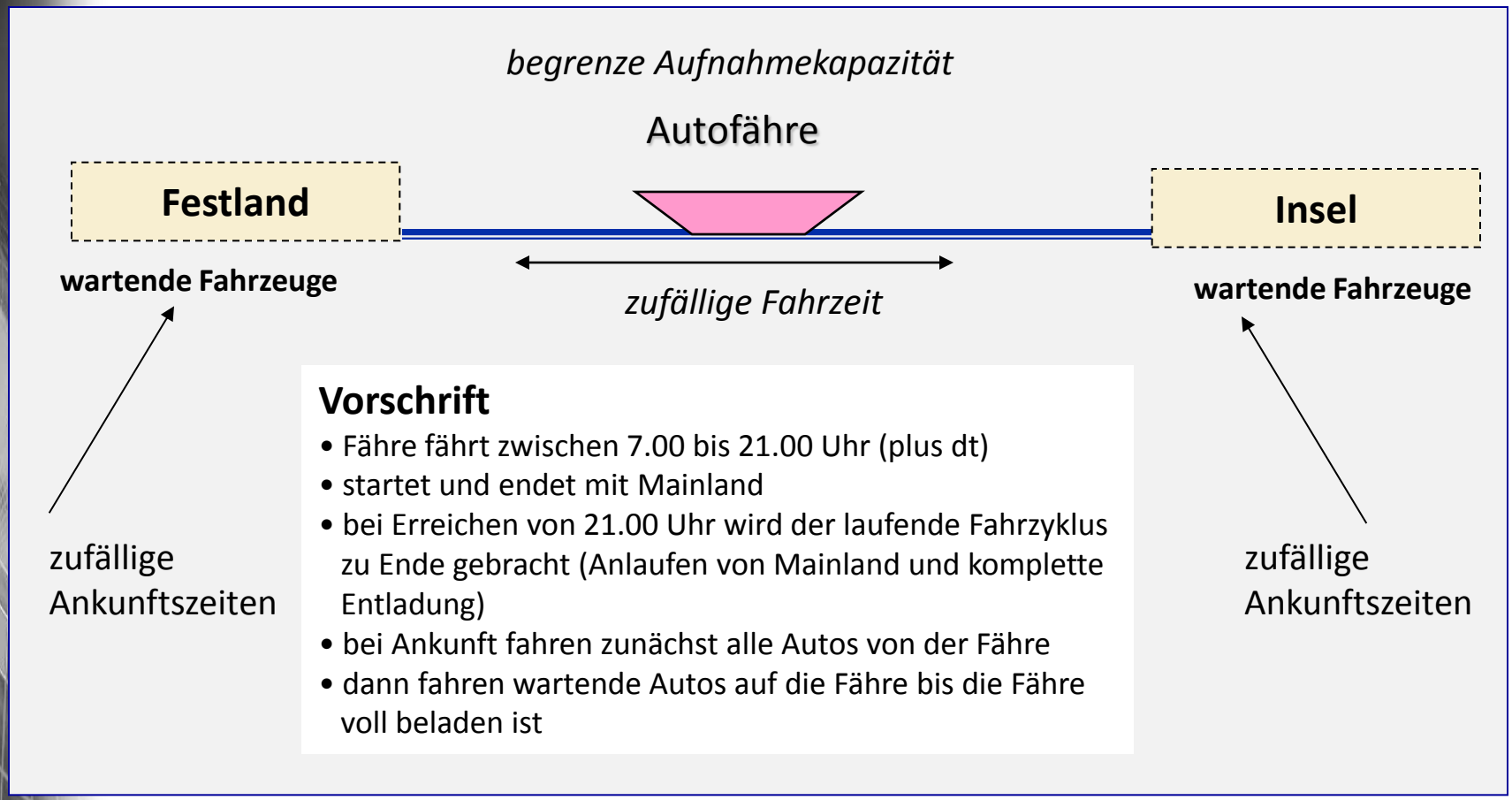
- umschließt eine oder mehrere Aktionen/Aktivitäten
- Neben den “normalen” Kanten gibt es eine Unterbrechungskante
- Wird der Bereich über diese Kante verlassen, werden
  - sämtliche Abläufe abgebrochen,
  - alle vorhandenen Token verworfen und
  - es wird mit dem Zielknoten der Unterbrechungskante fortgesetzt

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. **Diskussion offener Fragen**
6. Datenströme
7. Parametersätze: Gruppierung von Parametern
8. Behandlung von Ausnahmen
9. Beispiel: Fähre
10. Weitere Aspekte (summarische Betrachtung)

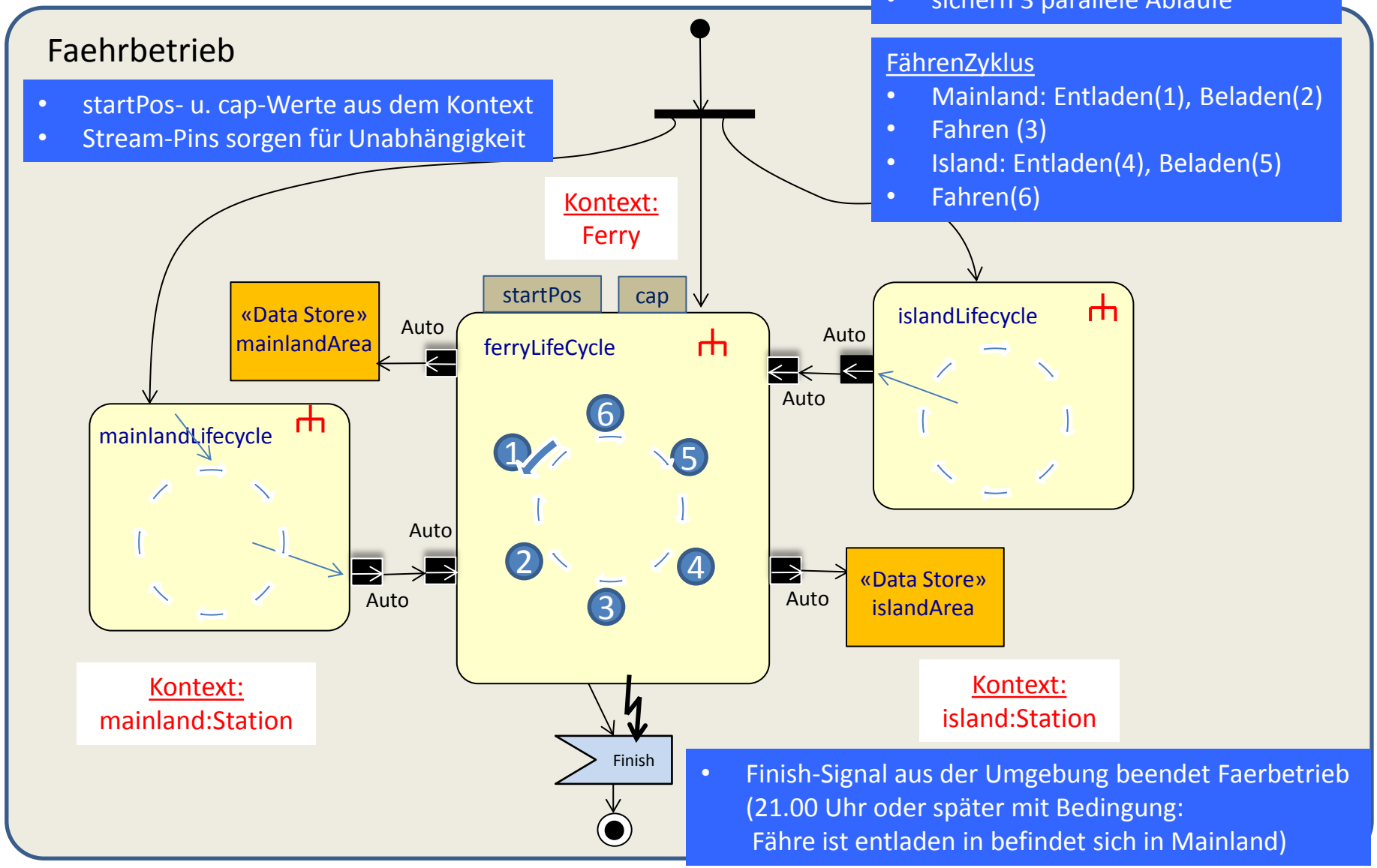
Beispiel: Autofähre (Variante-1)

- Ziel: Darstellung als Aktivitätsmodell in UML

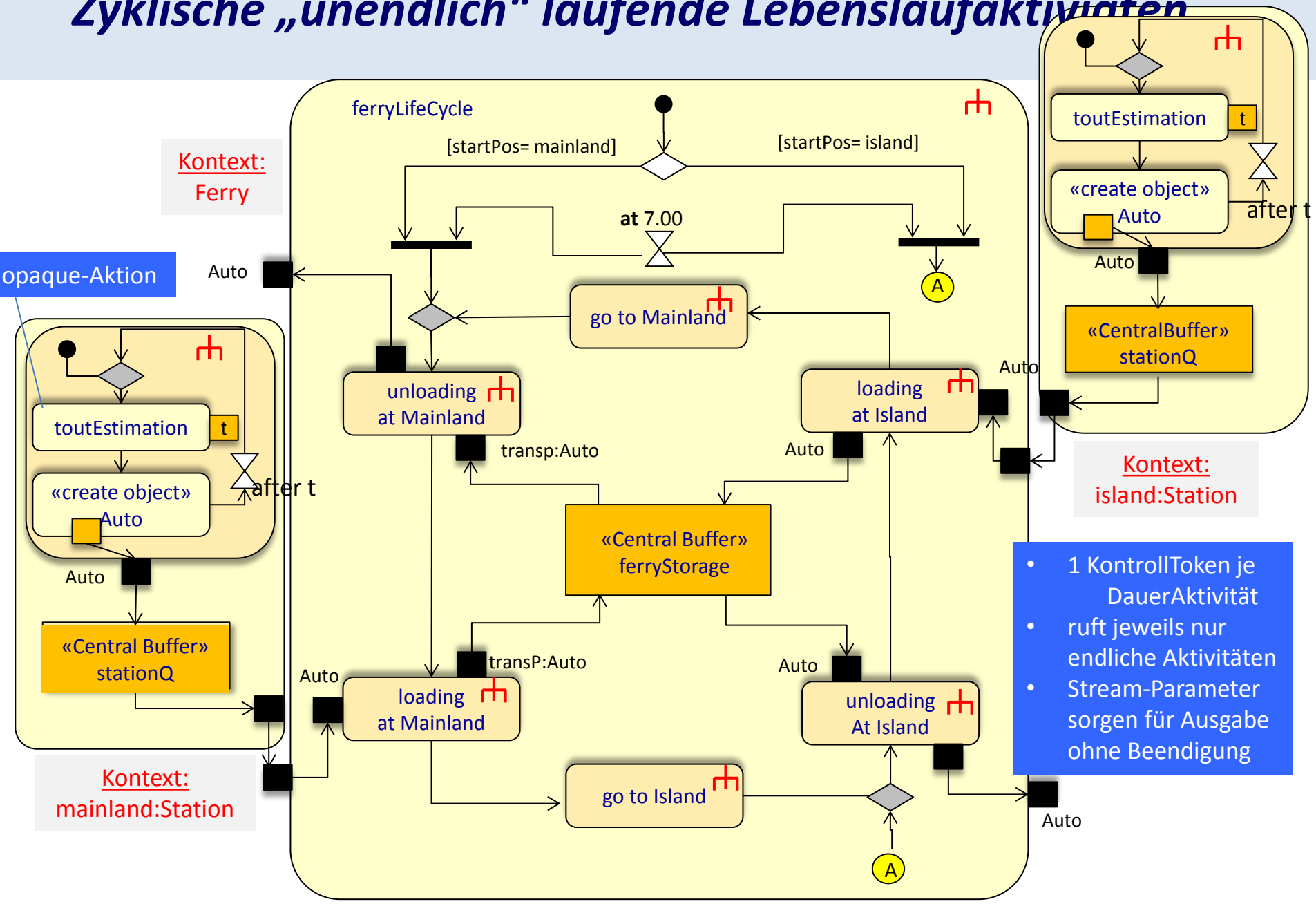


Systemverhalten: *drei strukturierte Dauer-Aktivitäten*

- 3 KontrollToken
- sichern 3 parallele Abläufe

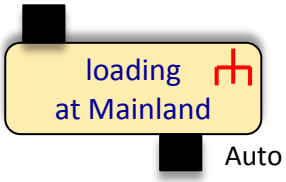


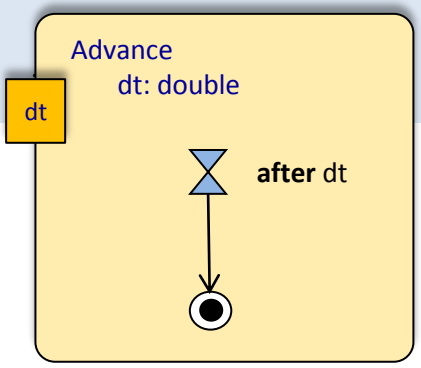
Zyklische „unendlich“ laufende Lebenslaufaktivitäten

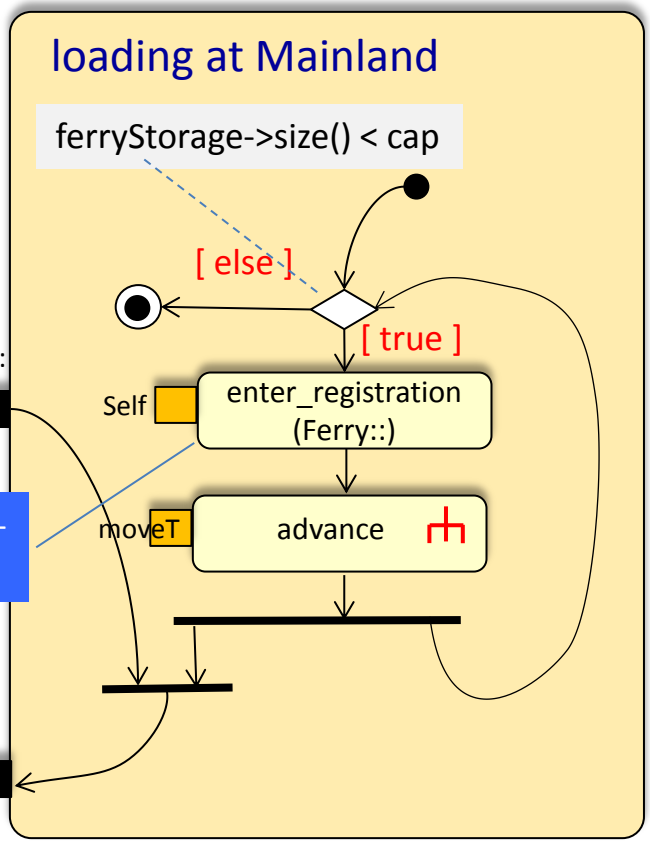


Elementare endlich laufende Aktivitäten

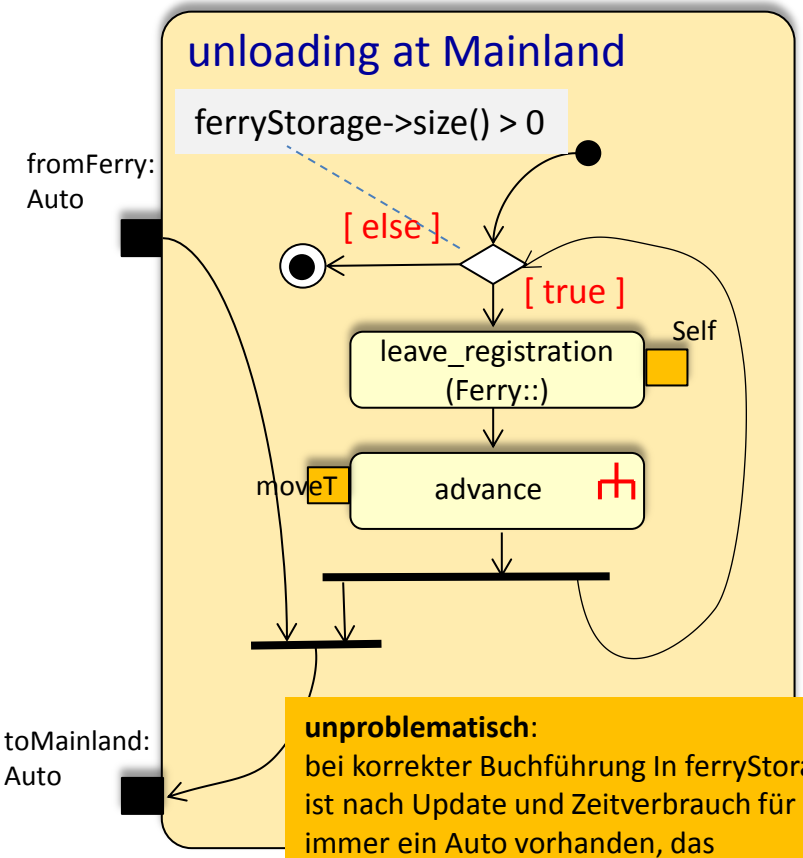
problematisch:
- Registrierung eines Autos und Zeitverbrauch für Beladung erfolgen unabhängig, ob ein Auto bereits wartet oder später (u.U. garnicht) eintrifft







callOperation-Aktion

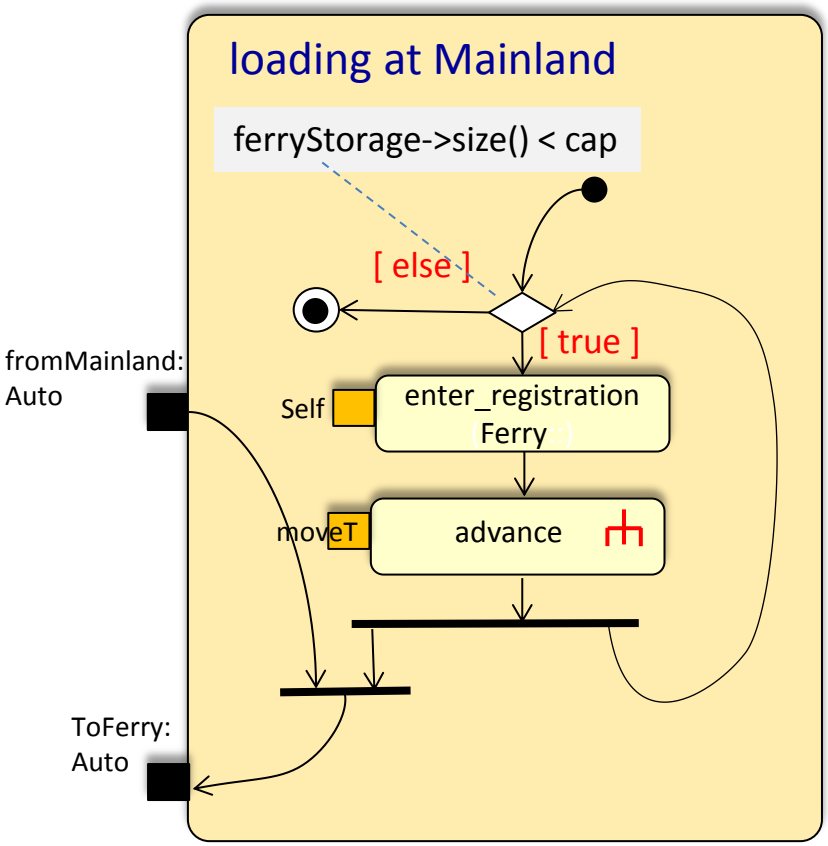


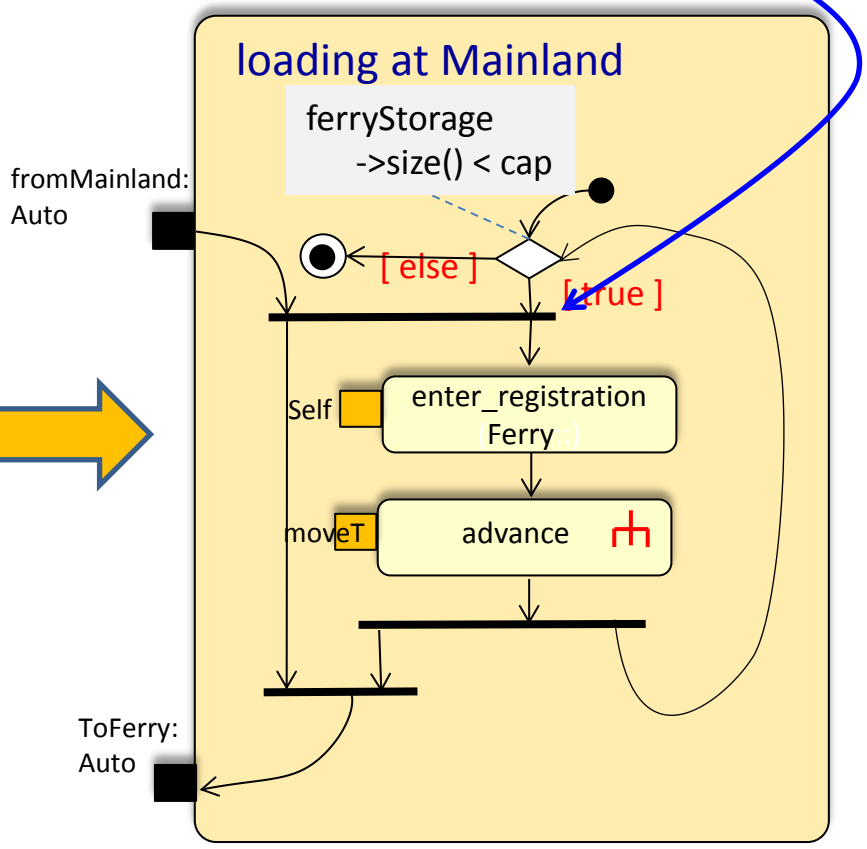
unproblematisch:
bei korrekter Buchführung In ferryStorage ist nach Update und Zeitverbrauch für Entladung immer ein Auto vorhanden, das tatsächlich (als ObjektToken) bewegt werden kann

Loading at …: Korrektur der problematischen Lösung

problematisch:
- Registrierung eines Autos und Zeitverbrauch für Beladung erfolgen unabhängig, ob ein Auto bereits wartet oder später (u.U. garnicht eintrifft)

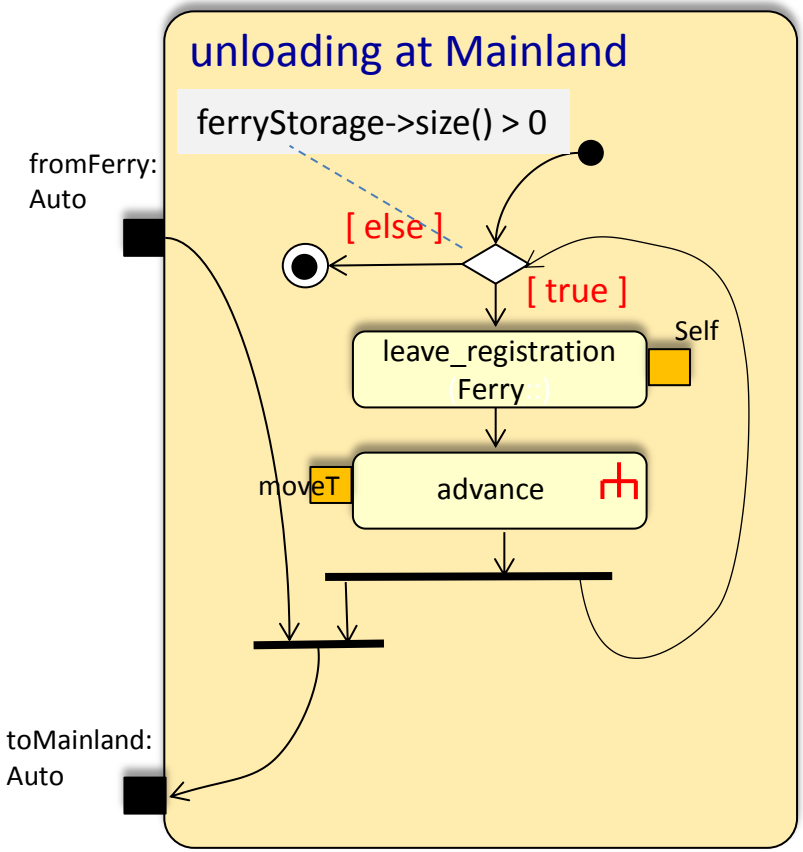
<u>geändert:</u>
Vor Registrierung eines Autos und Zeitverbrauch für Beladung wird solange gewartet, bis ein Objekt-Token Auto am Eingangspin anliegt

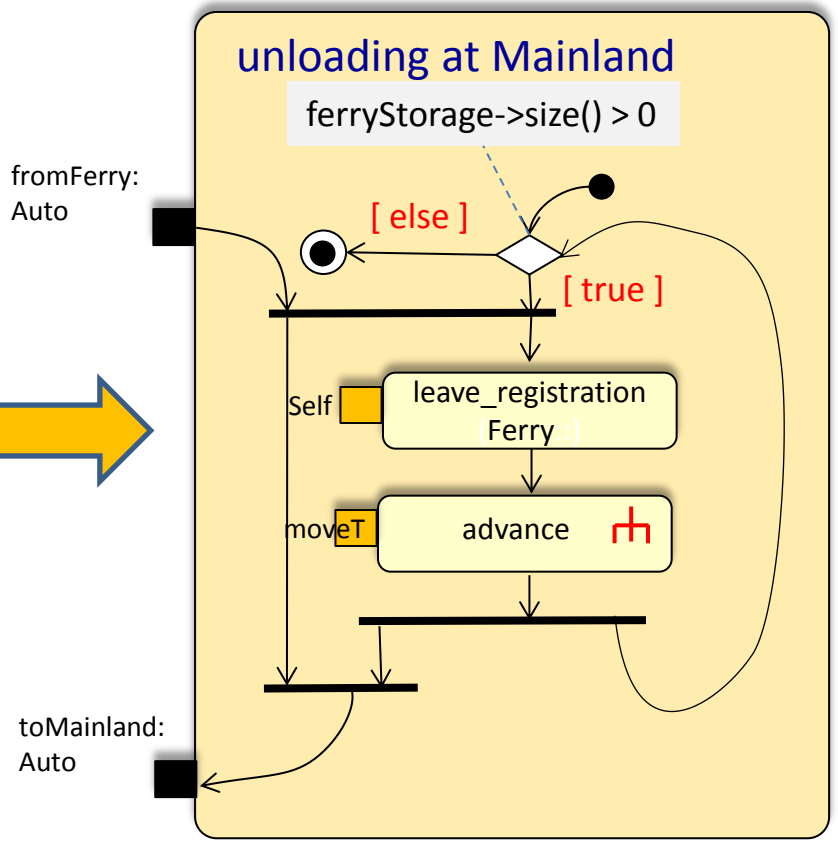




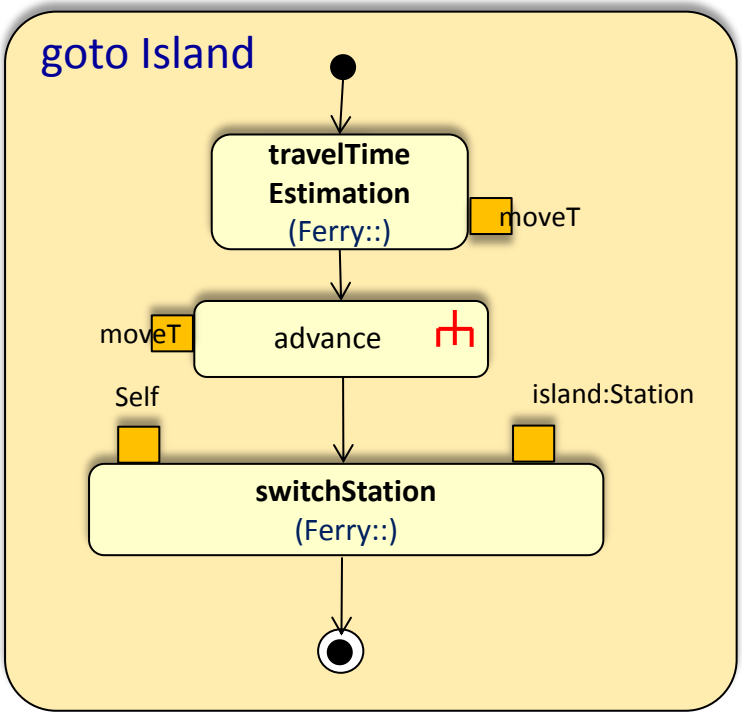
Unloading at …: Korrektur der unproblematischen Lösung

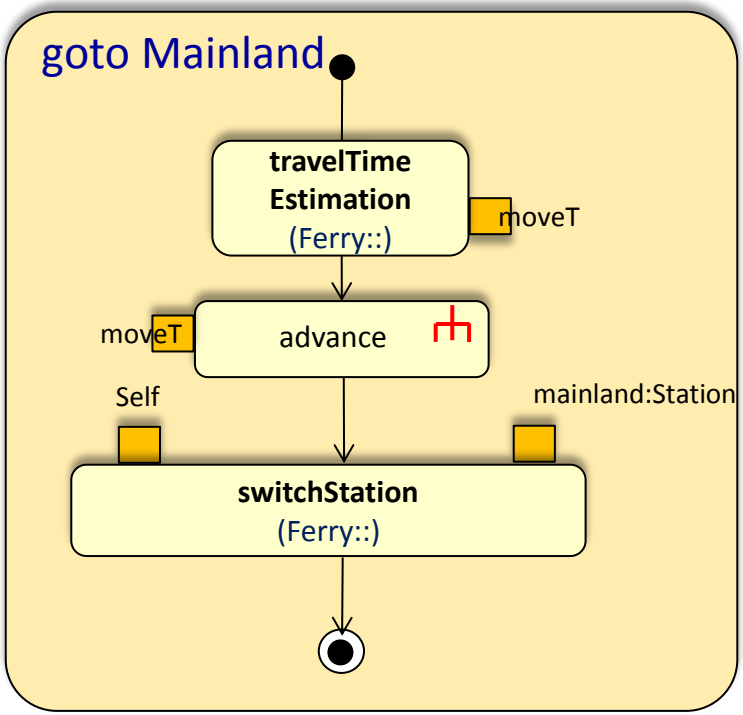
Analoge Änderung zu loading at Mainland verbessert konzeptionelle Klarheit

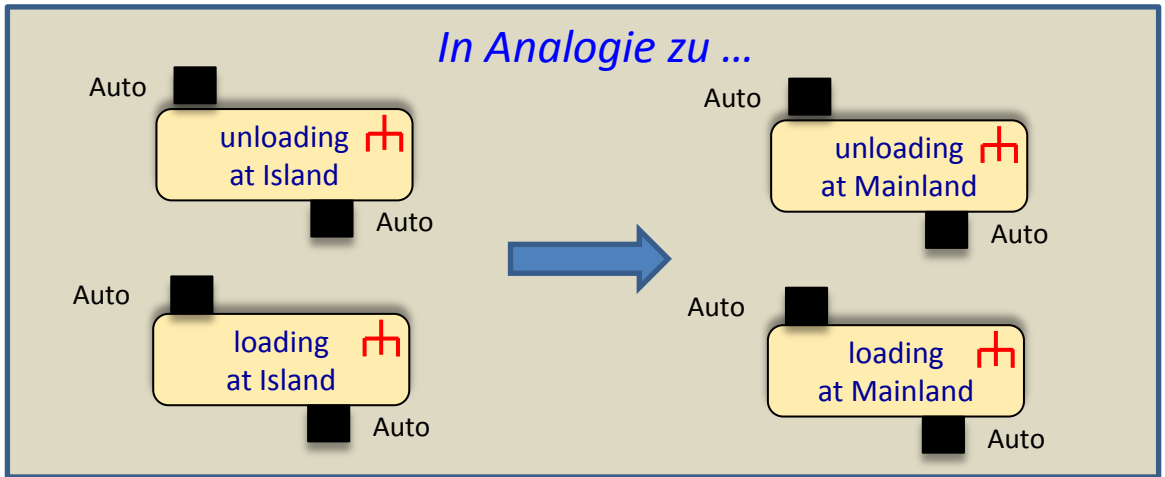




Modellierung weiterer Aktivitäten

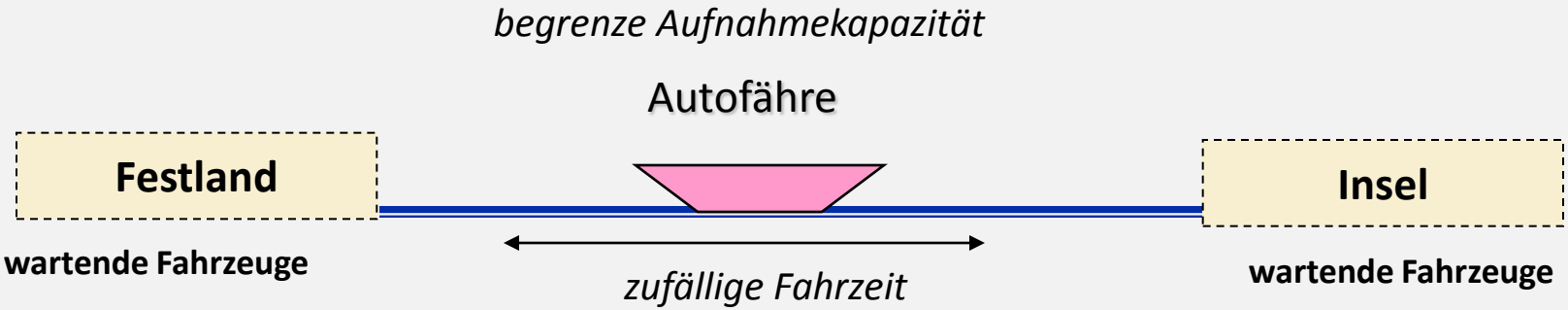






Beispiel: Autofähre (Variante-2)

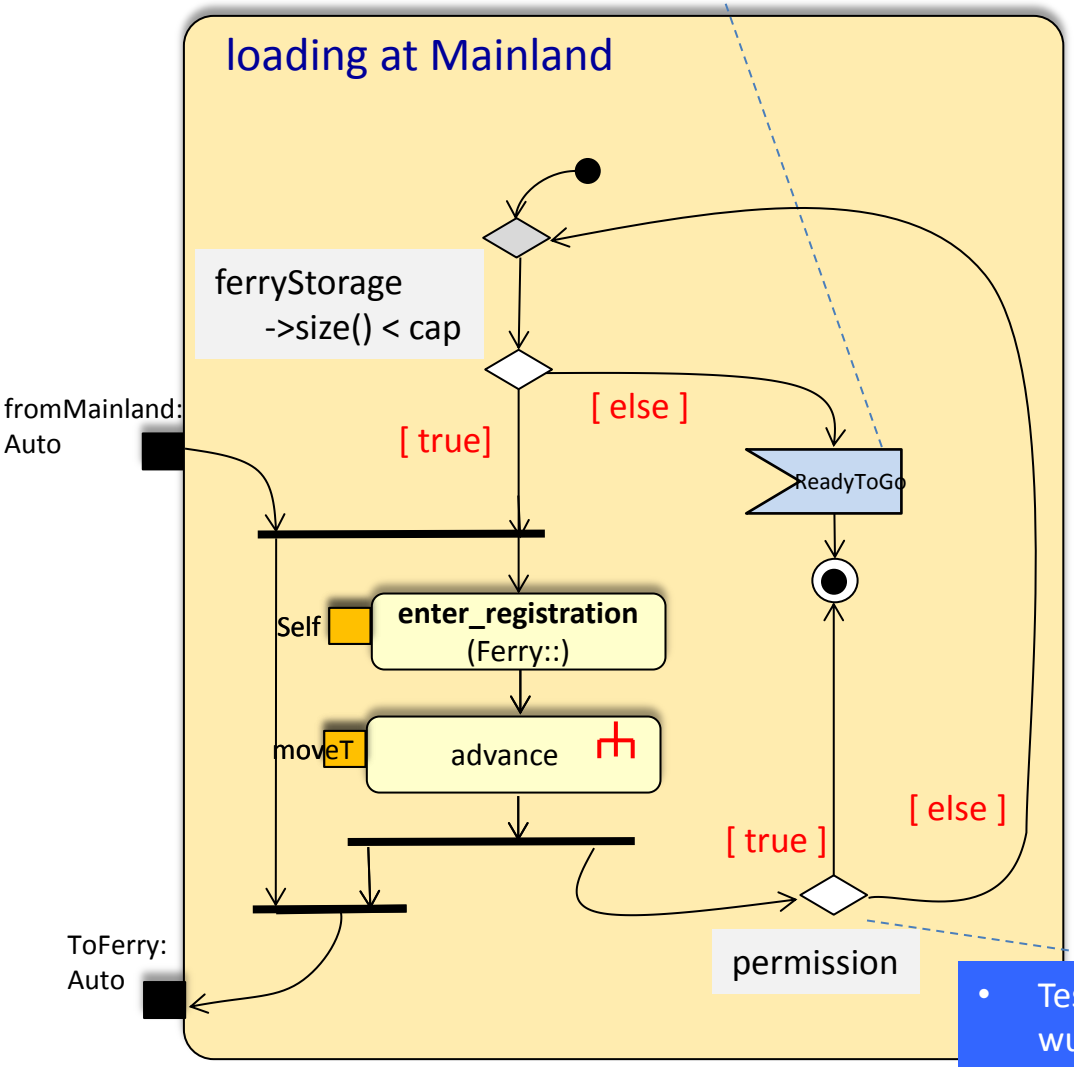
- Ziel: Darstellung als Aktivitätsmodell in UML

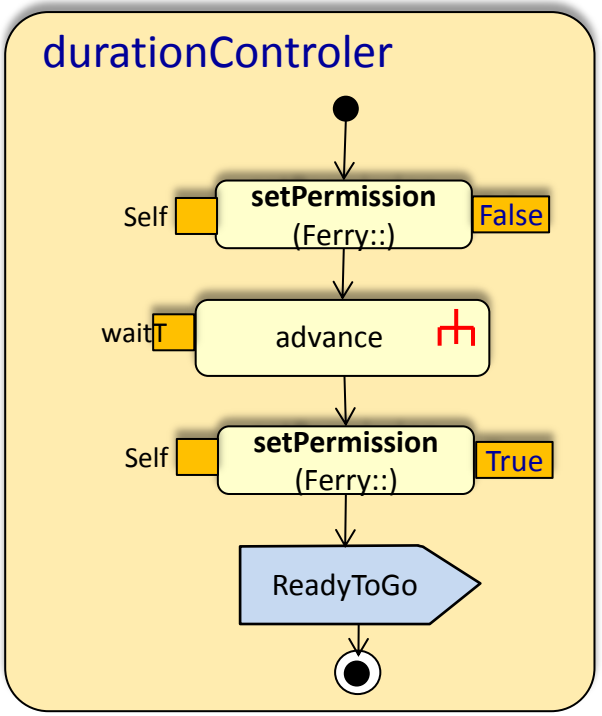


Vorschrift

- Fähre fährt zwischen 7.00 bis 21.00 Uhr (evtl. plus dt)
- startet und endet mit Mainland
- bei Erreichen von 21.00 Uhr wird der laufende Fahrzyklus zu Ende gebracht (dt ~ Anlaufen von Mainland und komplette Entladung)
- bei Ankunft fahren zunächst alle Autos von der Fähre
- Vorgabe einer Richtzeit für Verweildauer an einer Station (plus delta)
  delta: Komplettierung der evtl. laufenden Beladung
  a) obwohl noch Platz auf der Fähre und weitere Autos warten, legt die Fähre ab
  b) obwohl Fähre bereits voll beladen ist, wartet sie ab

- Warten bei voller Fähre
  auf Ablegesignal (was nach einer best. Zeit garaniert kommt)
  Aktivität muss dazu mit Beginn von *unloading at Mainland* gestartet werden (als Starttransition)





- Test, ob Ablegesignal bereits gegeben wurde, ohne darauf zu warten
  permission ~ Attribut im Kontext

Weitere Präzisierungen Erweiterungen

Fragestellungen

- Stellen Sie über eine Abbruchaktivität sicher, dass das Finish-Signal genau dann an den Fährbetrieb gesendet wird, wenn
  a) es 21.00 Uhr oder später ist und
  b) die Fähre in Mainland angelegt hat und
  c) alle transportierten Autos die Fähre verlassen haben

- Wie müssen die Klassendiagramme von Station, Fähre, usw. gestaltet werden, um die erwarteten Kontexte zu sichern?

- Was ist zu tun, um
  Unloading, Loading, GoTo mit der jeweiligen Station zu parametrisieren ?

- Was ist zu tun, um einen Fährberieb mit einer variablen Anzahl von Fähren zu realisieren?

- Wodurch ist die Zeitdauer einer Fähre für einen Fährzyklus bestimmt? Nur durch die Advance-Aktivitäten?