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

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

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
4. Aktionen
5. Objektfluss-Steuerung

Letzte Vorlesung

jetzt mit Präzisierungen

Aktivität, Aktion, Pin

Definition:

- Eine **Aktivität** (*activity*) beschreibt Abläufe,
  die aus mehreren elementaren **Aktionen** bestehen.

- Ein Ablauf kann
  - parallelisiert und synchronisiert sowie
  - auf der Basis von Bedingungen geteilt und wieder zusammengeführt

  werden.

- Eine **Aktion** (*action*) ist ein elementarer, ausführbarer Schritt in einer Aktivität.

Der **Pin** ist das Bindeglied zwischen den (formalen) Parametern einer Aktion und dem Objektfluss (aktuelle Parameter).
Es wird zwischen Ein- und Ausgabe-Pins unterschieden (wie bei der Aktivität auch)

Konkrete Syntax einer Aktion i.allg. wie die einer Aktivität (aber nicht weiter zerlegbar), es gibt aber auch Sonderformen: SendSignalAction, …

Eine größere Menge **vordefinierter** Aktionen in UML: CallBehaviourAction, …

Allgemeine Charakterisierung

7 Aspekte

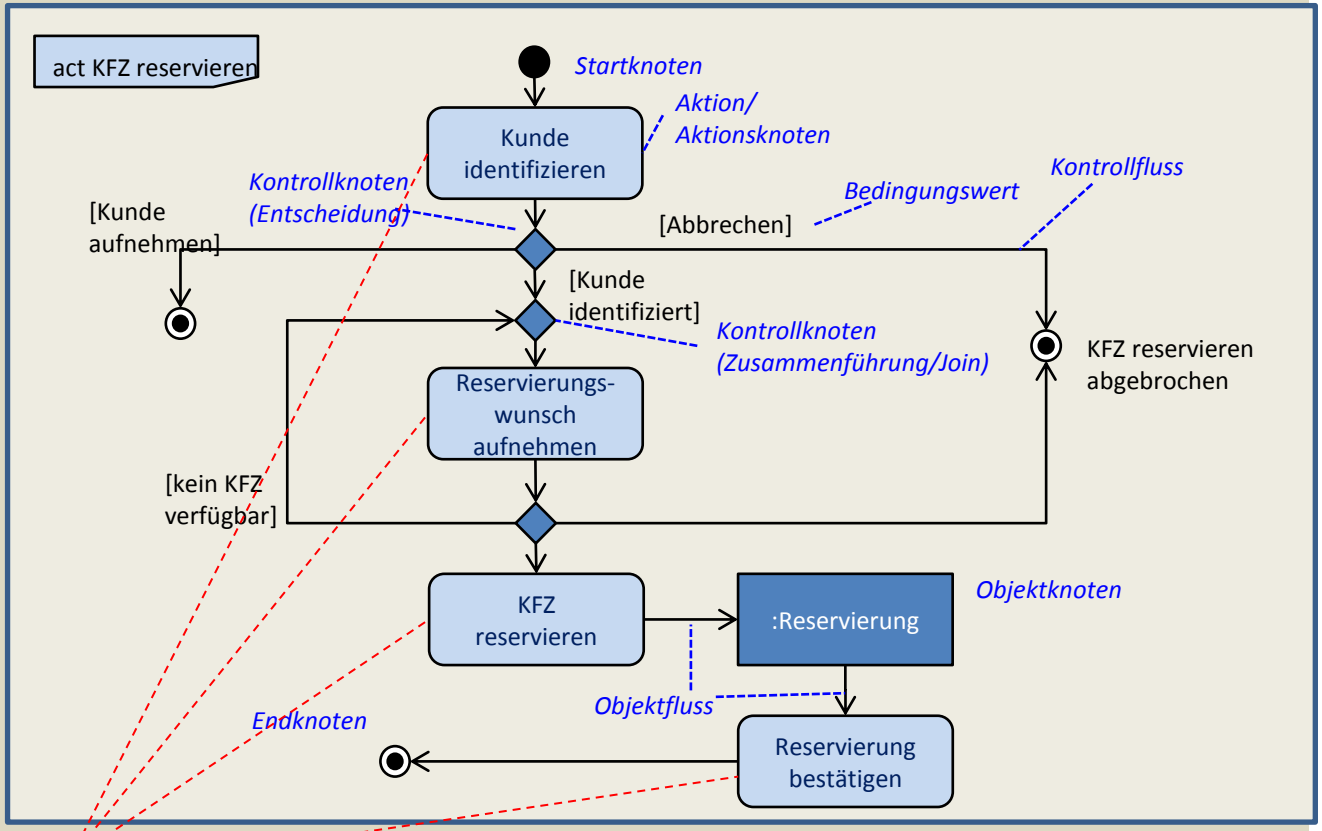
(1) Spielart der Verhaltensbeschreibung in UML

(2) Kontext

(3) Struktureller Aufbau

(4) Namensraum

(5) Aktionsknoten, Objektknoten, Steuerungsknoten



Elementare Aktion (nicht weiter formal beschrieben, dekomponiert)

Allgemeine Charakterisierung (Wdh.)

7 Aspekte

(1) Spielart der Verhaltensbeschreibung in UML

(2) Kontext

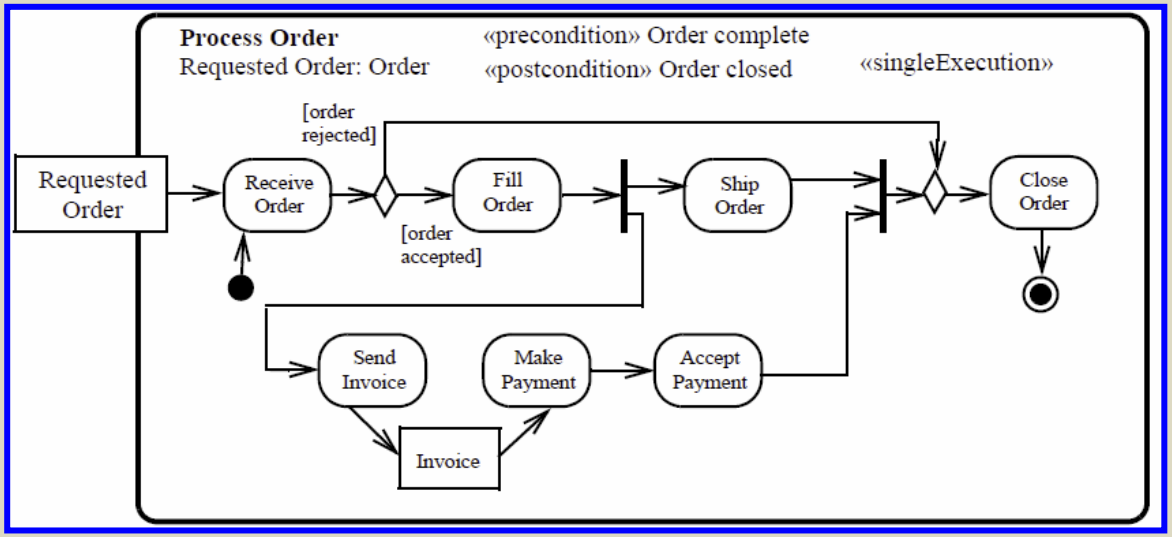
(3) Struktureller Aufbau

(4) Namensraum

(5) Aktionsknoten, Objektknoten, Steuerungsknoten

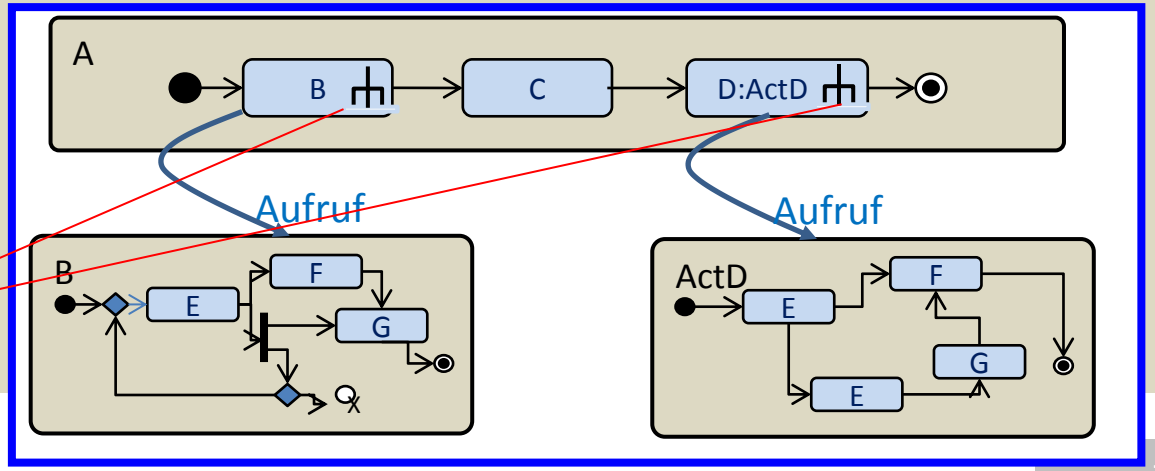
(6) Parameter und Ausführungsbedingungen

(7) Darstellungskraft: Dekomposition, Konnektoren

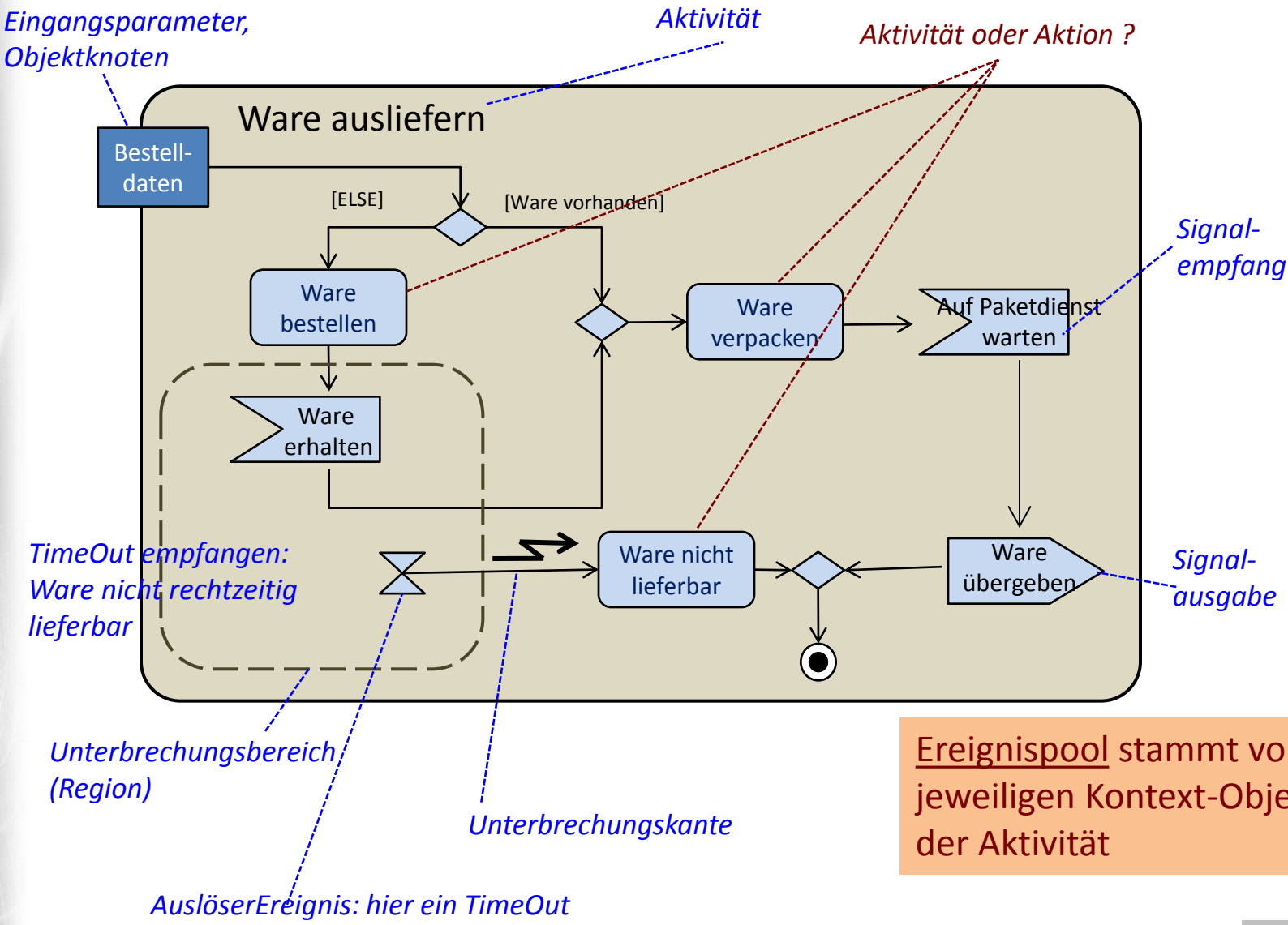


1.Änderung

CallBehaviourAction)



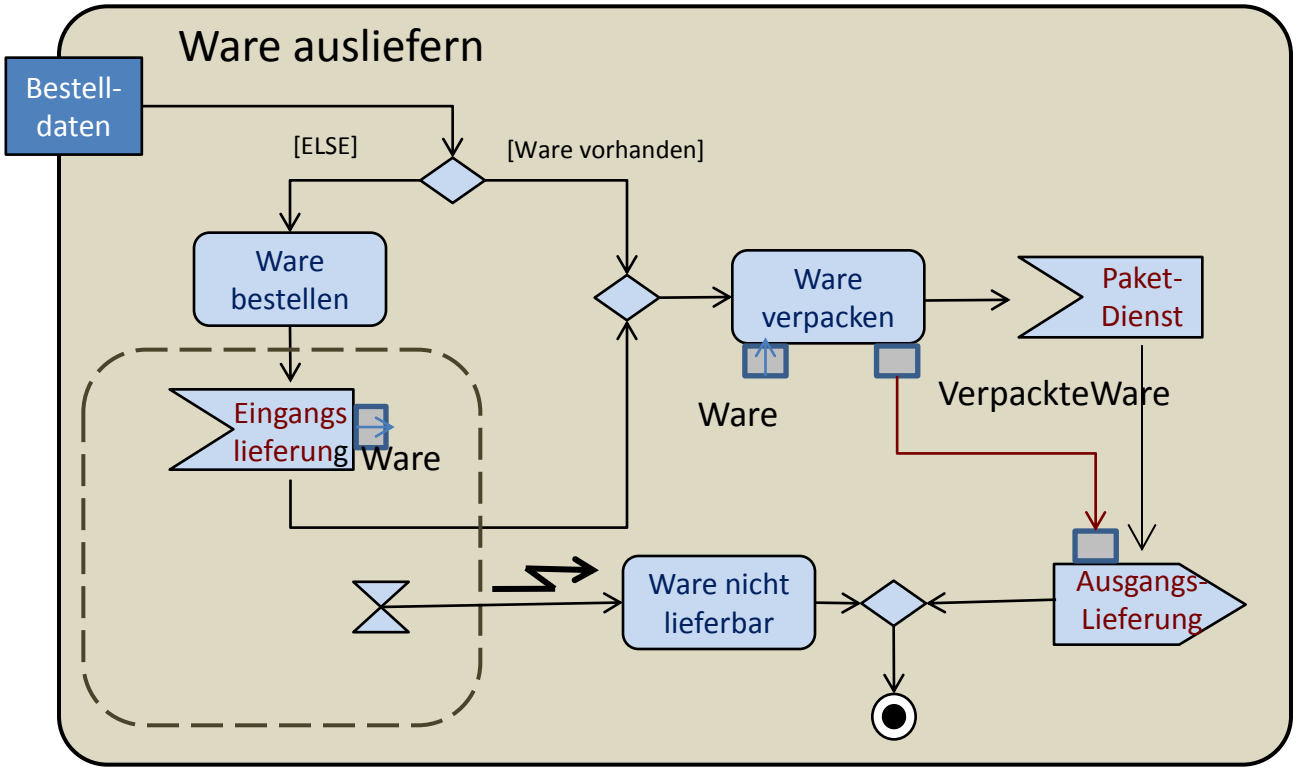
Beispiel: komplexere Aktivität



Ereignispool stammt vom jeweiligen Kontext-Objekt der Aktivität

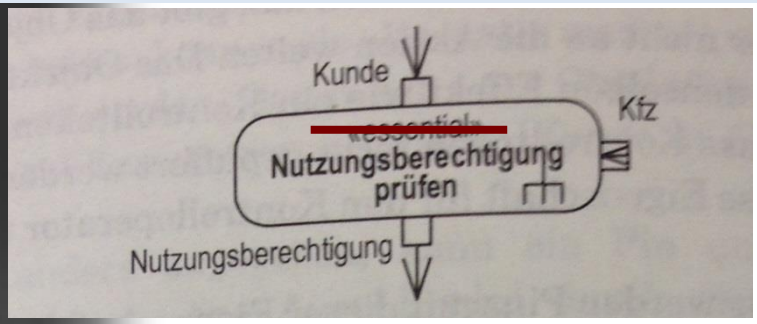
Beispiel nochmal: partielle Verbesserung

2.Änderung



- *Namenswahl für Signale*
- *Einführung von Aktions-Pins*

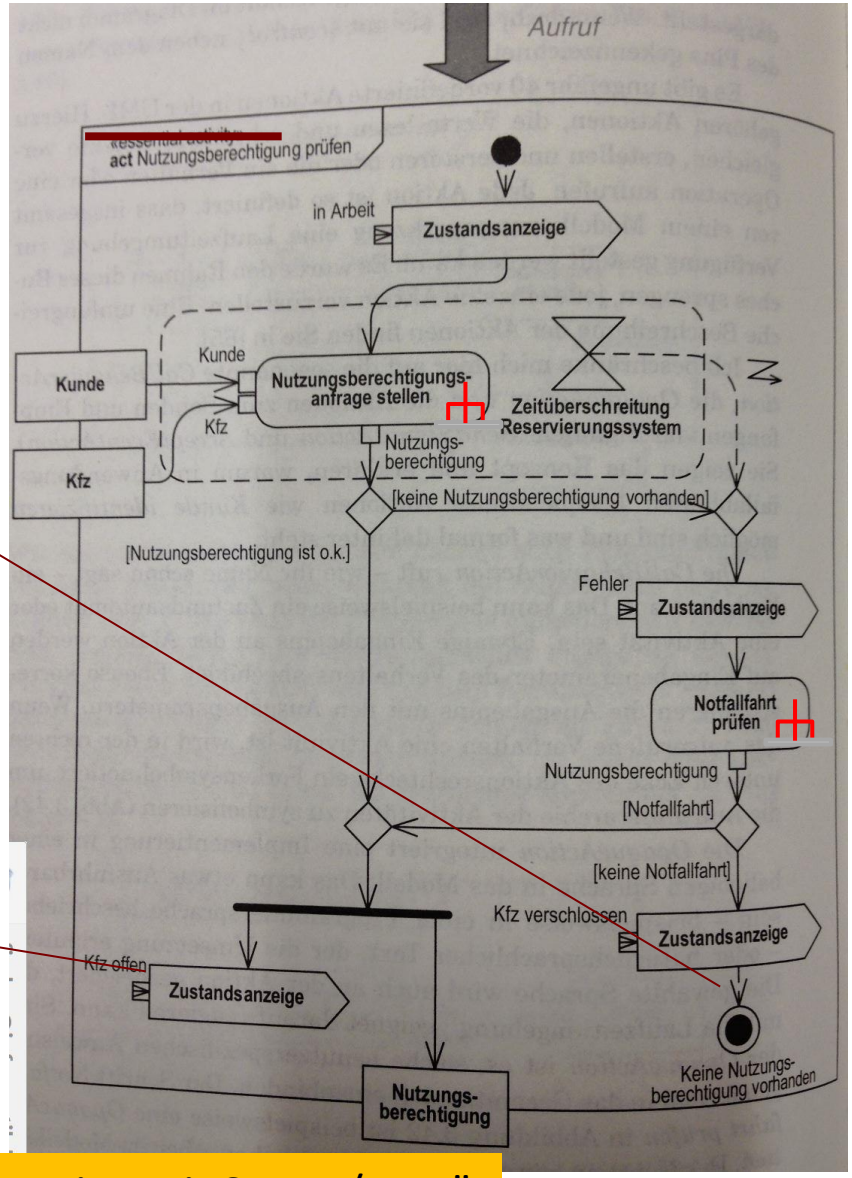
Weiteres Beispiel: Diskussion



Kein Ausgabeparameter
oder NULL-Token?

Regel: bei Stop werden
alle nicht gesetzten Ausgabeparameter
mit NULL-Token versorgt

zunächst unklar:
Impliziter Stop oder Verklemmung?
→ führen sicherheitshalber einen
expliziten Stop ein



aus: T. Weilkiens: “SystemsEngineering mit SysML/UML”

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
   - Objektknoten allgemein
   - Parameter von Aktivitäten und Aktionen, Werte-Pin (Konstante)
   - Pufferknoten

Objektknoten: Grundfunktionalität

- ObjektKnoten verwalten **Objekt-Token**, die ihrerseits **Referenzen(?)** auf Objekte besitzen (Sprechweise im UML-Standard:
  Objekt-Token sind Container von Objekten

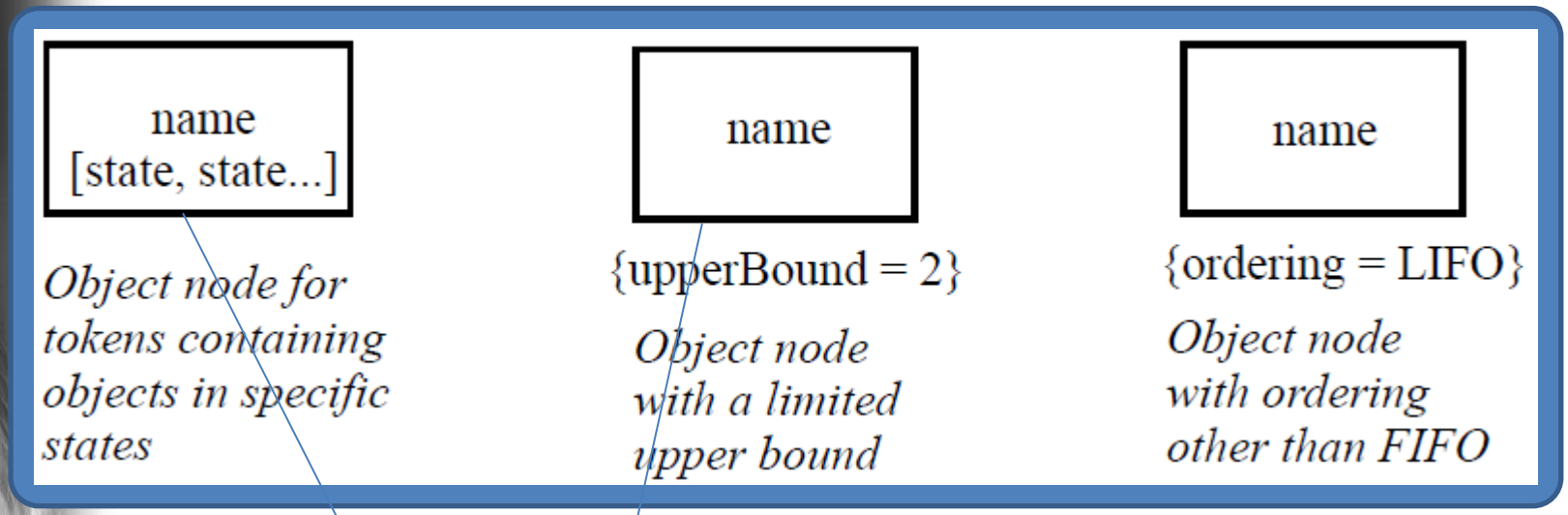
Präzise Aussagen dazu fehlen (noch)

Wir gehen aber weiterhin von einer Referenzsemantik aus

- Sie sind **ausschließlich über Objektflüsse** miteinander verbunden:
  → Objekt-Token werden ausgetauscht zwischen
  a) zwei ObjektKnoten oder
  b) einem ObjektKnoten und einer Aktion über deren Pin
- Objekt-Token werden dabei durch **vorangegangene Aktionen** erzeugt und modifiziert
- **Typ-Angabe** ist optional.
  Aber zulässige Daten sind nur von dem angegebenen Typ oder einem Untertyp.
- ObjektKnoten können auf einen bestimmten **Zustand beschränkt** sein **[…]**.
  D.h., nur DatenToken mit dieser Beschränkung werden aufgenommen.
- ObjektKnoten **ohne Typangabe und Zustandsbeschränkung** können beliebige Daten und Objekte aufnehmen (~ void*)
- **Unterscheidung** von ObjektKnoten nach Art der
  – Aufnahme, Verarbeitung, Speicherung und Weitergabe

Annotationen von Objektknoten

name ~ üblicherweise: *Typbezeichner, kann aber auch freier Name sein*

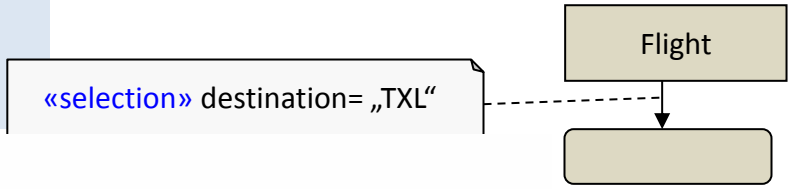


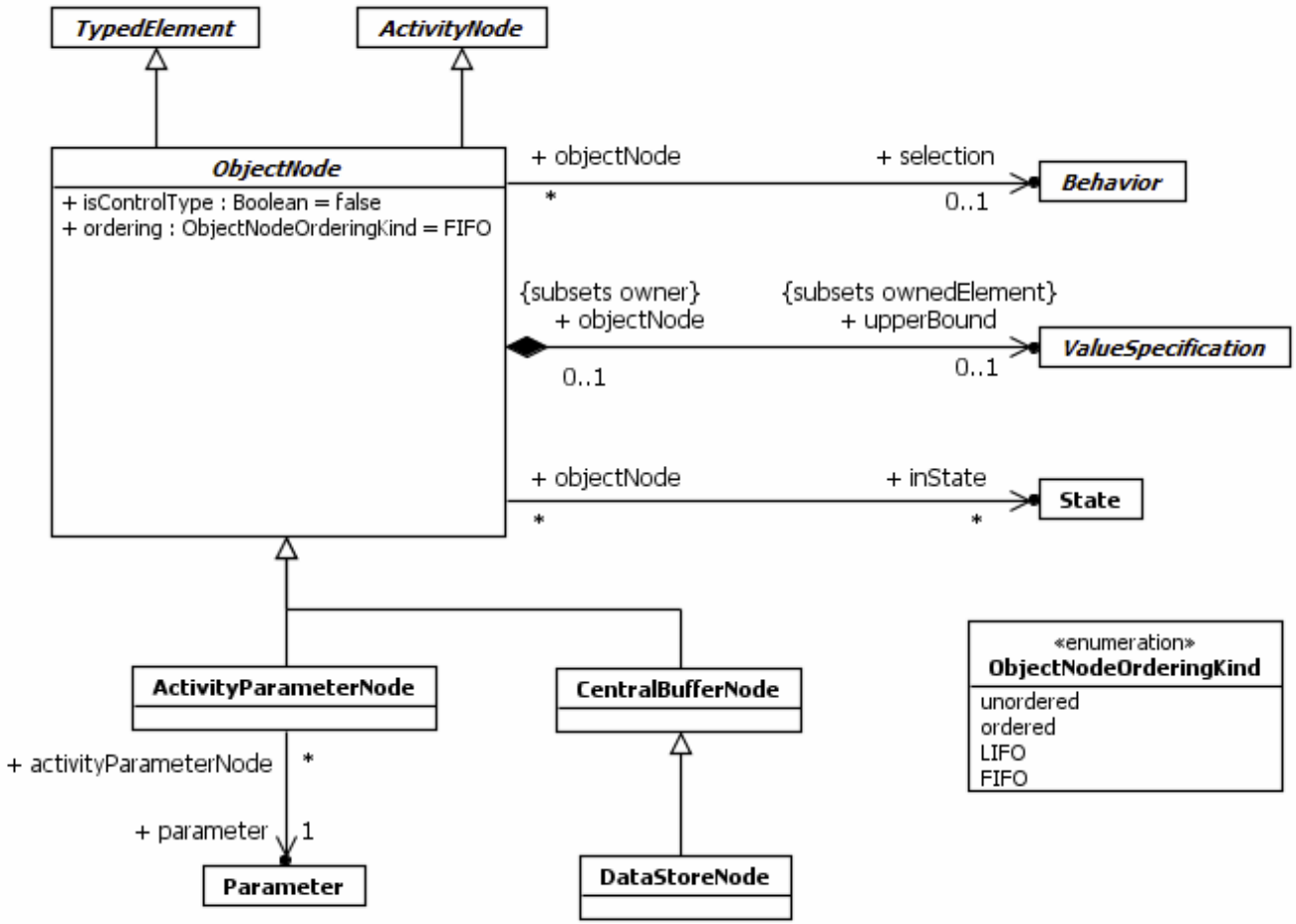
object-node-ordering-kind ::= 'unordered' | 'ordered' | 'FIFO' | 'LIFO'

ObjektKnoten können auf einen bestimmten **Zustand beschränkt** sein **[…]**.
D.h., nur Objekt-Token mit dieser Beschränkung werden aufgenommen.

d.h. Objekt-Token, die der Forderung **nicht** genügen, verbleiben in den jeweiligen Vorgängerknoten

Objektknoten (Metamodell)





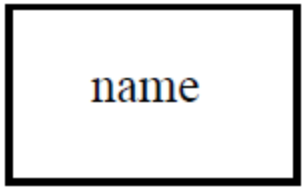
Konsequenz:

Objekte aktiver Klassen werden zu ObjektToken. Diese verfügen über eine Zustandsmaschine zur Verhaltensbeschreibung (1.Dimension)

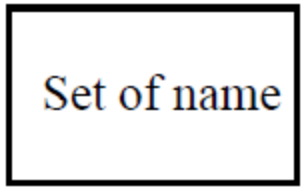
Eine Darstellung des Objektflusses ist aber auch durch Aktivitäten darstellbar (2.Dimension)

→ mehrdimensionale Verhaltensbeschreibung

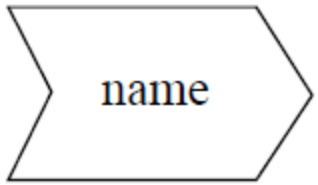
Arten von Objekt-Knoten



Object node



Object node for tokens containing sets

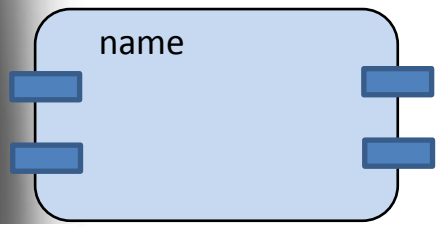


Object node for tokens with signal as type

Aktivitätsdiagramme

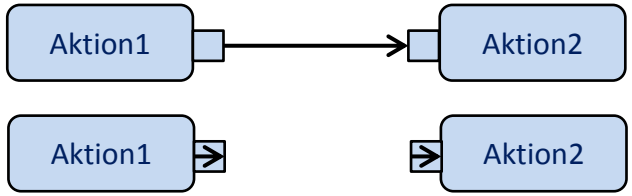
1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
   - Objektknoten allgemein
   - Parameter von Aktivitäten und Aktionen, Werte-Pin (Konstante)
   - Pufferknoten

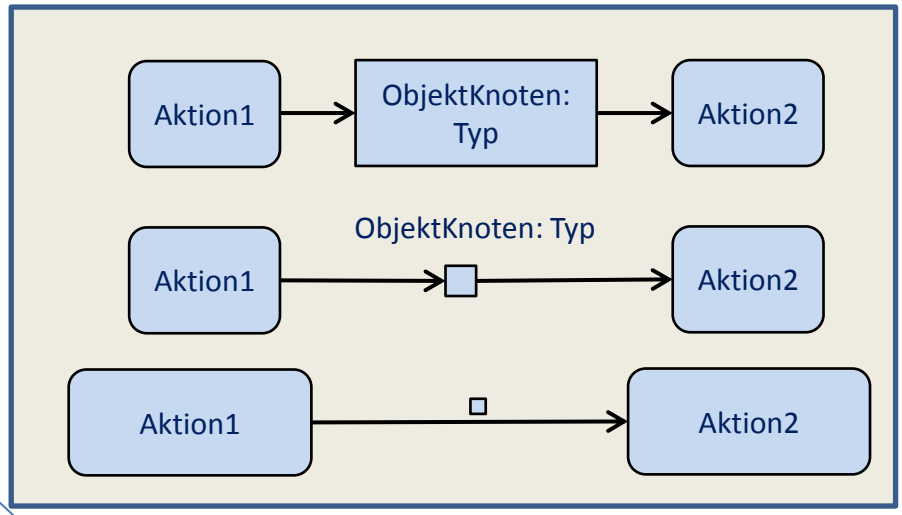
Objektknoten als Parameter *von Aktivitäten und Aktionen*

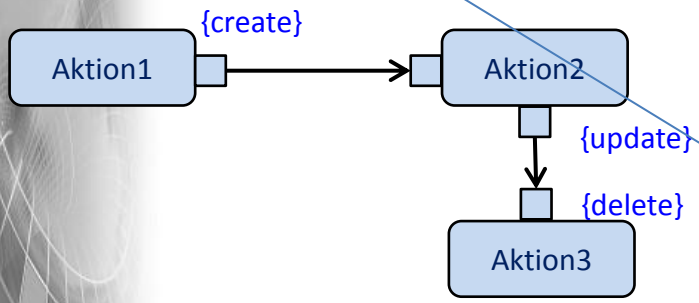


Eingabepins Ausgabepins

- Formale Parameter werden als Pins bezeichnet
- Notationsvarianten für Pins
- Mögliche Effektangabe für die Parameter:
  {create, read, update, delete}







alternative Darstellung bei Beibehalt der Typen

Präzise Aussagen zur Bedeutung fehlen (noch)

WertePins

als spezielle Ausprägung eines Eingabepins

Achtung:
Aktivität wird durch Wert allein noch nicht gestartet

Ausführung einer Aktivität mit Parametern

Allg. Prinzip:

- Aufrufer der Aktivität stellt aktuelle Parameter zur Verfügung
- Kontroll-Token des Aufrufers wird aber nur zum Kontroll-Token des Gerufenen, wenn Vorbedingung erfüllt ist
- Nach Beendigung gibt die gerufene Aktivität Daten über AusgabeParameter zurück

Aktivitätsdiagramme

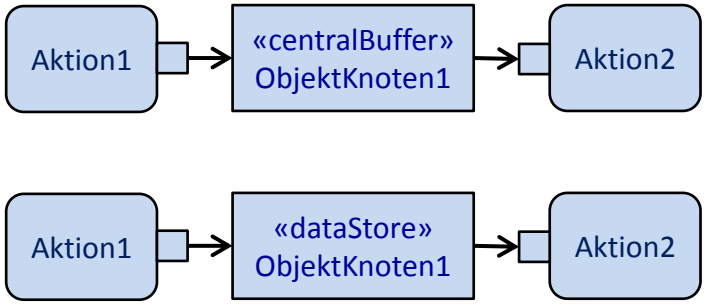
1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
   - Objektknoten allgemein
   - Parameter von Aktivitäten und Aktionen, Werte-Pin (Konstante)
   - Pufferknoten

Pufferknoten

- derartige ObjektKnoten dienen der **Pufferung von Objekt-Token** beliebig vieler Vorgängerknoten, die beliebig vielen Nachfolgerknoten zur Verfügung gestellt werden

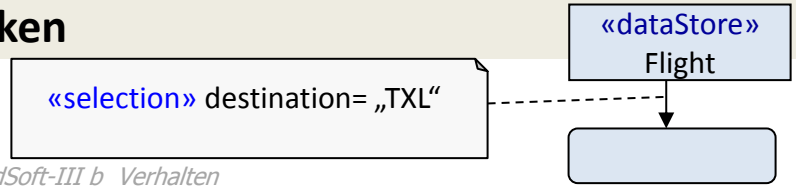
- **Unterscheidung** erfolgt nach der **Speicherungsdauer** der eingehenden DatenToken:
  a) Transienter PufferKnoten «centralBuffer»
  Speicherung nur zur momentanen Verarbeitung
  Weiterreichung von Token erfolgt ohne Duplizierung

  b) Persistente PufferKnoten «dataStore»
  vor Weiterreichung erfolgt eine Duplizierung
  (Referenz oder Wert???)



Achtung: keine Mehrfachspeicherung identischer Objekte
(wie bei herkömmlichen Objektknoten)

- explizite Entnahme der Objekt-Token aus einem persitenten Pufferknoten durch Einsatz nutzer-spezifischer **SelektionsOperatoren,** die Objekt-Token-Flüsse **einschränken**



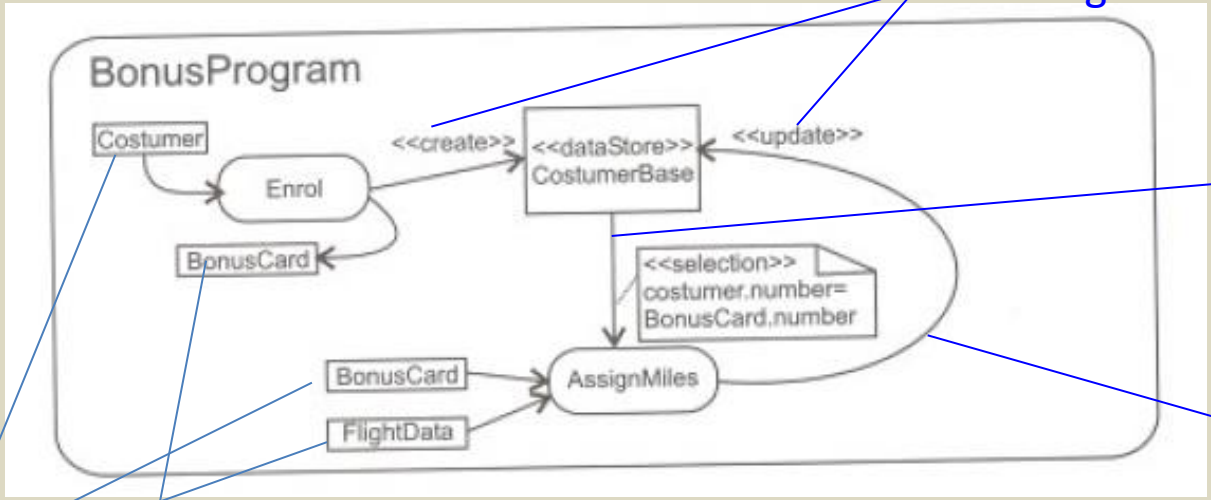
Problematisches Beispiel aus der Literatur "UML-Glasklar"

- **unklar**: Bei weiterer Annahme einer Referenzsemantik ist das Beispiel **falsch**

Bei Annahme einer Wertesemantik (Vorschlag aus dem Auditorium: Token verfügt über über Objektwert und einen ID-Wert) lässt sich das Beispiel interpretieren

Eine Angabe des Tokentyps wurde nicht vorgenommen

Beispiele für zusätzliche Effektangabe im Objektfluss



Tokenanzahl in CostumerBase wird nicht reduziert

Wird Tokenanzahl Erhöht?

Ein- und Ausgabepins sind am Rand platzieren

Aktivitätsdiagramme

1. Allgemeine Charakterisierung
2. Kontrollflussknoten: Typen
3. Objektknoten: Typen
   - Objektknoten allgemein
   - Parameter von Aktivitäten und Aktionen, Werte-Pin (Konstante)
   - Pufferknoten
4. Aktionen, vordefinierte Aktionen

Invocation-Aktionen (1)

- **Call Actions**

  A CallAction is an **InvocationActio**n that calls a Behavior or an Operation.

  1. A CallBehaviorAction is a CallAction that invokes a Behavior directly, rather than calling a BehavioralFeature that, in turn, results in the invocation of a Behavior.

  2. A CallOperationAction is a CallAction that transmits an Operation call request message to the target object, where it may cause the invocation of an associated Behavior. The target object is taken from the target InputPin of the CallOperationAction. The handling of an Operation call request by the target object is described under Behavioral Features and Methods in sub clause 13.2.3 and Message Event in sub clause 13.3.3.

  3. A StartObjectBehaviorAction is a CallAction that starts the execution either of a directly instantiated Behavior or of the classifierBehavior of an object. The object to be started is taken from the object InputPin. If the input object is an instantiated Behavior that is not already executing, then it begins executing. If the input object has a classifierBehavior that is not already executing, then it is instantiated and started. In either case, if the identified Behavior is already executing, then the StartObjectBehavior has no effect.

Invocation-Aktionen (2)

- **Send Actions**

  A send Action is an action that transmits an object asynchronously to one or more target objects. As a send Action is always asynchronous, it may have argument inputs, but it has no result outputs. The Action completes as soon as the object is sent, whether or not it has been received yet.

  1. A SendSignalAction is an InvocationAction that creates a Signal instance and transmits the instance to the object given on its target InputPin. A SendSignalAction must have argument InputPins corresponding, in order, to each of the (owned and inherited) Properties of the Signal being sent, with the same type, ordering and multiplicity as the corresponding attribute. The values of each Property of the transmitted Signal instance are taken from the argument InputPin corresponding to the Property. The handling of the Signal instance by the target object is described under Behavioral Features and Methods in sub clause 13.2.3 and Message Events in sub clause 13.3.3.
  2. A BroadcastSignalAction is an InvocationAction that creates a Signal instance using values from its argument InputPins, similarly to a SendSignalAction. However, instead of sending the Signal instance to a single target object, it transmits the instance potentially to all available target objects in the system. The manner of identifying the exact set of objects that are broadcast targets is not defined in this specification, however, and may be limited to some subset of all the objects that exist.
  3. A SendObjectAction is an Invocation action that transmits any kind of object to the object given on its target InputPin. The object to be transmitted is given on the single request InputPin of the SendObjectAction. If the object is a Signal instance, then it may be handled by the target object in the same way as an instance sent from a SendSignalAction or BroadcastSignalAction. Otherwise, the reception of the object can only be handled using an AnyReceiveEvent (as described under Message Events in sub clause 13.3.3).

Invocation-Actions und Ports

- A CallOperationAction, SendSignalAction, or SendObjectAction may send a request through a Port by targeting an object having the Port and identifying the Port with the Action's onPort attribute.

  Other kinds of InvocationActions shall not have a value for the onPort attribute.

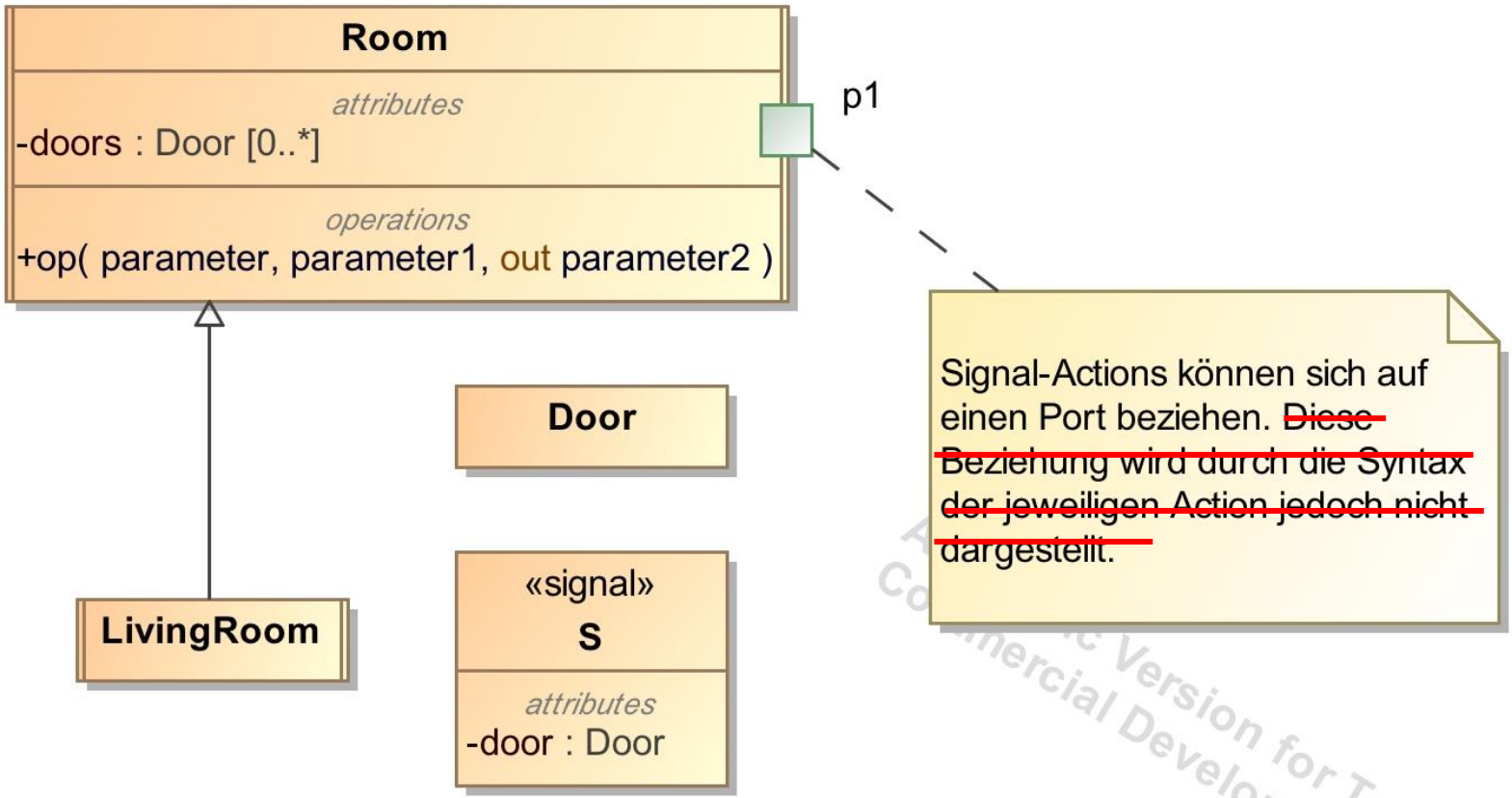
The value of onPort is shown by the phrase "via <port>" in the name string of the symbol denoting the particular InvocationAction.

Wichtiges Puzzle-Stück zur Verhaltensbeschreibung (z.B. Zustandsautomaten von DemonGam,e

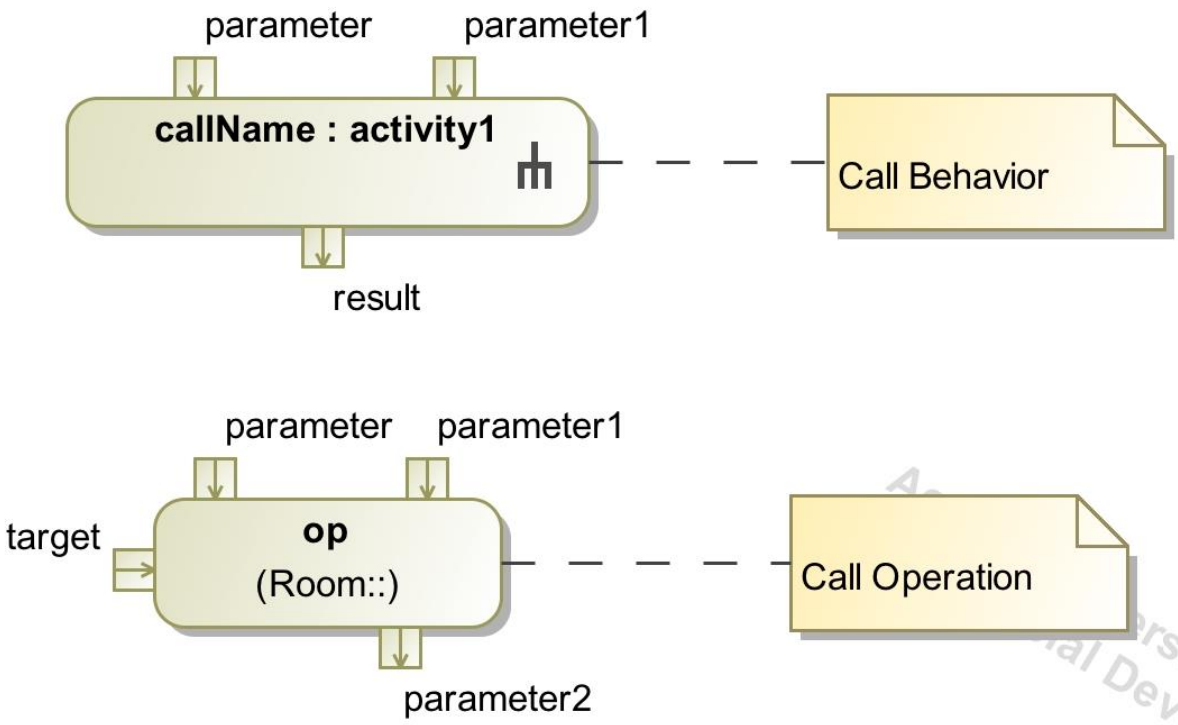
Objekt-Aktionen

- Create Object Actions
- Destroy Object Actions
- Test Identity Actions
- Read Self Actions
- Value Specification Actions
- Read Extent Actions
- Reclassify Object Actions
- Read-Is-Classified-Object Actions
- Start Classifier Behavior Actions

In den Action-Beispielen verwendete Strukturelemente

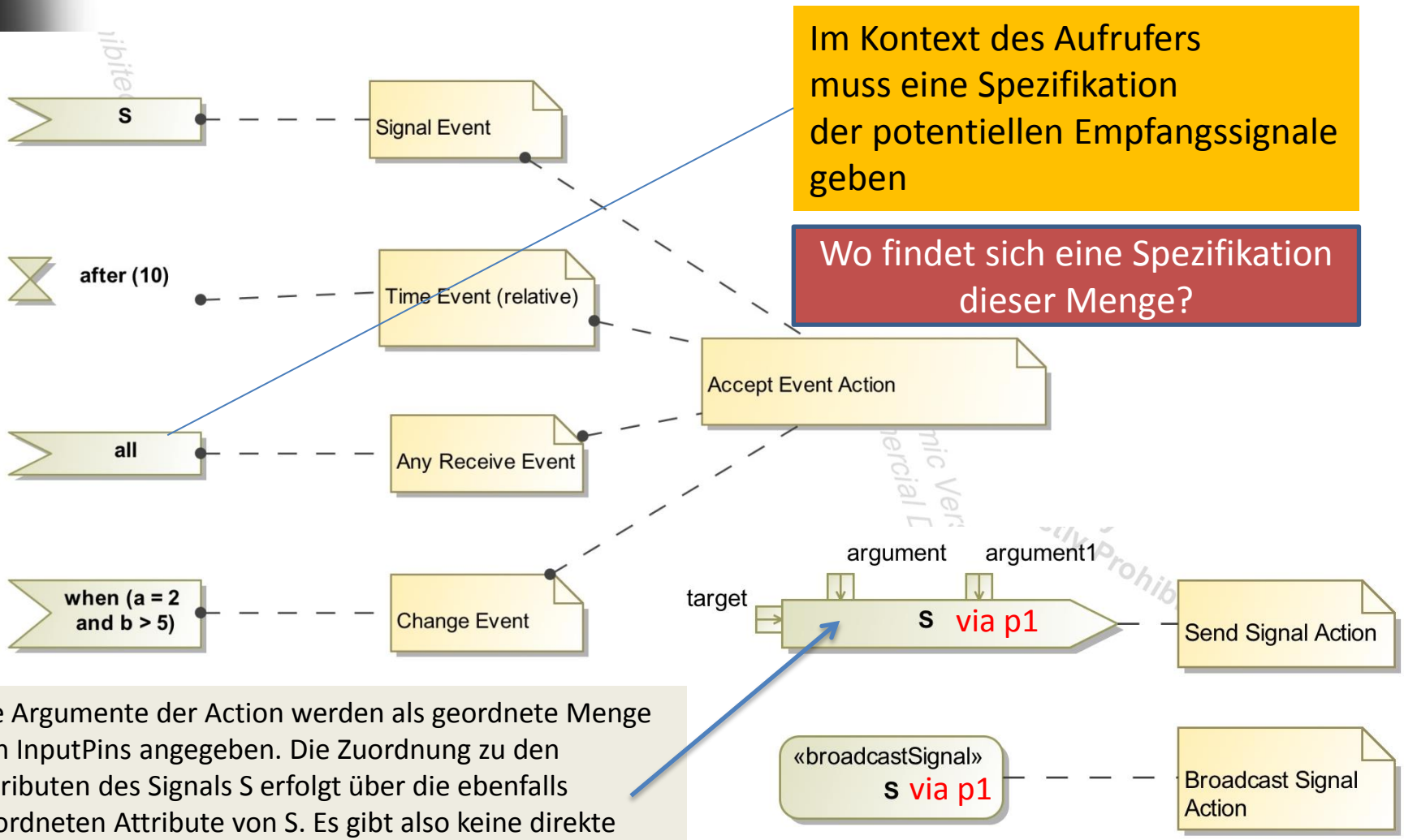


Call-Actions



Achtung: Reihenfolge der Parameter bei der Definition entspricht beim Anlegen der Pins der CallAction

Actions mit Ereignis-Bezug



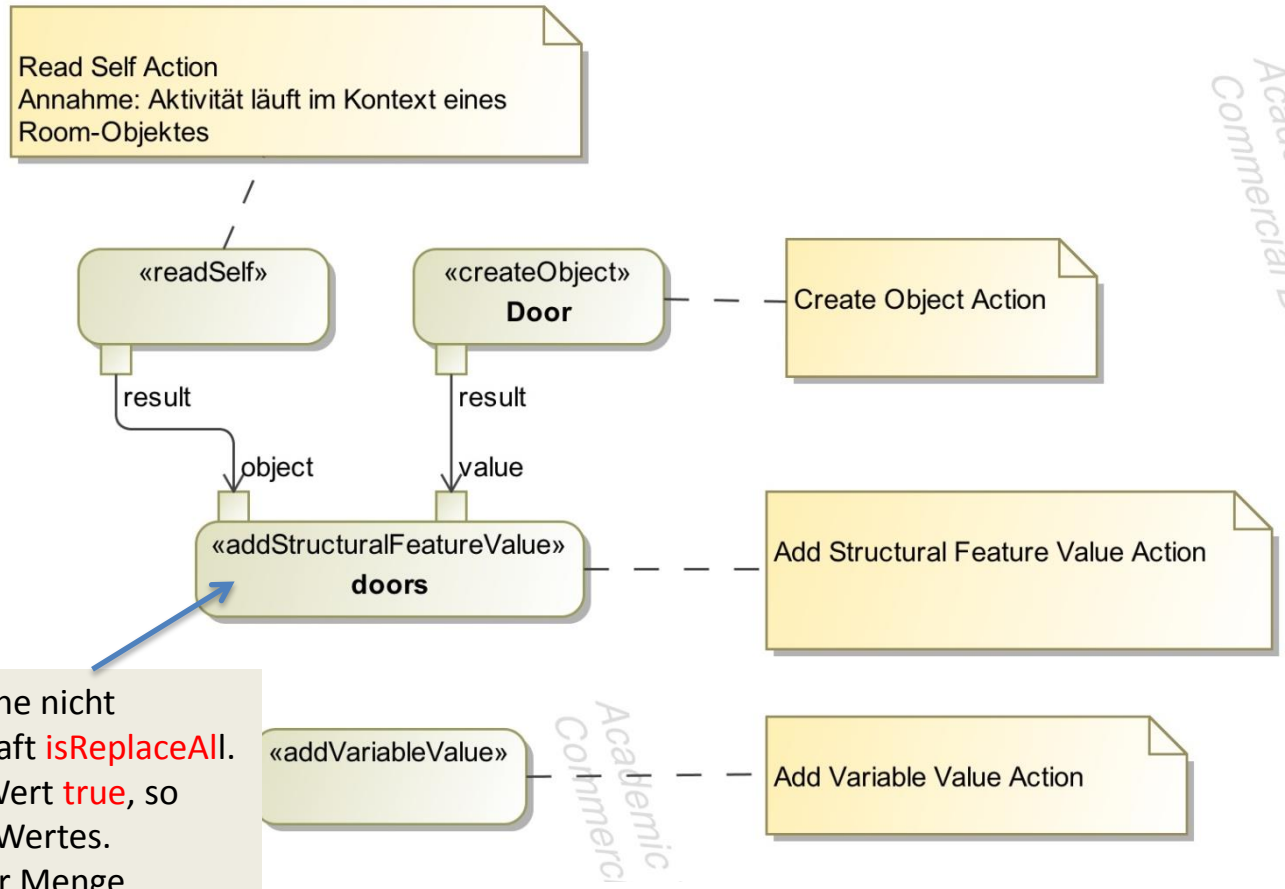
Im Kontext des Aufrufers muss eine Spezifikation der potentiellen Empfangssignale geben

Wo findet sich eine Spezifikation dieser Menge?

Die Argumente der Action werden als geordnete Menge von InputPins angegeben. Die Zuordnung zu den Attributen des Signals S erfolgt über die ebenfalls geordneten Attribute von S. Es gibt also keine direkte Zuordnung eines InputPins zu einem Attribut. Die Art der Darstellung lässt dabei keinen Rückschluss auf die Ordnung zu.

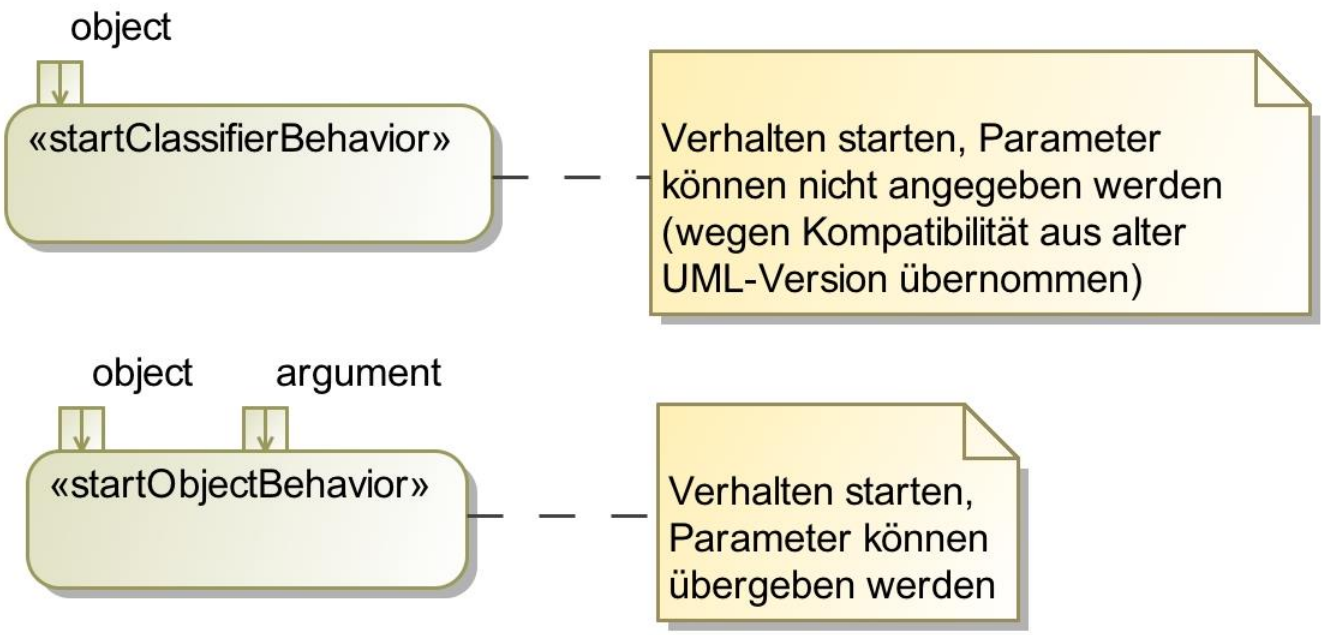
Objekt- und Variablen-Actions

Größte Überraschung: addStructuralFeatureValue-Action hat zwei Verhaltensspielarten

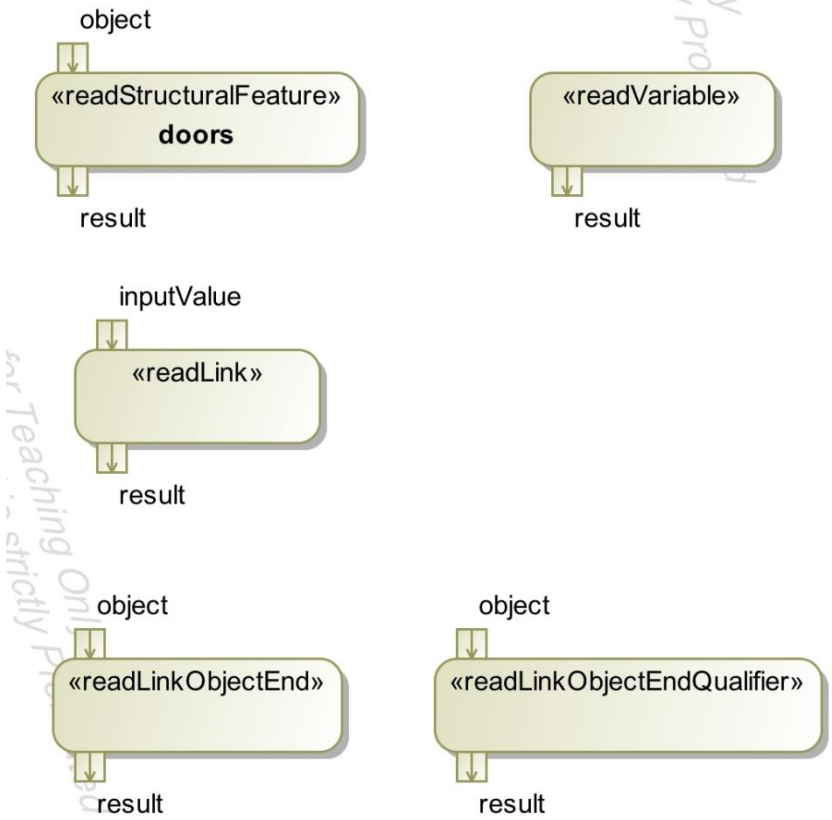


Die Action hat außerdem eine nicht sichtbare Boolean-Eigenschaft isReplaceAll. Hat diese Eigenschaft den Wert true, so erfolgt eine Zuweisung des Wertes. Sonst wird der Wert zu einer Menge hinzugefügt.

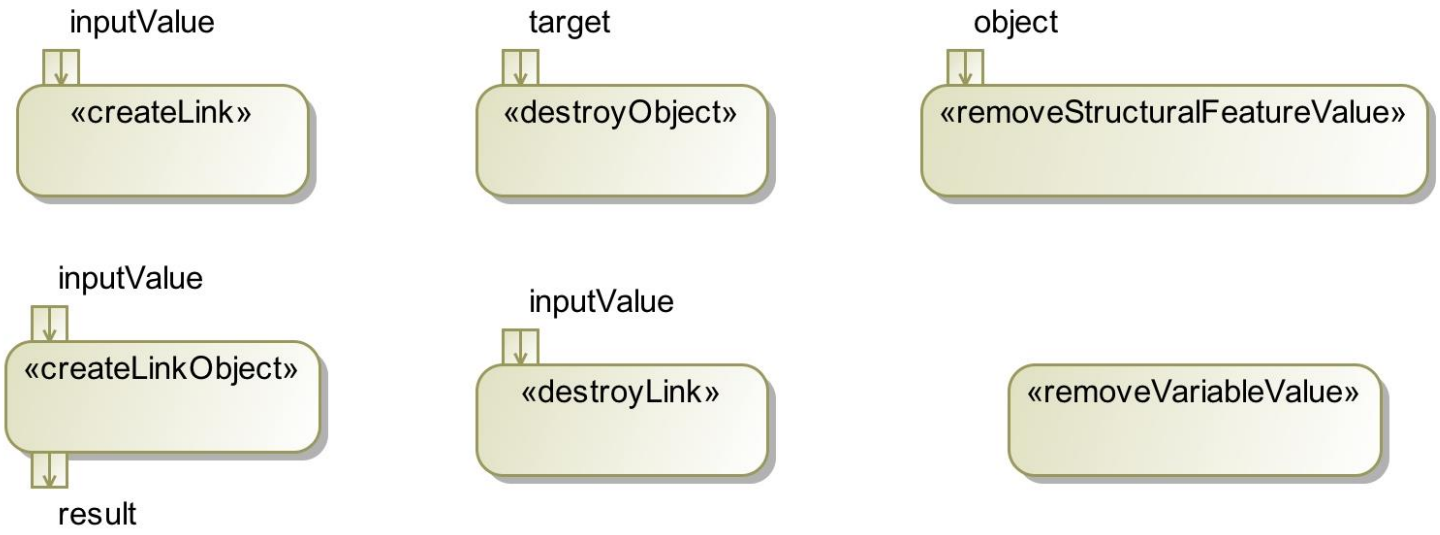
Actions zur Verhaltensinitiierung



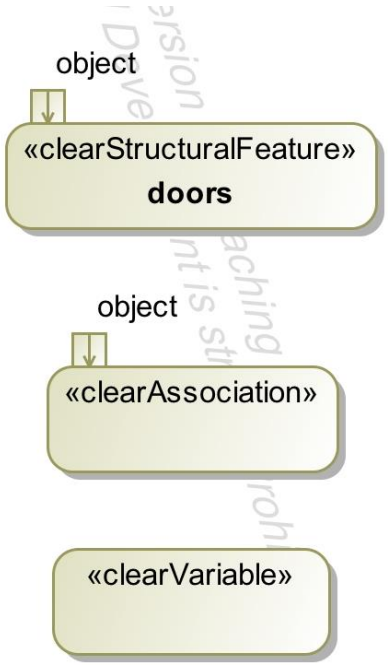
Objekt- und Variablen-Actions



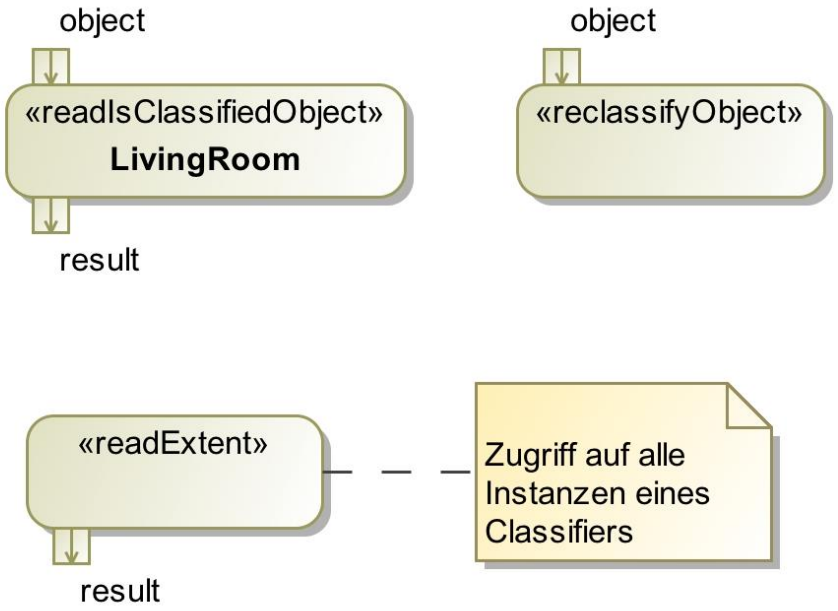
Objekt- und Variablen-Actions



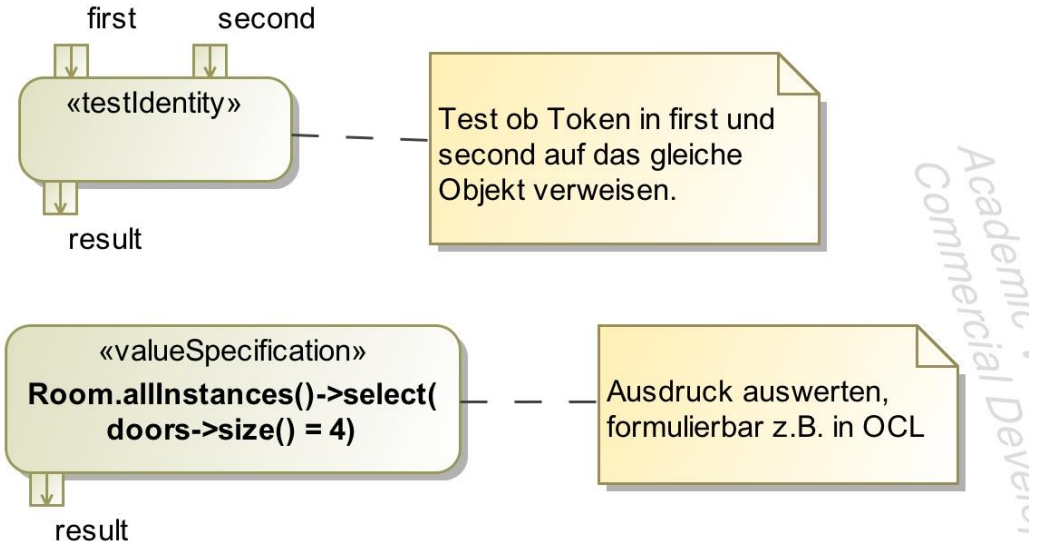
Objekt- und Variablen-Actions



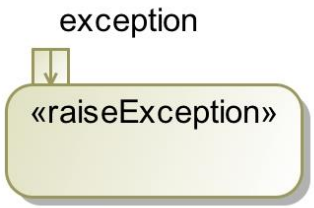
Objekt-Actions



Objekt- und Wert-Action



Exception-Action



Nicht standardisierte Actions