LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung (WS 2013/14)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

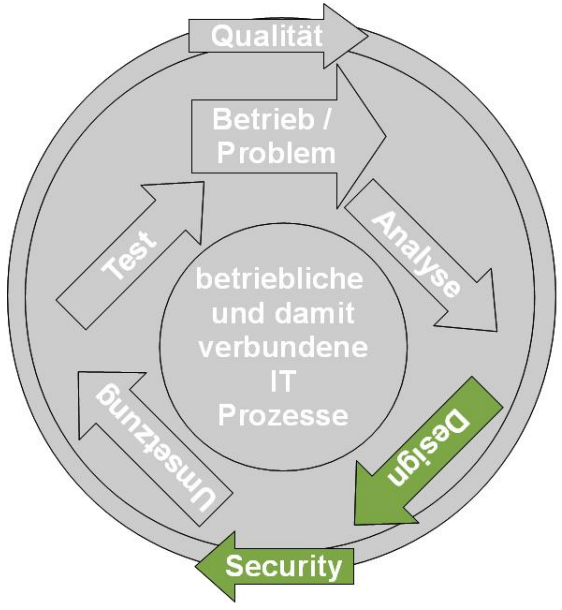
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

1.1: Grundlagen: Object Constraint Language

v. 09.11.2013

Einordnung
Object Constraint Language (OCL)

- Modellgetriebene SW-Entwicklung
  - Einführung
  - Object Constraint Language (OCL)
  - Modellbasierte Softwareentwicklung
  - Eclipse Modeling Foundation (EMF)
  - Model Driven Architecture (MDA)
- Qualitätsmanagement
- Testen
- Modellbasierte Sicherheit



Abschnitt basiert auf Vortrag "Introduction to OCL" von V. Bembenek, H. Schmidt, M. Heisel.

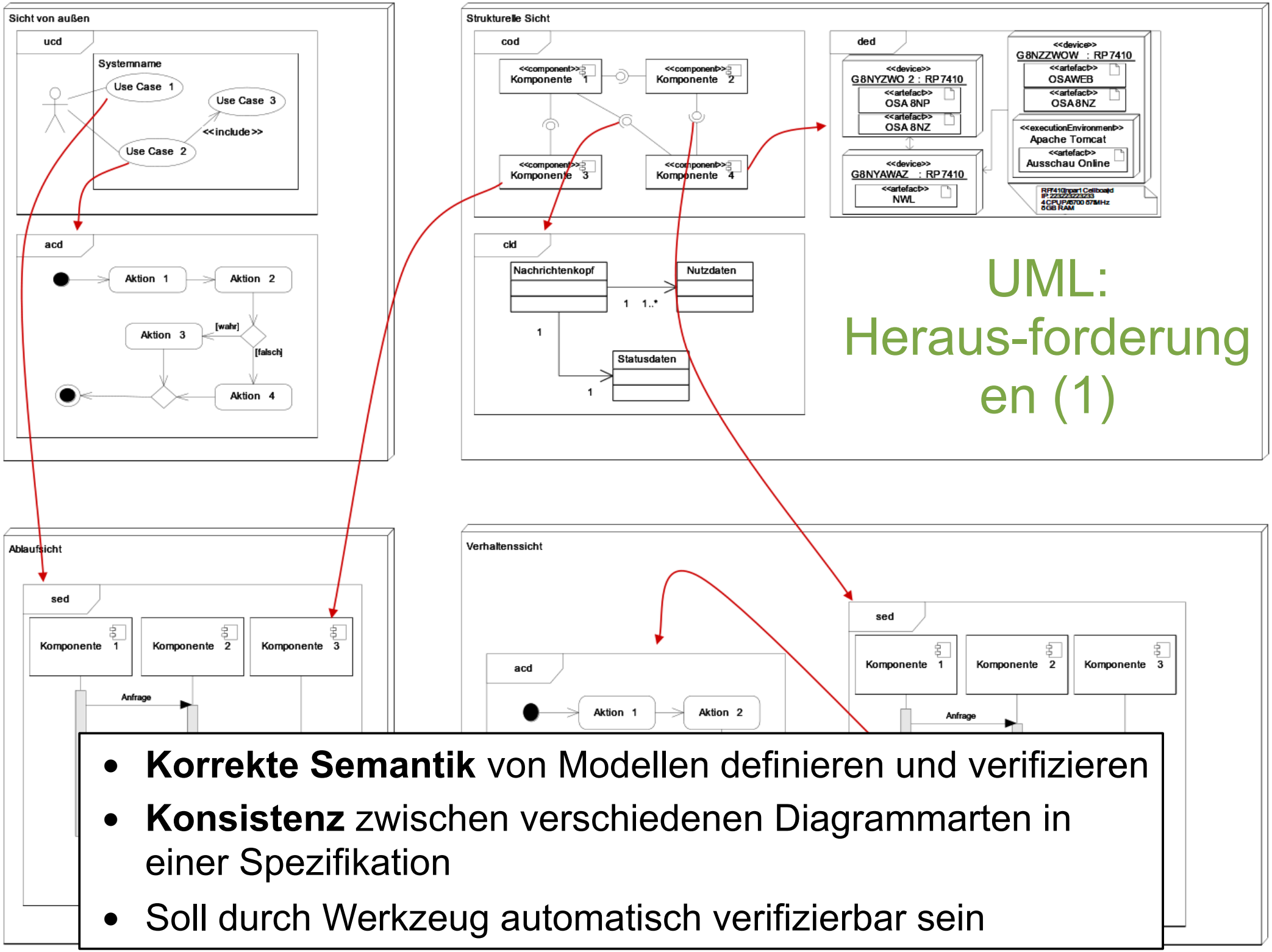
Literatur (s. Webseite):

- V. Gruhn: **MDA - Effektives Software-Engineering**. Kapitel 6.3.
- J. Seemann, J.W. Gudenberg: **Software-Entwurf mit UML 2**. Kapitel 14.5.
- J. Warmer, A. Kleppe: ***The Object Constraint Language***.

1.1 Object Constraint Language (OCL)

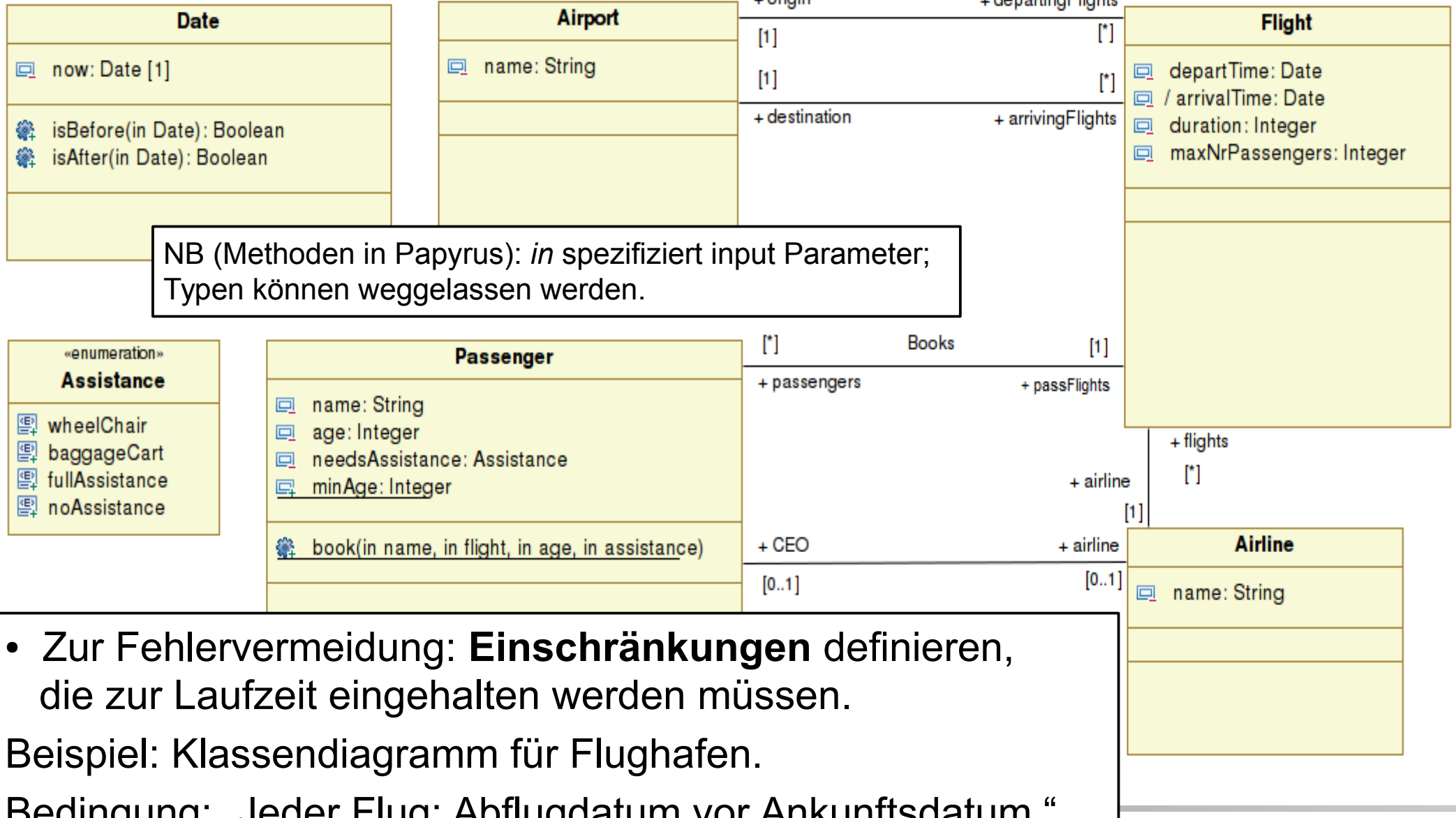
1.1 OCL

- Motivation & Einführung
- Assoziationen, Navigationen, Operationen
- Vor- und Nachbedingungen
- Anhang: OCL Typen


Sicht von außen
ucd
Systemname
Use Case 1
Use Case 3
<<include>>
Use Case 2
acd
Aktion 1
Aktion 2
Aktion 3
[wahr]
[falsch]
Aktion 4
Strukturelle Sicht
cod
<<component>>
Komponente 1
Komponente 2
Komponente 3
Komponente 4
cld
Nachrichtenkopf
Nutzdaten
1
1..*
Statusdaten
ded
<<device>>
G8NYZWO 2 : RP7410
<<artefact>>
OSA 8NP
OSA 8NZ
G8NYAWAZ : RP7410
NWL
G8NZZWOW : RP7410
OSAWEB
OSA8NZ
<<executionEnvironment>>
Apache Tomcat
Ausschau Online
UML:
Heraus-forderung
en (1)
Ablaufsicht
sed
Anfrage
Verhaltenssicht
acd
Aktion 1
Aktion 2
sed
Anfrage
• Korrekte Semantik von Modellen definieren und verifizieren
• Konsistenz zwischen verschiedenen Diagrammarten in einer Spezifikation
• Soll durch Werkzeug automatisch verifizierbar sein

UML: Herausforderungen (2)

LEHRSTUHL 14
SOFTWARE ENGINEERING



- Zur Fehlervermeidung: **Einschränkungen** definieren, die zur Laufzeit eingehalten werden müssen.

Beispiel: Klassendiagramm für Flughafen.

Bedingung: „Jeder Flug: Abflugdatum vor Ankunftsdatum.“

Zur Diskussion: Wie in UML-Modell spezifizieren ?

Lösung: Object Constraint Language (OCL)

OMG Spezifikation: **„Object Constraint Language 2.0“**, Teil von UML
http://www.omg.org/spec/OCL/2.2/PDF

Logik-basierte Notation für **Einschränkungen** (Constraints) in UML-Modellen:

- Welche **Klassen erreichbar**; welche Attribute, Operationen und Assoziationen für Objekte dieser Klassen vorhanden.
- **Bedingungen** an Wertebelegungen während Ausführung der modellierten Systemteile (vgl. Assertions in Programmquellcode).

Unterstützt QS bei modellbasierter Softwareentwicklung.

Zwei Arten von **Einschränkungen** spezifizieren und verifizieren:

Vorteile:

- Automatische Verifikation der Bedingung.
- Präzise Spezifikation der Bedingungen **beseitigt Mehrdeutigkeit.**
- **Werkzeuge** erzeugen aus OCL **Assertions** in **Java**.

Wie verwende ich OCL ?

OCL-Ausdrücke: an UML-Modell gebunden und im UML-Modell oder separatem Dokument angegeben. Beschreiben Einschränkungen für Elemente des Modells, zu dem sie gehören, z.B.:

- **Fortlaufende** Zustandsbeschränkung (mit Invarianten).
- Zustandsbeschränkungen **vor** bzw. **nach Methodenaufruf** (mit Vor- und Nachbedingungen).

Keine Programmiersprache:

→ Modellierung von Programmlogik nicht vorgesehen und möglich !

- Insbesondere: OCL-Ausdrücke **ohne Seiteneffekte:**
  - Kann nichts im Modell verändern.
  - OCL-Ausdruck verursacht keine Zustandsänderung, auch wenn er sie spezifizieren kann (z.B. in einer Nachbedingung).

OCL unterstützt Prüfung von **strenger Typisierung.**

Einschränkungen (Constraints)

Constraint (Einschränkung):
Einschränkung auf einem oder mehreren Teilen eines UML-Modells.

Gibt folgende **Arten** von Einschränkungen:

Class Invariant (Klasseninvariante):
Muss *immer* von allen Instanzen einer Klasse erfüllt sein.

Pre-condition (Vorbedingung):
Muss erfüllt sein, *bevor* Operation ausgeführt wird.

Post-condition (Nachbedingung):
Muss *nach* Ausführen einer Operation erfüllt sein.

Guard condition (Abdeckungsbedingung):
Muss *vor* Transition im Zustandsdiagramm, vor einer Nachricht im Sequenzdiagramm oder vor anderen UML-Modellen, die Verhalten modellieren, erfüllt sein.

Basisform eines OCL-Ausdrucks

context <identifier>
<constraintType> [<constraintName>]: <boolean expression>

context

- Schlüsselwort, um zugehöriges Modellelement zu markieren.
- Kann andere Modellelemente referenzieren.
- Schlüsselwort selbst kann innerhalb <boolean expression> genutzt werden, um Kontext zu betreten.

<identifier>	Klassen- oder Operationsname.
<constraintType>	eins der Schlüsselwörter *inv*, *pre* oder *post*.
<constraintName>	optionaler Name für Einschränkung.
<boolean expression>	boolescher Ausdruck.

OCL Typen und Schlüsselwörter

Folgende **Typen** im OCL-Ausdruck benutzbar:

- **Vordefinierte Typen:**
  - Primitive Typen: String, Integer, Real, Boolean.
  - Kollektionstypes: Set, Bag, Sequence, OrderedSet.
  - Tupel-Typen: Tuple.
  - Spezielle Typen: OclType, OclAny, …
- **Klassifikatoren** vom UML-Modell und seiner Eigenschaften:
  - Klassen, Enumerationsklassen und Rollennamen.
  - Attribute und Operationen.

Folgende **Schlüsselwörter** im OCL-Ausdruck benutzbar:

- Konditionalausdrücke: *If-then-else-endif*
- Boolesche Operatoren: *Not, or, and xor, implies*
- Globale Definitionen: *Def*
- Lokale Definitionen: *Let-in*

Klasseninvarianten

Invariante:

- **Bedingung**, **die** fortwährend **erfüllt sein muss**:
  „An … invariant ... must be true for all instances of that type at any time“
  („Object Constraint Language 2.0“ p. 8)
- Anhand des Schlüsselwortes *inv* im Kontext der Instanz eines Klassifikators (Klasse, Rollenname...) spezifizierbar**.**

Gebräuchliche Typen von Invarianten

Beschränkung von Domänen:

- Beschränkung der Werte, die Attribut annehmen kann.

Beschränkung auf Einmaligkeit:

- Attribut oder Attributmenge einer Klasse für die gilt:
  - Für zwei unterschiedliche Instanzen dieser Klasse dürfen keine gleichen Werte zugewiesen werden.

Zeitliche Beschränkung:

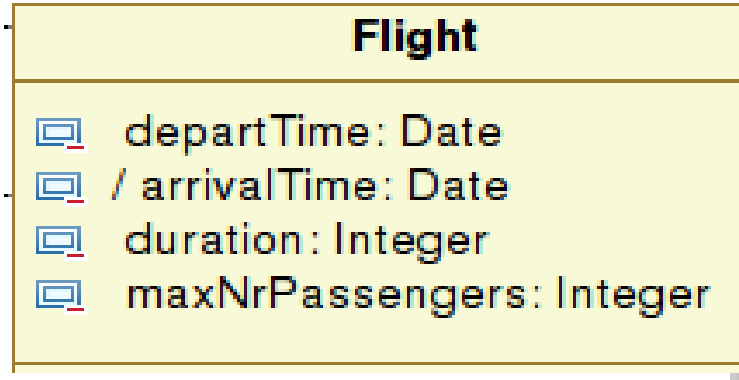
- Bedingungen, die abgeleitete Modellelemente definieren (z.B abgeleitet Attribute).

Regeln für Existenz:

- Bestimmte Objekte / Werte müssen existieren / definiert sein.
- Bevor andere Objekte / Werte definiert / erzeugt werden können.

Invarianten auf Attributen

- Klasse, die von Invarianten referenziert wird: **Kontext der Invarianten.**
- **Gefolgt vom booleschen Wert**, der Invariante angibt.



- Alle Attribute der Kontextklasse in Invarianten nutzbar.
- Beispiel: „Jeder Flug dauert weniger als 4 Stunden.“

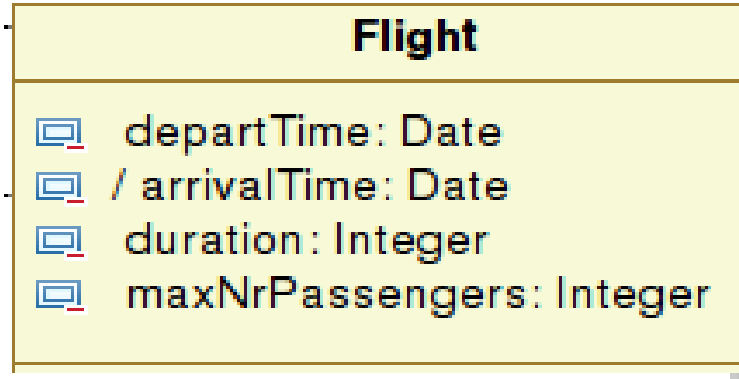
Beispiel

context <identifier>
<constraintType>: <boolean expression>

context Schlüsselwort
<identifier> Klassen- oder Operationsname
<constraintType> Schlüsselwort inv, pre, post
<boolean expression> boolescher Ausdruck

Invarianten auf Attributen

- Klasse, die von Invarianten referenziert wird: **Kontext der Invarianten.**
- **Gefolgt vom booleschen Wert**, der Invariante angibt.



- Alle Attribute der Kontextklasse in Invarianten nutzbar.
- Beispiel: „Jeder Flug dauert weniger als 4 Stunden.“

Beispiel

context Flight

inv: duration < 4

context <identifier>
<constraintType>: <boolean expression>

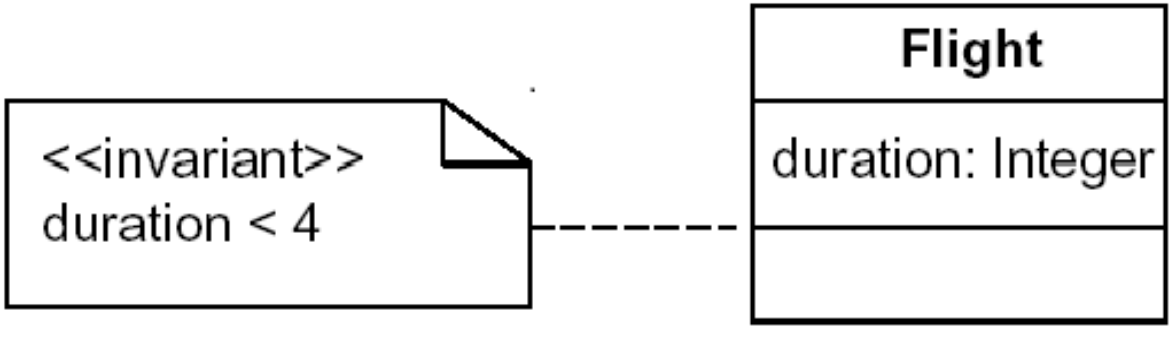
context Schlüsselwort
<identifier> Klassen- oder Operationsname
<constraintType> Schlüsselwort inv, pre, post
<boolean expression> boolescher Ausdruck

Gebrauch von self

Folgende Notationen sind äquivalent:

context Flug
inv: self.duration < 4

context Flug
inv: duration < 4



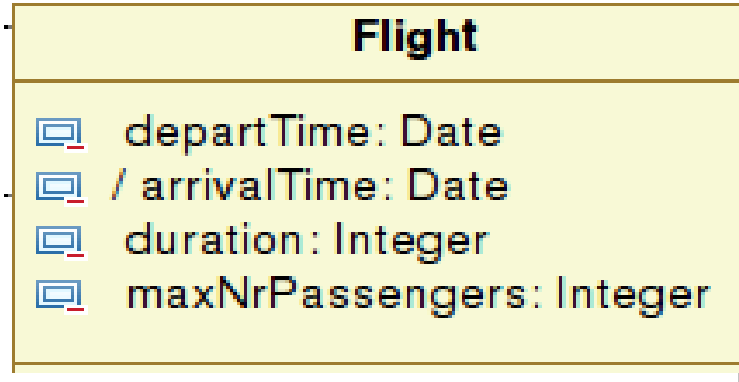
Invarianten auf Attributen: Abfrageoperationen

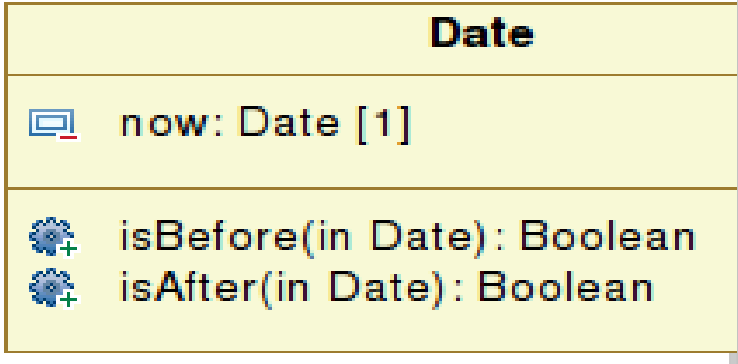
LEHRSTUHL 14
SOFTWARE ENGINEERING

- Wenn Attributtyp Klasse ist:

  → Attribute und **Abfrageoperationen** dieser Klasse für Erstellung der Invarianten nutzbar (anhand Punkt-Notation).

- Abfrageoperation: Operation, die **Wert von Attributen nicht ändert.**
- Beispiel: „Abflugdatum eines Fluges ist vor Ankunftsdatum.“





Beispiel

context

context <identifier>
<constraintType>: <boolean expression>

context Schlüsselwort
<identifier> Klassen- / Operationsname.
<constraintType> Schlüsselwort inv, pre, post.
<boolean expression> boolescher Ausdruck

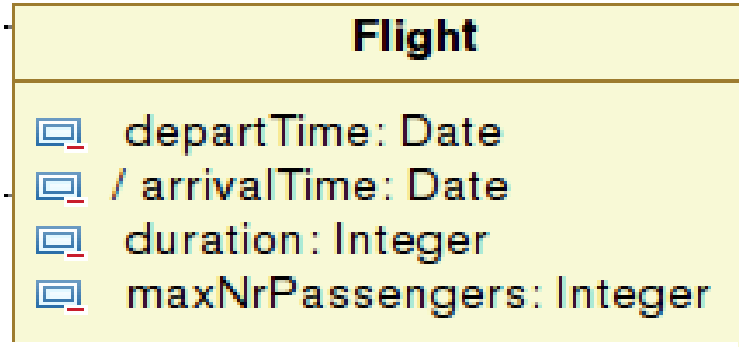
Invarianten auf Attributen: Abfrageoperationen

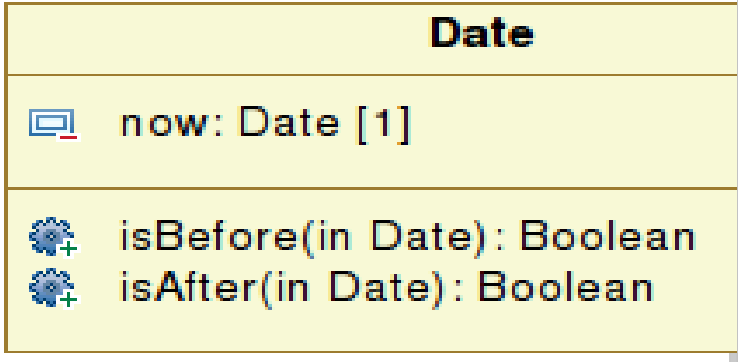
LEHRSTUHL 14
SOFTWARE ENGINEERING

- Wenn Attributtyp Klasse ist:

  → Attribute und **Abfrageoperationen** dieser Klasse für Erstellung der Invarianten nutzbar (anhand Punkt-Notation).

- Abfrageoperation: Operation, die **Wert von Attributen nicht ändert.**
- Beispiel: „Abflugdatum eines Fluges ist vor Ankunftsdatum."





Beispiel

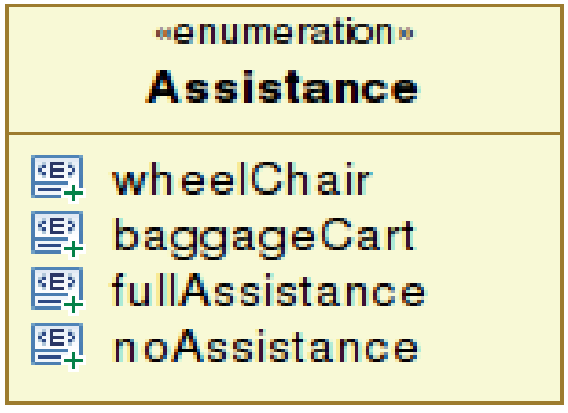
context Flight

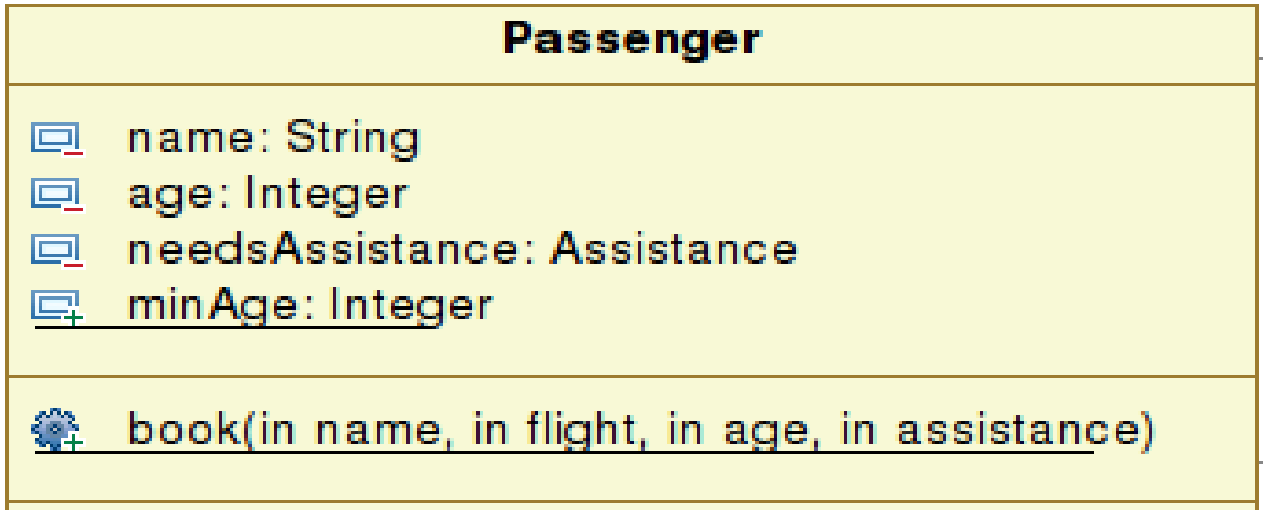
inv: departTime.isBefore(arrivalTime)

context <identifier>
<constraintType>: <boolean expression>

context Schlüsselwort
<identifier> Klassen- / Operationsname.
<constraintType> Schlüsselwort inv, pre, post.
<boolean expression> boolescher Ausdruck

Aufzählungstypen





Aufzählung nutzt Datentypen gefolgt von :: und einem Wert.

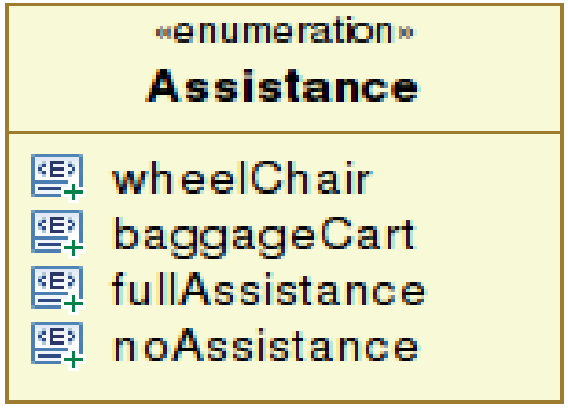
Beispiel

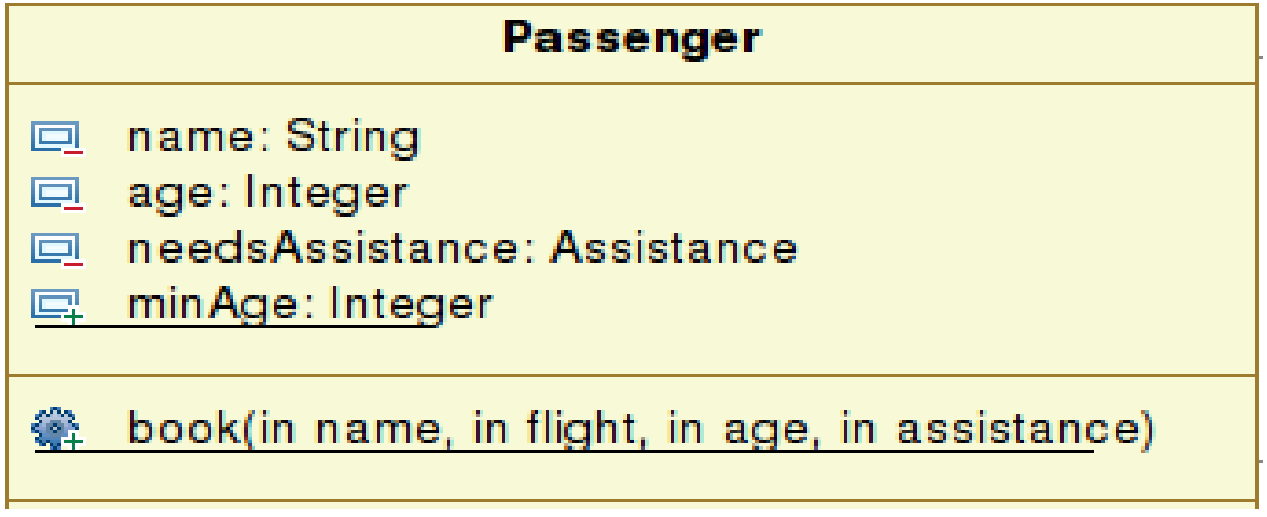
context Passenger

inv: self.age > 95 implies
self.needsAssistance = Assistance :: wheelChair

Bedeutung: ?

Aufzählungstypen





Aufzählung nutzt Datentypen gefolgt von :: und einem Wert.

Beispiel

context Passenger

inv: self.age > 95 implies
self.needsAssistance = Assistance :: wheelChair

Bedeutung: Jeder Passagier über 95 braucht einen Rollstuhl.

1.1 OCL

- Motivation & Einführung
- **Assoziationen, Navigationen, Operationen**
- Vor- und Nachbedingungen
- Anhang: OCL Typen

Assoziation und Navigation

- Jede Assoziation ist **Navigationspfad.**
- Kontext des Ausdrucks ist **Startpunkt.**
- Rollennamen (oder Assoziationsenden) werden genutzt, um **navigierte Assoziationen** zu identifizieren.

Assoziation und Navigation: Beispiel

Beispiel: „Abflugort eines Fluges ist immer ungleich Ziel.“

Beispiel

context

Beispiel: „Abflugort eines Fluges ist immer Duisburg.“

Beispiel

context

Assoziation und Navigation: Beispiel

Beispiel: „Abflugort eines Fluges ist immer ungleich Ziel.“

Beispiel

context Flight

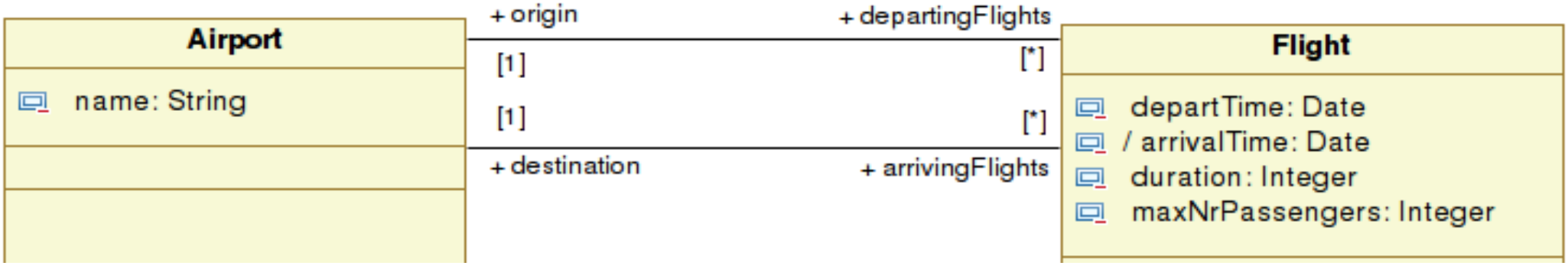
inv: origin <> destination

Beispiel: „Abflugort eines Fluges ist immer Duisburg.“

Beispiel

context

Assoziation und Navigation: Beispiel



Beispiel: „Abflugort eines Fluges ist immer ungleich Ziel.“

Beispiel

context Flight

inv: origin <> destination

Beispiel: „Abflugort eines Fluges ist immer Duisburg.“

Beispiel

context Flight

inv: origin.name = 'Duisburg'

Collection-Operationen

- Assoziationen: Insbes. **one-to-many** oder **many-to-many** Beziehungen.
  => Navigation zu Assoziationsende resultiert in Collections.
- OCL Ausdrücke geben entweder Fakt über **alle** Objekte in Collection an oder über Collection **selbst**.
- Beschränkungen beziehen sich oft nur auf Teil einer Collection.
  => Mit Collection-Operationen auswählen:
  - **Pfeil** (→) spezifiziert Nutzung vordefinierter Operation für Collections. Zum Beispiel für Klasse *Passenger* und Größe einer Collection *size()*: *Passenger* → *size()*.
- Zur **Abgrenzung:** Verwendung von Operation aus UML-Modell (z.B. *departTime.isBefore(arrivalTime)*).

Collection-Operationen: Beispiel *size()*

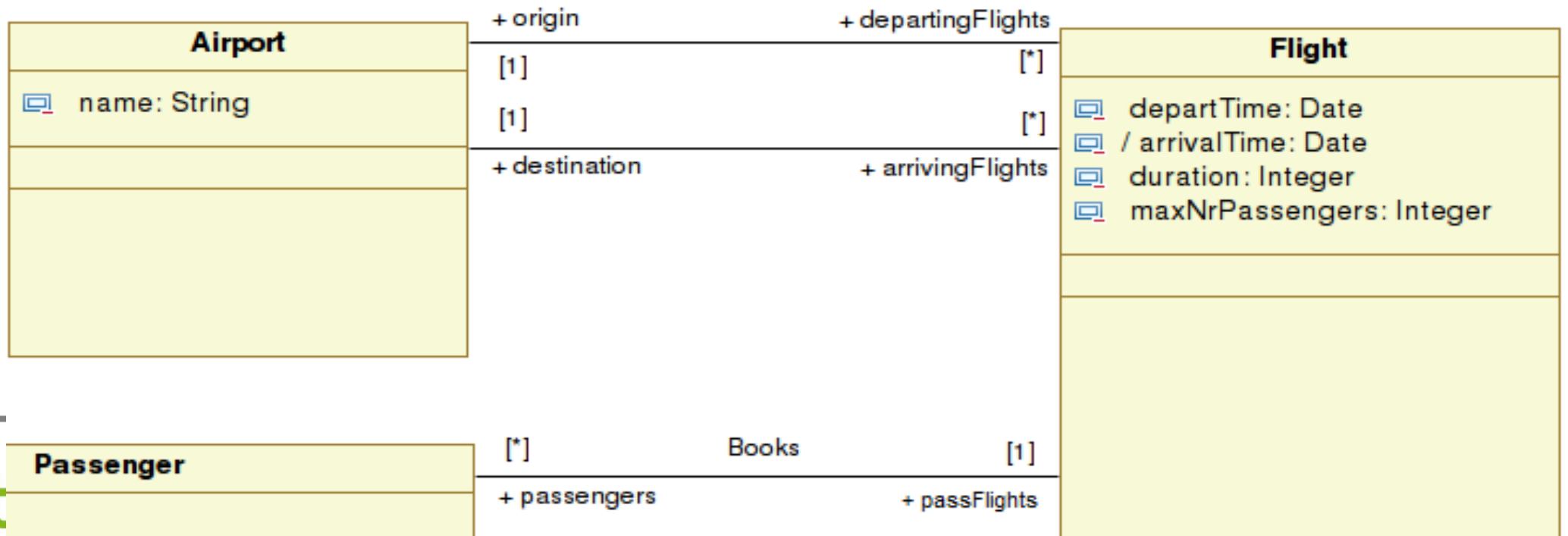
LEHRSTUHL 14
SOFTWARE ENGINEERING

- *Collection* → *size()* spezifiziert Größe von *Collection*.

Beispiel: Anzahl der Passagiere eines Fluges ist kleiner oder gleich der vorgegebenen maximalen Anzahl Passagiere.

Beispiel

context



Collection-Operationen: Beispiel *size()*

LEHRSTUHL 14
SOFTWARE ENGINEERING

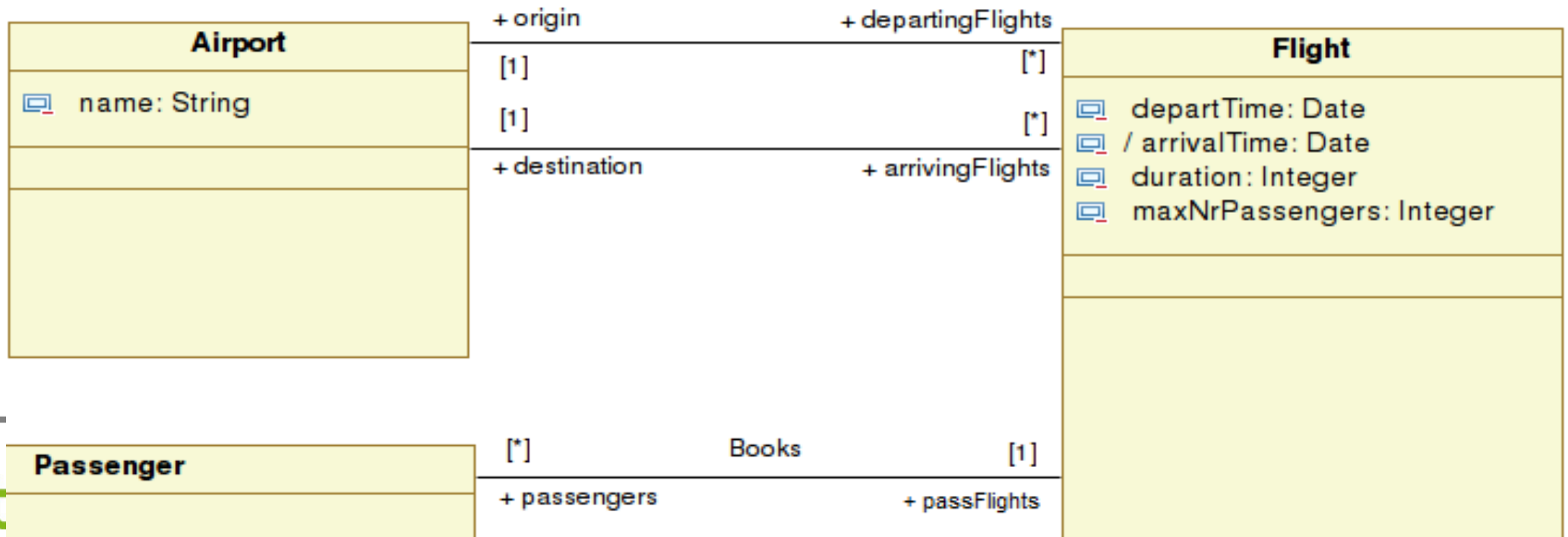
- *Collection* → *size()* spezifiziert Größe von *Collection*.

Beispiel: Anzahl der Passagiere eines Fluges ist kleiner oder gleich der vorgegebenen maximalen Anzahl Passagiere.

Beispiel

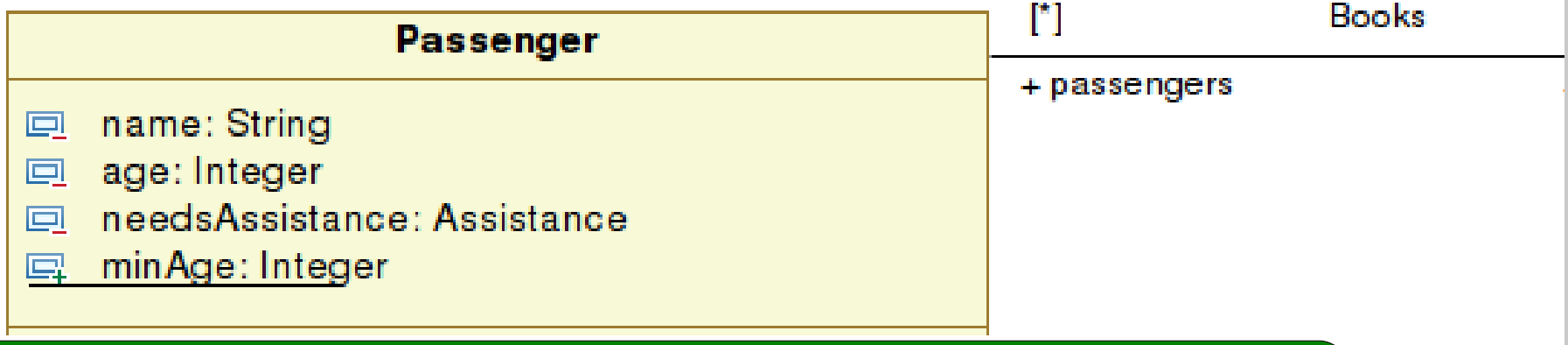
context Flight

inv: passengers → size() <= maxNrPassengers



Collection-Operationen: *collect()* - Attribute

collect(): Operation, um **Attributwerte** zu **sammeln**, z.B.:
passengers → *collect(name).*



Bedeutung (in Pseudocode)

```
Collection<String> c = new Collection();
foreach (p: passengers) {c.add(p.name);}
return c;
```

Per Definition erzeugt *collect()* Multimengen / Bags (d.h. Elemente können mehrfach auftreten).

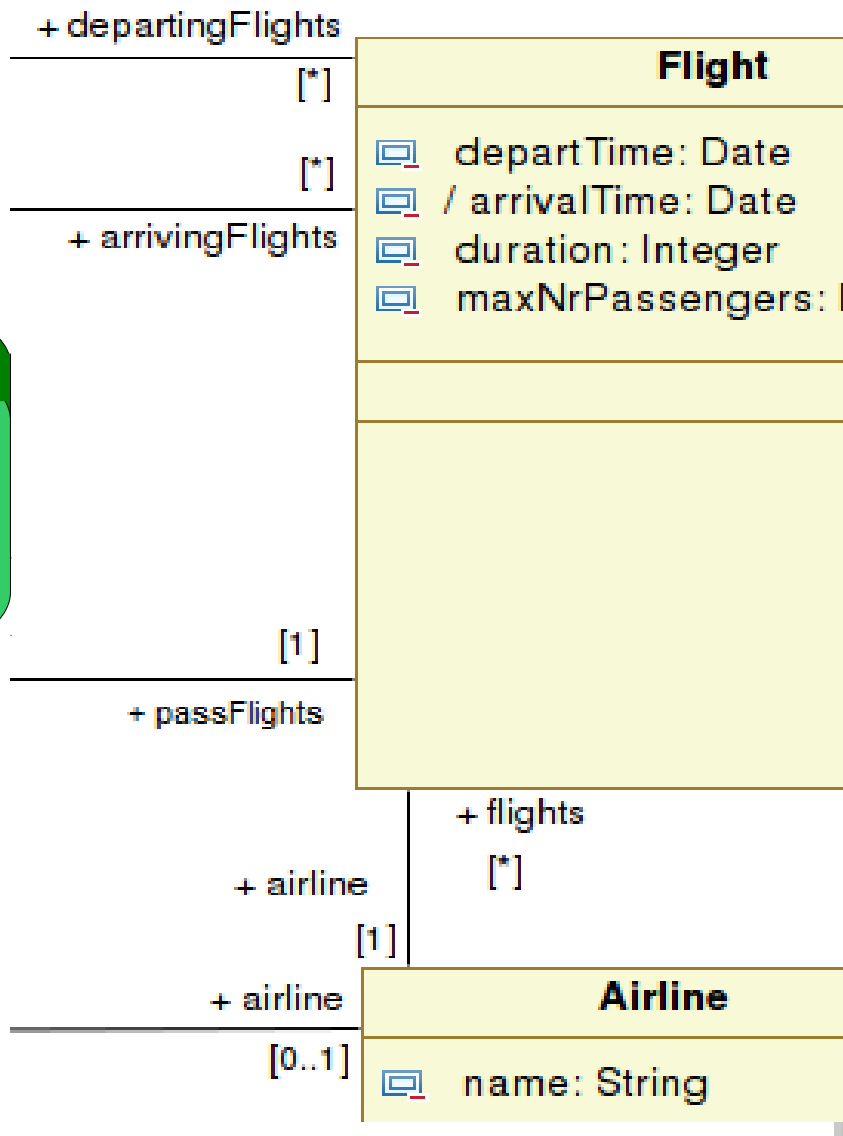
Punktnotation als verkürzte Schreibweise, z.B. passengers.name
(nicht verwechseln mit Attributzugriff !).

Collection-Operationen: *collect()* - Objekte

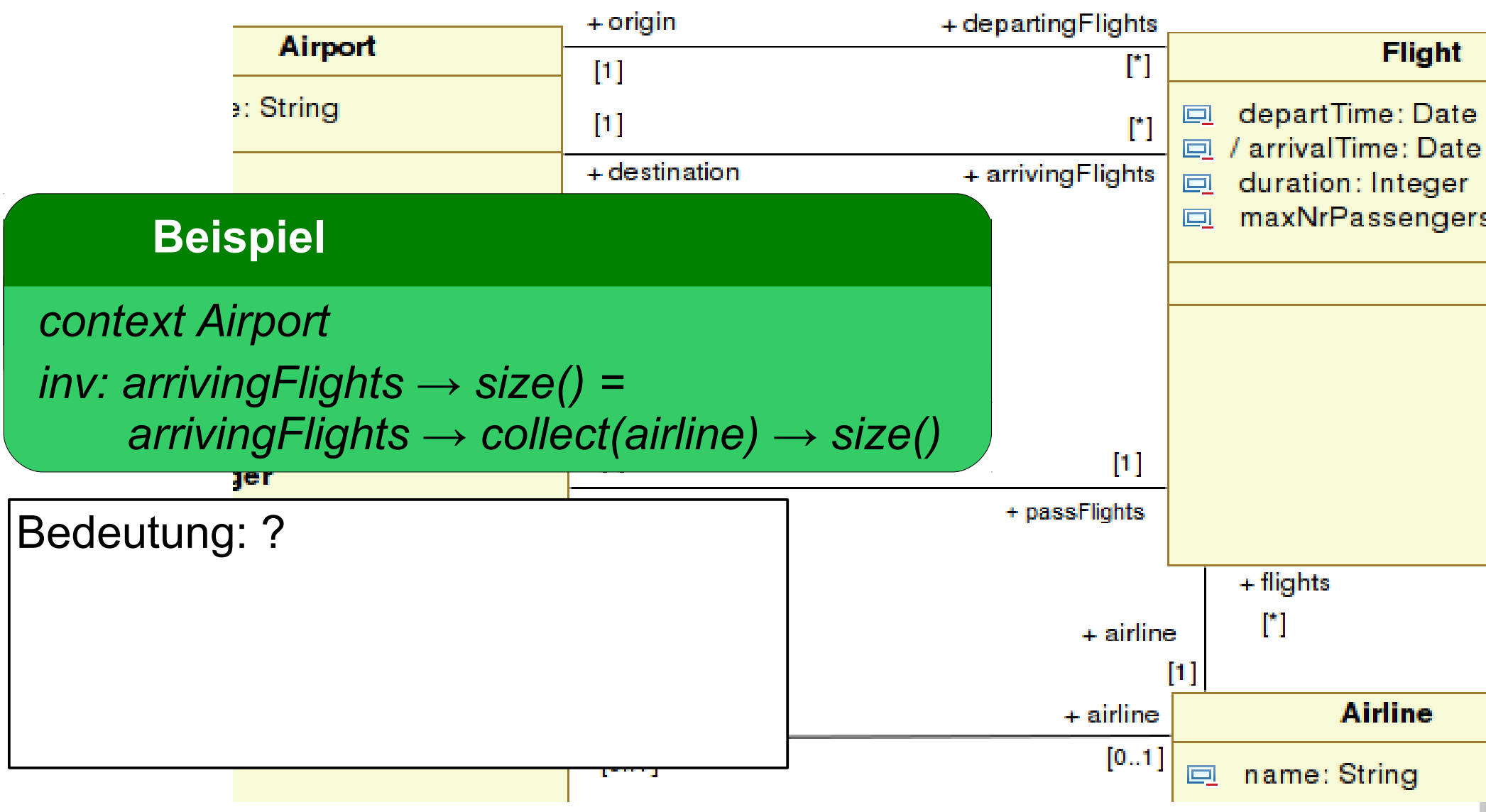
Kann benutzt werden, um neue Collections aus Objekten am Ende der Assoziation zu bilden, z.B. *arrivingFlights → collect(airline).*

Bedeutung (in Pseudocode)

```
Collection<Airline> c = new Collection();
foreach (f: arrivingFlights) {c.add(f.airline);}
return c;
```



collect(): Beispiel



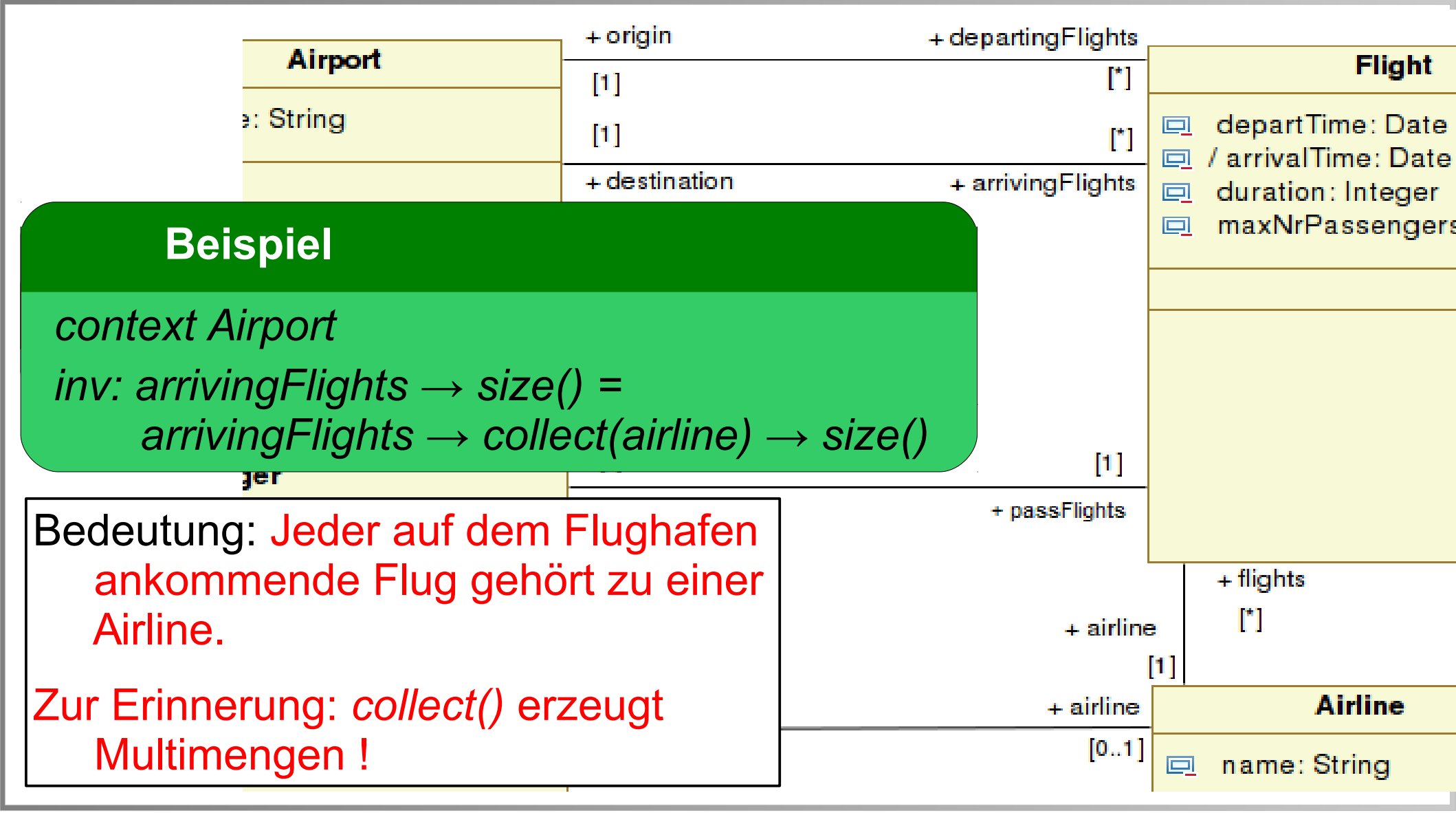
Beispiel

context Airport

inv: arrivingFlights → size() =
arrivingFlights → collect(airline) → size()

Bedeutung: ?

collect(): Beispiel



Beispiel

context Airport

inv: arrivingFlights → size() =
arrivingFlights → collect(airline) → size()

Bedeutung: Jeder auf dem Flughafen ankommende Flug gehört zu einer Airline.

Zur Erinnerung: *collect()* erzeugt Multimengen !

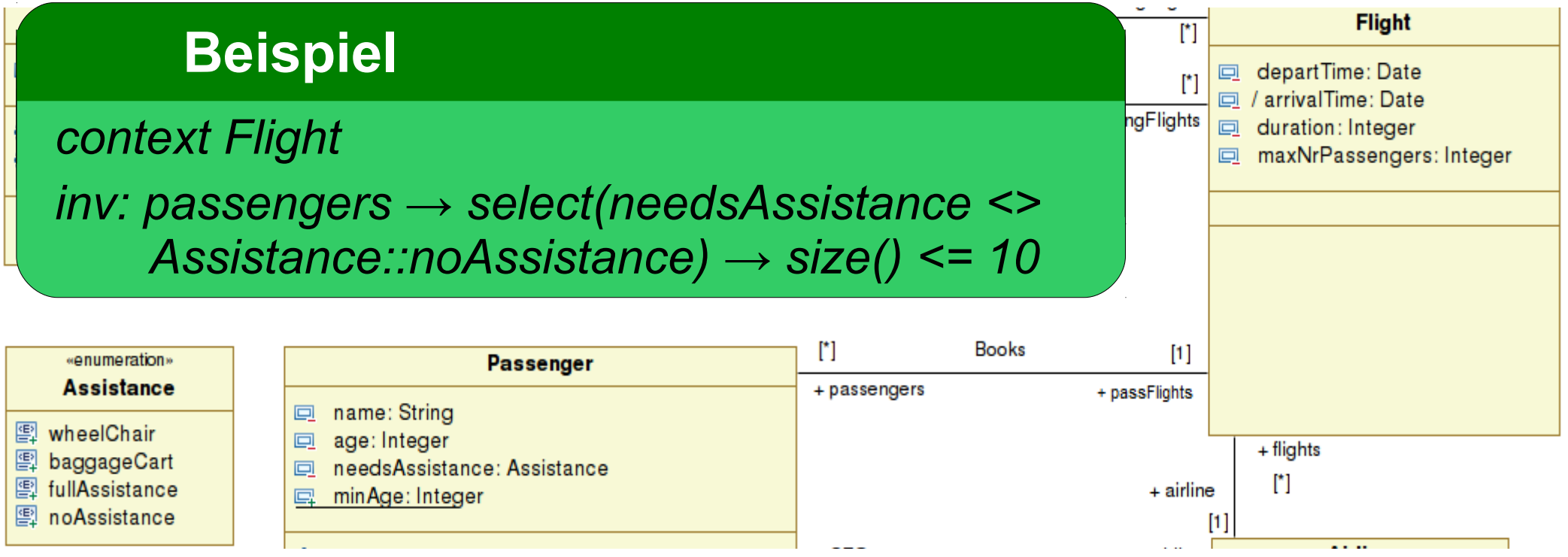
Operation *select()*

- Bekommt OCL-Ausdruck als Parameter übergeben.
- **Ergebnis: Subcollection** der verwendeten Collection.
- Liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist.

Beispiel

context Flight

inv: passengers → select(needsAssistance <> Assistance::noAssistance) → size() <= 10



- Bedeutung: ?

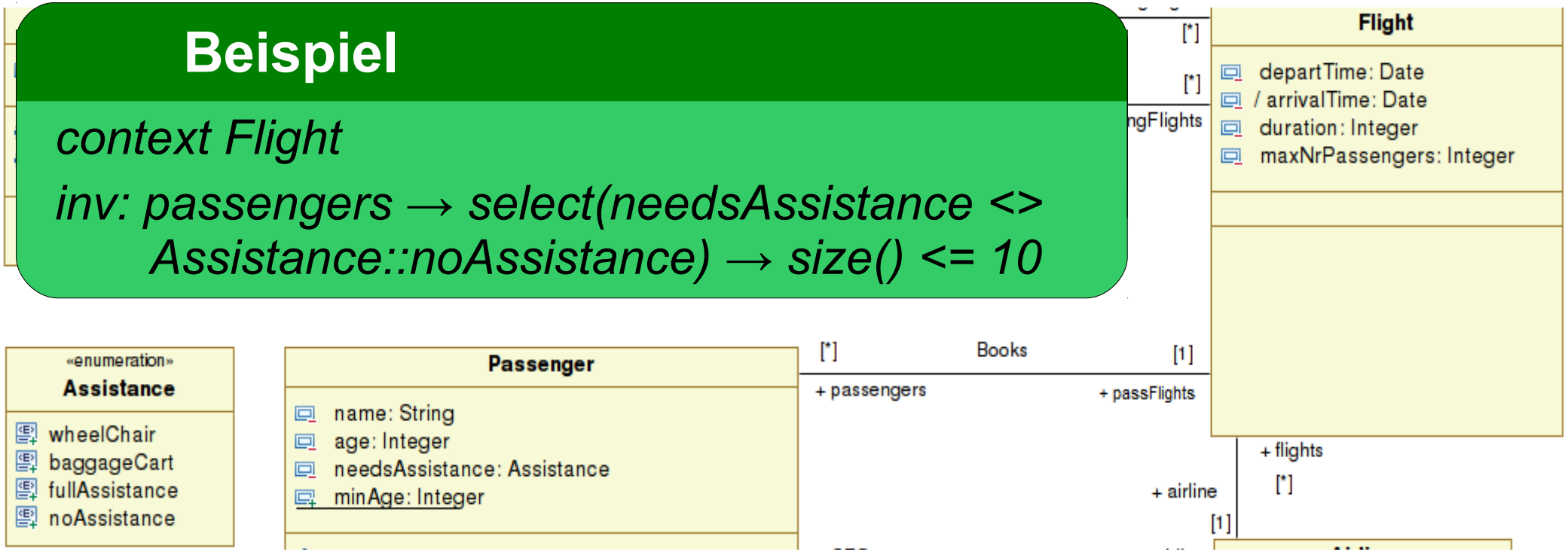
Operation *select()*

- Bekommt OCL-Ausdruck als Parameter übergeben.
- **Ergebnis: Subcollection** der verwendeten Collection.
- Liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist.

Beispiel

context Flight

inv: passengers → select(needsAssistance <> Assistance::noAssistance) → size() <= 10



- Bedeutung: Anzahl der Passagiere eines Fluges, die Hilfe brauchen, ist kleiner oder gleich 10.

Operation *reject()*

- Verhält sich mengentheoretisch komplementär zu *select()*.
- Liefert alle Elemente einer Collection, für die der Ausdruck **falsch** ist.

Beispiel

context Flight

inv: passengers → select(needsAssistance <>
Assistance::noAssistance) → size() <= 10

- Beispiel: Anzahl der Passagiere, die Hilfe brauchen, ist kleiner oder gleich 10. Wie mit *reject()* spezifizieren ?

Beispiel

context Flight

Operation *reject()*

- Verhält sich mengentheoretisch komplementär zu *select()*.
- Liefert alle Elemente einer Collection, für die der Ausdruck **falsch** ist.

Beispiel

context Flight

inv: passengers → select(needsAssistance <>
Assistance::noAssistance) → size() <= 10

- Beispiel: Anzahl der Passagiere, die Hilfe brauchen, ist kleiner oder gleich 10. Wie mit *reject()* spezifizieren ?

Beispiel

context Flight

inv: passengers → reject(needsAssistance =
Assistance::noAssistance) → size() <= 10

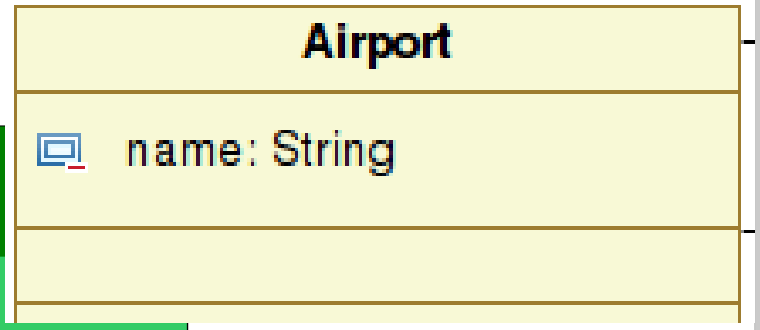
Operation *forAll()*

- Nutzbar, um **Bedingung** zu **definieren.**

  → Muss von allen Elementen in Collection eingehalten werden.

- **Erhält OCL-Ausdruck** als Parameter.
- Wird verwendet, wenn es (Sub-)Set von allen Instanzen einer Klasse gibt und dieses überprüft werden soll.
- **Liefert booleschen Wert** zurück:
  - Wahr, wenn Bedingung von allen Elementen erfüllt wird.
  - Falsch, sonst.
- *class.allInstances():* Collection mit allen Elementen einer Klasse.

Operation *forAll()*: Beispiel

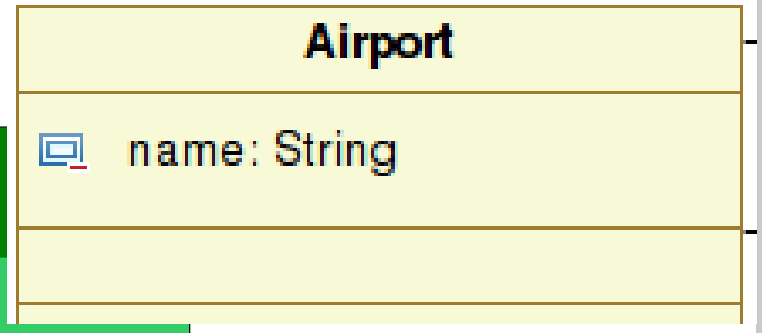


Beispiel

context Airport
inv: Airport.allInstances() → forAll (a1, a2 |
a1 <> a2 implies a1.name <> a2.name)

Bedeutung: *?*

Operation *forAll()*: Beispiel



Beispiel

context Airport
inv: Airport.allInstances() → forAll (a1, a2 |
a1 <> a2 implies a1.name <> a2.name)

Bedeutung: Jeder Flughafenname ist einzigartig.

Äquivalent mit Operation *isUnique()*:

context Airport
inv : Airport.allInstances() → isUnique(name)

1.1 OCL

- Motivation & Einführung
- Assoziationen, Navigationen, Operationen
- **Vor- und Nachbedingungen**
- Anhang: OCL Typen

Vor- und Nachbedingungen

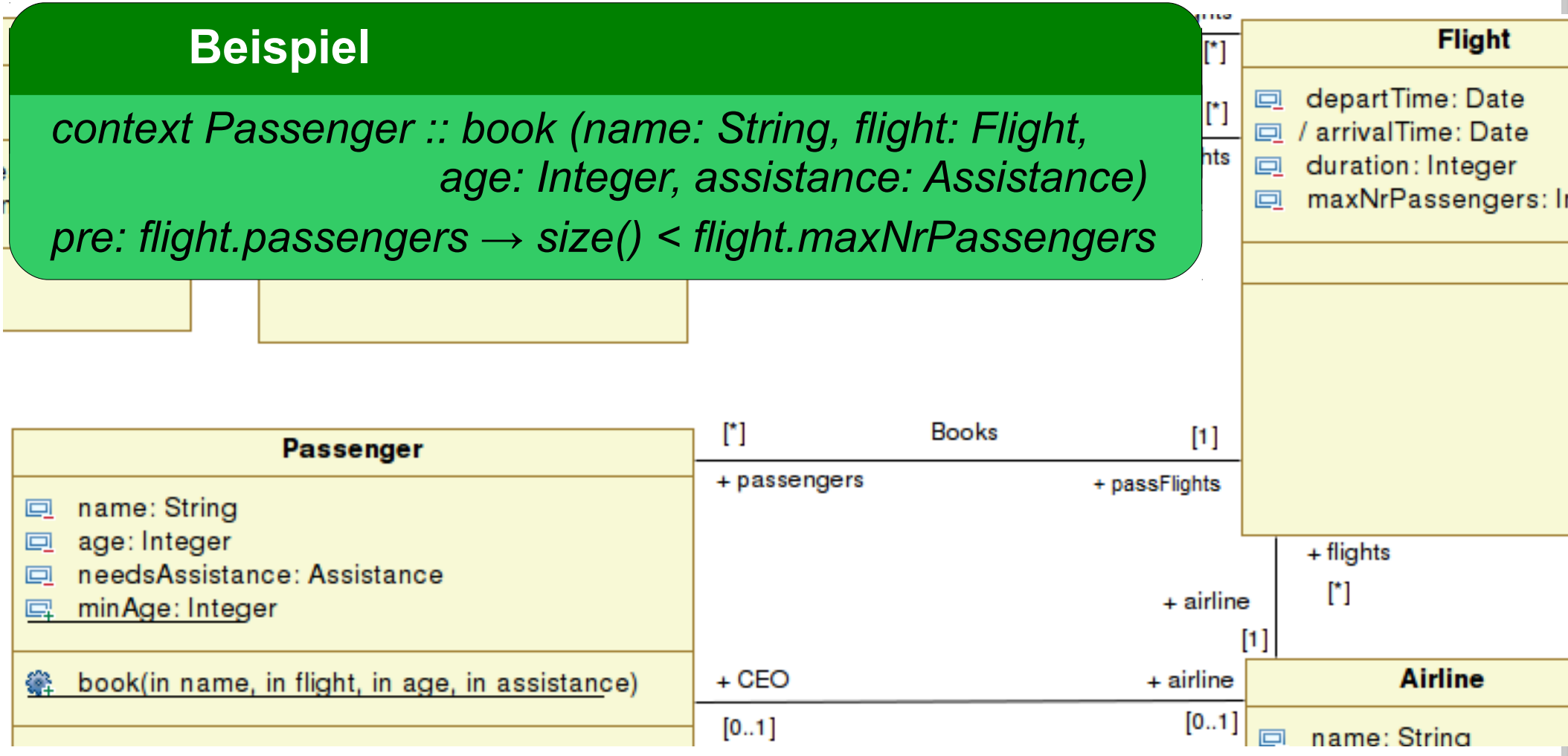
- In Klassendiagrammen nur Syntax und Signatur einer Operation definierbar.
- **Semantik** einer Operation mittels **Vor- und Nachbedingungen** in OCL spezifizierbar.
- Vorbedingung:
  - Bedingungen von Argumenten und dem initialen Objektzustand müssen erfüllt werden.

Vorbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

pre: flight.passengers → size() < flight.maxNrPassengers



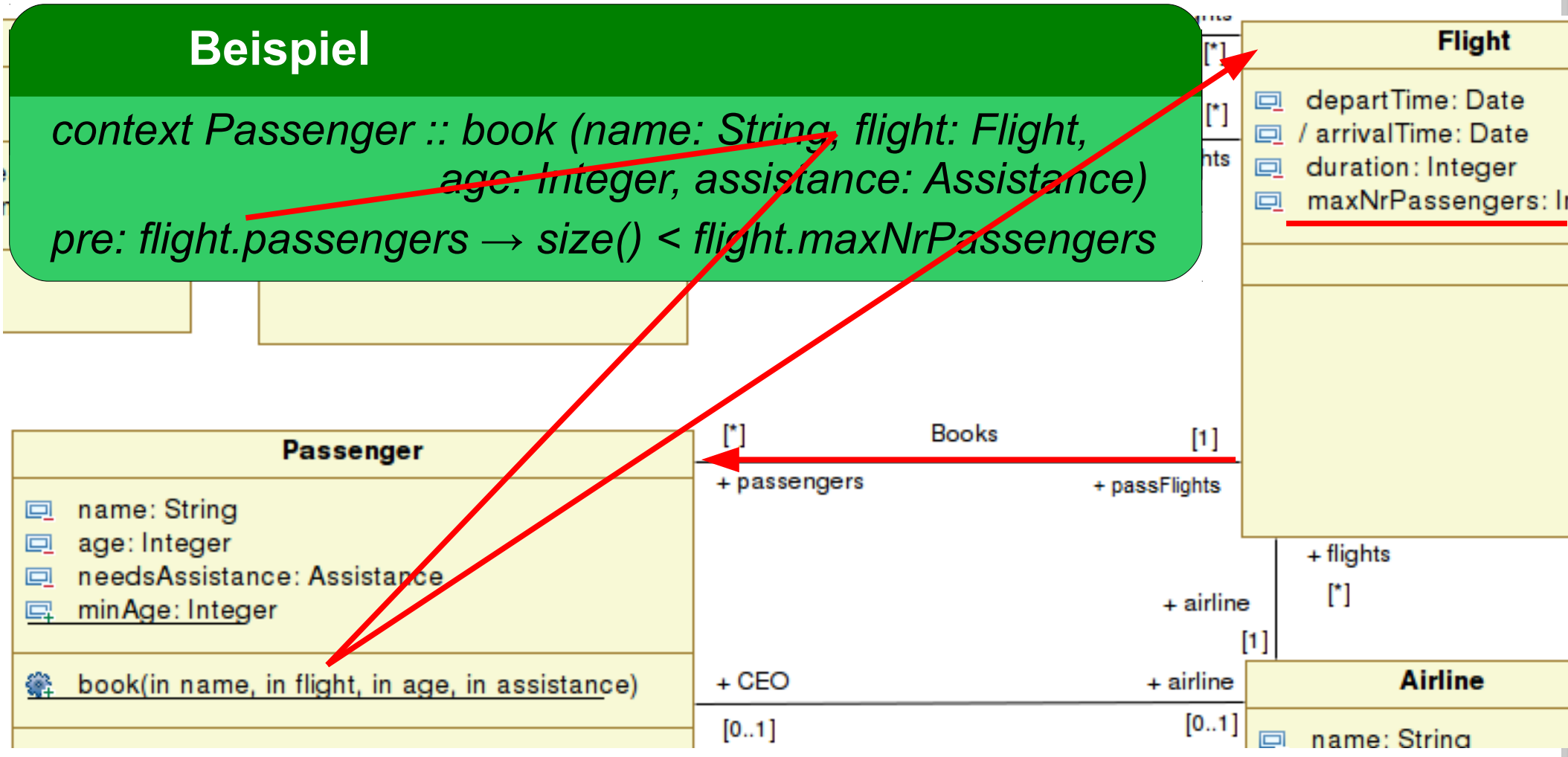
Bedeutung: ?

Vorbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

pre: flight.passengers → size() < flight.maxNrPassengers



Bedeutung: Vor Buchung von Passagier auf Flug muss Anzahl registrierter Passagiere für diesen Flug kleiner als maximale Anzahl für den Flug sein.

Nachbedingungen

Nachbedingung:

- Bedingungen, die **am Ende einer Operationsausführung** vom Rückgabewert, finalem Objektzustand, Argumenten und initialem Objektzustand erfüllt sein müssen.
- In Annahme, dass Vorbedingungen erfüllt sind.
- **Spezifiziert** beabsichtigte Ergebnisse und **Zustandsänderungen** (was), aber nicht wie sie geschehen (wie).
- Kann initialen Zustand eines Objektfelds mit Postfix-Notation *@pre* referenzieren (z.B. *flight.passengers@pre*).
- Kann Rückgabewert mit Schlüsselwort *result* referenzieren.

Nachbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

post: flight.passengers → size() - flight.passengers@pre → size() = 1
and
flight.passengers → exists (p: Passenger | p.age = age and p.name = name and p.needsAssistance = assistance)

Bedeutung: ?

Nachbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

post: flight.passengers → size() - flight.passengers@pre → size() = 1
and
flight.passengers → exists (p: Passenger | p.age = age and p.name = name and p.needsAssistance = assistance)

Bedeutung:

Nach Ausführung von *book()*:

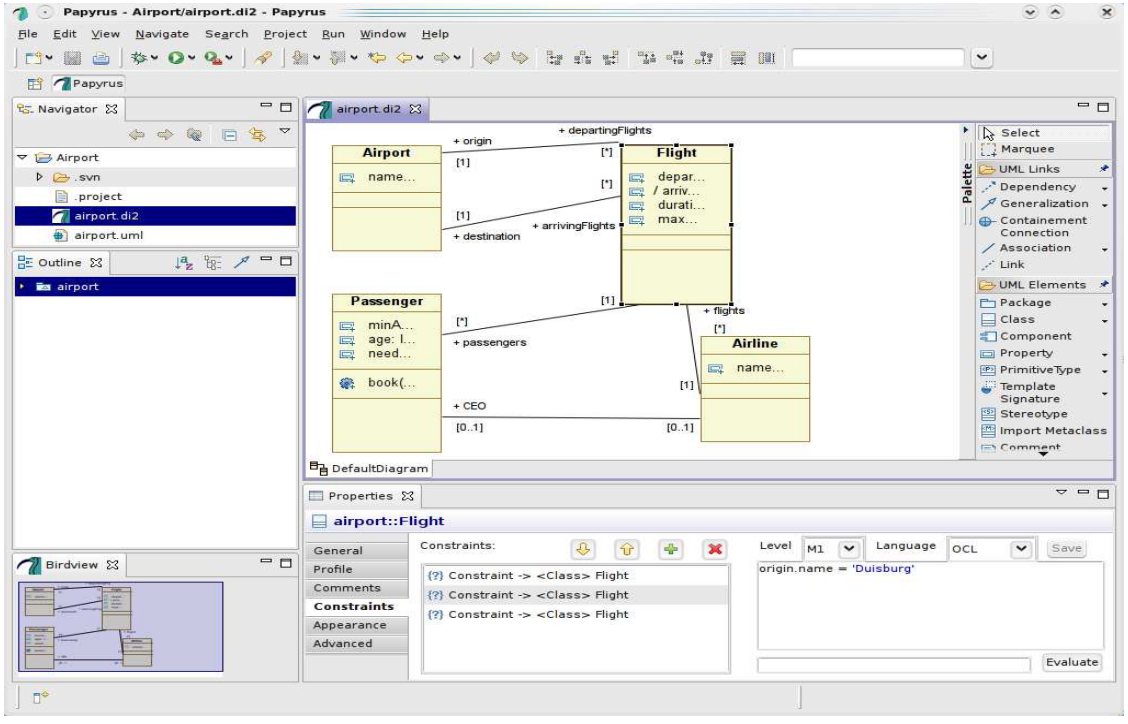
- erreicht die Assoziation *passengers* ein zusätzliches Objekt und
- existiert ein *Passenger*-Objekt, dessen Attribute die Werte aus *book* enthalten.

Werkzeugunterstützung

Verschiedene UML-Editoren unterstützen Einbindung von OCL-Bedingungen.

Zum Beispiel: Papyrus.

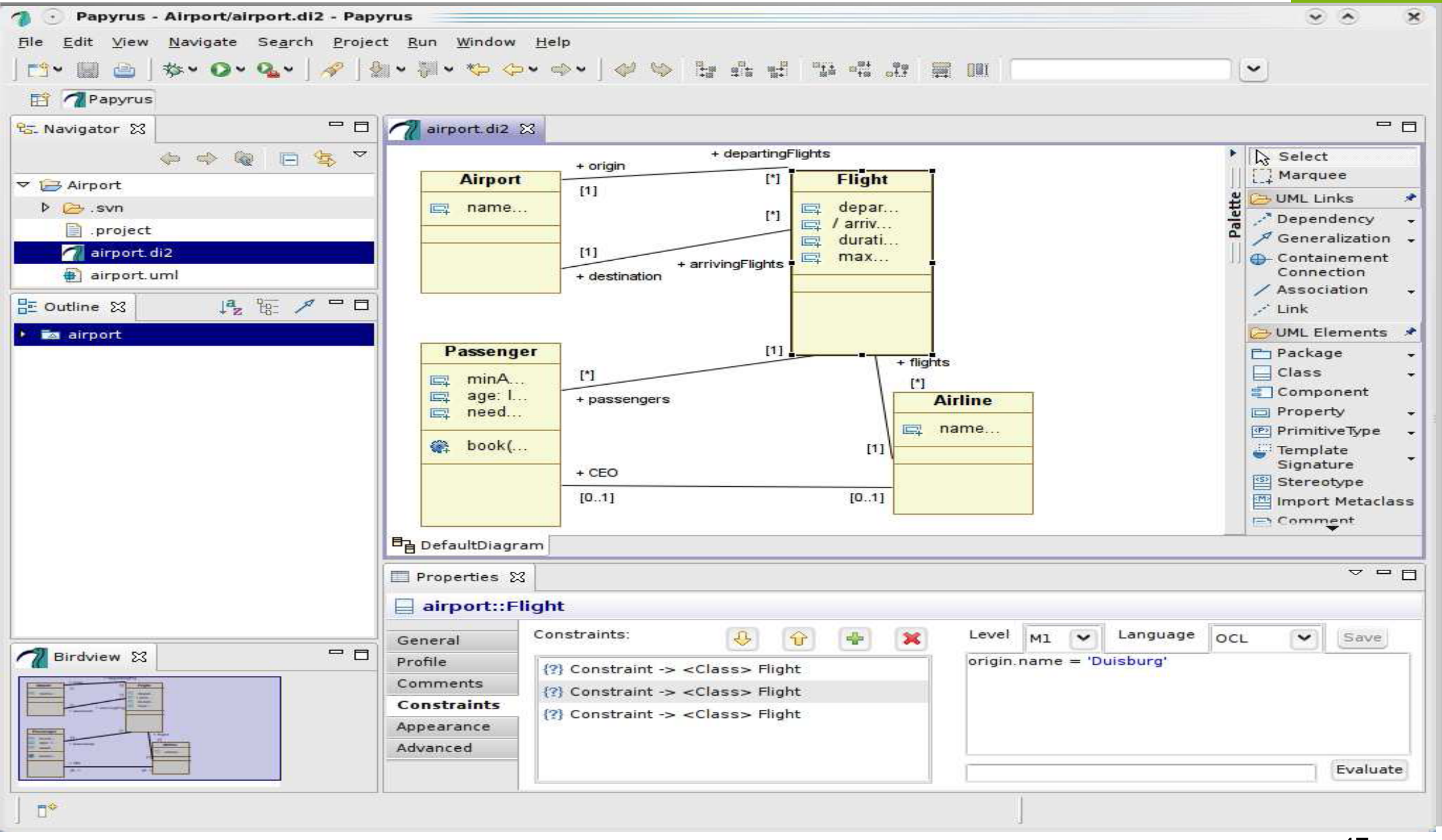
- Frei erhältliches Open-Source Werkzeug für Modellierung mit UML 2.0, s. http://www.papyrusuml.org/
- Basiert auf Entwicklungs-umgebung Eclipse.



- Erweiterbare Architektur von Papyrus erlaubt Hinzufügen von Diagrammen, neuen Codegeneratoren, etc.
- Erlaubt Einbinden von OCL-Bedingungen.

Flughafen-Beispiel in Papyrus

LEHRSTUHL 14
SOFTWARE ENGINEERING



Zusammenfassung: Object Constraint Language (OCL)

Logik-basierte Notation für **Einschränkungen** in **UML**-Modellen.

- **Bedingungen** an Ausführung der modellierten Systemteile formulieren und analysieren: **Invarianten, Collections, Vor-** und **Nachbedingungen, Guard Conditions**.
- Erlaubt **Qualitätssicherung** der Software bereits im Design auf **Modellebene**, wo Fehler kostengünstig repariert werden können.

Vorteile:

- Automatische Verifikation der Bedingung.
- **Werkzeuge** erzeugen aus OCL **Assertions** in **Java**.

Nächster Abschnitt: Verwendung von OCL im Kontext der modellbasierten Softwareentwicklung mit UML.

- Zum Beispiel: Verwendung der OCL im UML-Metamodell für Definition von **Wohlgeformtheit** von UML-Modellen.

1.1 OCL: Anhang

Weitere Informationen zum selbständigen Nachlesen / Nachschlagen.

1.1 OCL

- Motivation & Einführung
- Assoziationen, Navigationen, Operationen
- Vor- und Nachbedingungen
- Anhang: OCL Typen

Literatur

Literatur:

• V. Gruhn, D. Pieper, C. Röttgers: **MDA - Effektives Software-Engineering mit UML 2 und Eclipse**. Xpert.press / Springer-Verlag, 2006.
UB e-Book: http://www.ub.tu-dortmund.de/katalog/titel/1223129 .

- Kapitel 6.3.

• J. Seemann, J.W. Gudenberg: **Software-Entwurf mit UML 2**. Xpert.press / Springer-Verlag, 2006.
UB e-Book: http://www.ub.tu-dortmund.de/katalog/titel/1223020.

- Kapitel 14.5.

• J. Warmer, A. Kleppe: ***The Object Constraint Language: Getting Your Models Ready for MDA***. *Addison-Wesley Longman Publ. & Co.,Inc., 2003.*
UB: http://www.ub.tu-dortmund.de/katalog/titel/901443
http://www.ub.tu-dortmund.de/katalog/titel/787903

Primitive Typen

Type	Description	Values	Operators and Operations
Boolean		true, false	=, <>, and, or, xor, not, implies, if-then-else-endif (note 2)
Integer	A whole number of any size	-1, 0, 1, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), / (real), abs(), max(b), min(b), mod(b), div(b)
Real	A real number of any size	1.5, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), /, abs(), max(b), min(b), round(), floor()
String	A string of characters	'a', 'John'	=, <>, size(), concat(s2), substring(lower, upper) (1<=lower<=upper<=size), toReal(), toInteger()

Notes:
1) Operations indicated with parenthesis are applied with ".", but the parenthesis may be omitted.
2) Example: title = (if isMale then 'Mr.' else 'Ms.' endif)

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Wiederholung: Arten von Kollektionen

- Collection von Objekten:
  - **Set**:
    - Jedes Element kommt nur einmal vor.
    - Einfaches Navigieren einer Assoziation liefert Set zurück.
  - **Bag**:
    - Gleiche Elemente dürfen mehrmals vorkommen.
  - **OrderedSet**:
    - Satz von geordneten Elementen.
  - **Sequence**:
    - Bag in dem Elemente geordnet sind.

Collections und Tuples

Description	Syntax	Examples
Abstract collection of elements of type T	**Collection(T)**	
Unordered collection, no duplicates	**Set(T)**	Set{1 , 2}
Ordered collection, duplicates allowed	**Sequence(T)**	Sequence {1, 2, 1} Sequence {1..4} (same as {1,2,3,4})
Ordered collection, no duplicates	**OrderedSet(T)**	OrderedSet {2, 1}
Unordered collection, duplicates allowed	**Bag(T)**	Bag {1, 1, 2}
Tuple (with named parts)	**Tuple(field1: T1, ... fieldn : Tn)**	Tuple {age: Integer = 5, name: String = 'Joe' } Tuple {name = 'Joe', age = 5}

Note 1: They are *value types*: "=" and "<>" compare values and not references.

Note 2: Tuple components can be accessed with "." as in "t1.name"

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on Collection (T)

Operation	Description
size(): Integer	The number of elements in this collection (*self*)
isEmpty(): Boolean	size = 0
notEmpty(): Boolean	size > 0
includes(object: T): Boolean	True if *object* is an element of *self*
excludes(object: T): Boolean	True if *object* is not an element of *self*
count(object: T): Integer	The number of occurrences of *object* in *self*
includesAll(c2: Collection(T)): Boolean	True if *self* contains all the elements of *c2*
excludesAll(c2: Collection(T)): Boolean	True if *self* contains none of the elements of *c2*
sum(): T	The addition of all elements in *self* (T must support "+")
product(c2: Collection(T2)) : Set(Tuple(first:T, second:T2))	The cartesian product operation of *self* and *c2*.

Note: Operations on collections are applied with "->" and not "."

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Iterator Expressions on Collection (T)

Iterator expression	Description
iterate(iterator: T; accum: T2 = init | body) : T2	Returns the final value of an accumulator that, after initialization, is updated with the value of the *body* expression for every element in the *source* collection.
exists(iterators | body) : Boolean	True if *body* evaluates to true for at least one element in the *source* collection. Allows multiple iterator variables.
forAll(iterators | body): Boolean	True if *body* evaluates to true for each element in the source collection. Allows multiple iterator variables.
one(iterator| body): Boolean	True if there is exactly one element in the *source* collection for which *body* is true
isUnique(iterator| body): Boolean	Results in true if *body* evaluates to a different value for each element in the *source* collection.
any(iterator | body): T	Returns any element in the source collection for which *body* evaluates to true. The result is null if there is none.
collect(iterator| body): Collection(T2)	The Collection of elements resulting from applying *body* to every member of the *source* set.

Note: The iterator variable declaration can be omitted when there is no ambiguity.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Iterator expression on Collection (T)

Iterator expression	Description
select(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is true. The result collection is of the same type of the *source* collection.
reject(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is false. The result collection is of the same type of the *source* collection.
collectNested(iterator | body): CollectionWithDuplicates(T2)	The Collection of elements (allowing duplicates) that results from applying *body* (of type T2) to every member of the *source* collection. The result is not flattened. Conversions: Set -> Bag, OrderedSet -> Sequence.
sortedBy(iterator | body): OrderedCollection(T)	Returns an ordered Collection of all the elements of the *source* collection by ascending order of the value of the *body* expression. The type T2 of the *body* expression must support "<". Conversions: Set -> OrderedSet, Bag -> Sequence.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on Set (T) I

Operation	Description
=(s: Set(T)) : Boolean	Do *self* and *s* contain the same elements?
union(s: Set(T)): Set(T)	The union of *self* and *s*.
union(b: Bag(T)): Bag(T)	The union of *self* and bag *b*.
intersection(s: Set(T)): Set(T)	The intersection of *self* and *s*.
intersection(b: Bag(T)): Set(T)	The intersection of *self* and *b*.
-(s: Set(T)) : Set(T)	The elements of *self*, which are not in *s*.
including(object: T): Set(T)	The set containing all elements of *self* plus *object*.
excluding(object: T): Set(T)	The set containing all elements of *self* minus *object*.
symmetricDifference(s: Set(T)): Set(T)	The set containing all the elements that are in *self* or *s*, but not in both.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on Set (T) II

Operation	Description
flatten() : Set(T2)	If T is a collection type, the result is the set with all the elements of all the elements of *self*; otherwise, the result is *self*.
asOrderedSet(): OrderedSet(T)	OrderedSet with elements from *self* in undefined order.
asSequence(): Sequence(T)	Sequence with elements from *self* in undefined order.
asBag(): Bag(T)	Bag will all the elements from *self*.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on Bag (T)

Operation	Description
=(bag: Bag(T)) : Boolean	True if *self* and *bag* contain the same elements, the same number of times.
union(bag: Bag(T)): Bag(T)	The union of *self* and *bag*.
union(set: Set(T)): Bag(T)	The union of *self* and *set*.
intersection(bag:Bag(T)): Bag(T)	The intersection of *self* and *bag*.
intersection(set: Set(T)): Set(T)	The intersection of *self* and *set*.
including(object: T): Bag(T)	The bag with all elements of *self* plus *object*.
excluding(object: T): Bag(T)	The bag with all elements of *self* without *object*.
flatten() : Bag(T2)	If T is a collection type: bag with all the elements of all the elements of *self*; otherwise: *self*.
asSequence(): Sequence(T)	Seq. with elements from *self* in undefined order.
asSet(): Set(T)	Set with elements from *self*, without duplicates.
asOrderedSet(): OrderedSet(T)	OrderedSet with elements from *self* in undefined order, without duplicates.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on Sequence (T) I

Operation	Description
=(s: Sequence(T)) : Boolean	True if *self* contains the same elements as *s*, in the same order.
union(s: Sequence(T)): Sequence(T)	The sequence consisting of all elements in *self*, followed by all elements in *s*.
flatten() : Sequence(T2)	If T is a collection type, the result is the set with all the elements of all the elements of *self*; otherwise, it's *self*.
append(object: T): Sequence(T)	The sequence with all elements of *self*, followed by *object*.
prepend(obj: T): Sequence(T)	The sequence with *object*, followed by all elements in *self*.
insertAt(index : Integer, object : T) : Sequence(T)	The sequence consisting of *self* with *object* inserted at position *index* (1<=index<=size+1)
subSequence(lower : Integer, upper: Integer) : Sequence(T)	The sub-sequence of *self* starting at index *lower*, up to and including index *upper* (1<=lower<=upper<=size)

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on Sequence (T) II

Operation	Description
at(i : Integer) : T	The *i*-th element of *self* (1<=i<=size)
indexOf(object : T) : Integer	The index of *object* in *self*.
first() : T	The first element in *self*.
last() : T	The last element in *self*.
including(object: T): Sequence(T)	The sequence containing all elements of *self* plus *object* added as last element
excluding(object: T): Sequence(T)	The sequence containing all elements of *self* apart from all occurrences of *object*.
asBag(): Bag(T)	The Bag containing all the elements from *self*, including duplicates.
asSet(): Set(T)	The Set containing all the elements from *self*, with duplicates removed.
asOrderedSet(): OrderedSet(T)	An OrderedSet that contains all the elements from *self*, in the same order, with duplicates removed.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations on OrderedSet (T)

Operation	Description
append(object: T): OrderedSet(T)	The set of elements, consisting of all elements of *self*, followed by *object*.
prepend(object: T): OrderedSet(T)	The sequence consisting of *object*, followed by all elements in *self*.
insertAt(index : Integer, object : T) : OrderedSet(T)	The set consisting of *self* with *object* inserted at position *index*.
subOrderedSet(lower : Integer, upper : Integer) : OrderedSet(T)	The sub-set of *self* starting at number *lower*, up to and including element number *upper* (1<=lower<=upper<=size).
at(i : Integer) : T	The *i*-th element of *self* (1<=i<=size).
indexOf(object : T) : Integer	The index of *object* in the sequence.
first() : T	The first element in *self*.
last() : T	The last element in *self*.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Special Types

Type	Description
OclAny	Supertype for all types except for collection and tuple types. All classes in a UML model inherit all operations defined on OclAny.
OclVoid	The type OclVoid is a type that conforms to all other types. It has one single instance called *null*. Any property call applied on *null* results in *OclInvalid*, except for the operation oclIsUndefined(). A collection may have *null's*.
OclInvalid	The type OclInvalid is a type that conforms to all other types. It has one single instance called *invalid*. Any property call applied on *invalid* results in *invalid*, except for the operations oclIsUndefined() and oclIsInvalid().
OclMessage	Template type with one parameter T to be substituted by a concrete operation or signal type. Used in some postconditions that need to constrain the messages sent during the operation execution.
OclType	Meta type.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations defined in OclAny

Operation	Description
=(object2 : OclAny) : Boolean	True if *self* is the same object as *object2*.
<>(object2 : OclAny) : Boolean	True if *self* is a different object from *object2*.
oclIsNew() : Boolean	Can only be used in a postcondition. True if *self* was created during the operation execution.
oclAsType(t : OclType) : OclType	Cast (type conversion) operation. Useful for downcast.
oclIsTypeOf(t: OclType) : Boolean	True if *self* is of type *t*.
oclIsKindOf(t : OclType) : Boolean	True if *self* is of type *t* or a subtype of *t*.
oclIsInState(s : OclState) : Boolean	True if *self* is in state *s*.
oclIsUndefined() : Boolean	True if *self* is equal to *null* or *invalid*.
oclIsInvalid() : Boolean	True if *self* is equal to *invalid*.
allInstances() : Set(T)	Static operation that returns all instances of a classifier.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Operations defined in OclMessage

Operation	Description
hasReturned() : Boolean	True if type of template parameter is an operation call, and the called operation has returned a value.
result()	Returns the result of the called operation, if type of template parameter is an operation call, and the called operation has returned a value.
isSignalSent() : Boolean	Returns true if the OclMessage represents the sending of a UML Signal.
isOperationCall() : Boolean	Returns true if the OclMessage represents the sending of a UML Operation call.
parameterName	The value of the message parameter.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf