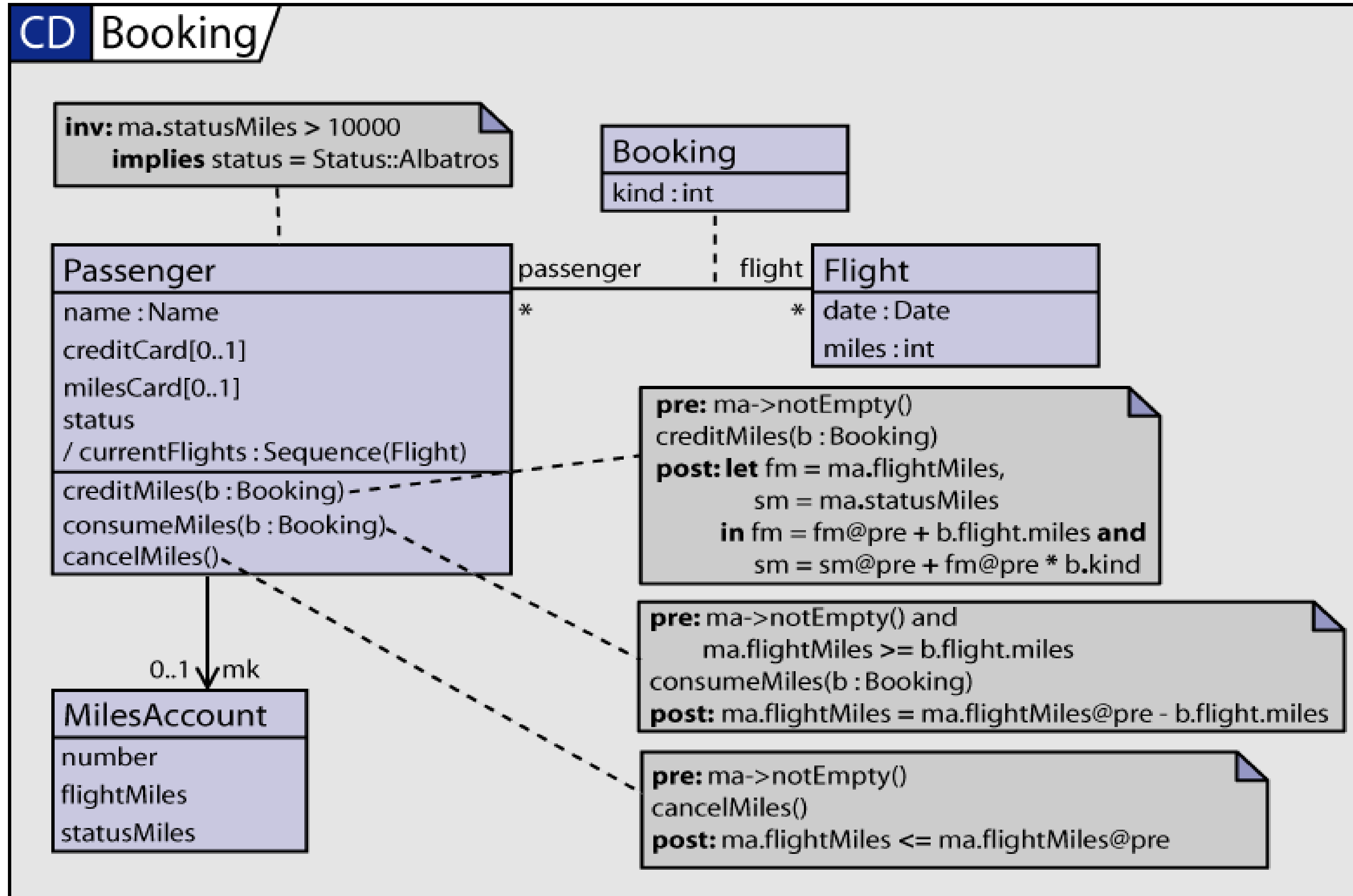
Object Constraint Language 2

A first glimpse



History and predecessors

- **Predecessors**
  - Model-based specification languages, like
    - Z, VDM, and their object-oriented variants; B
  - Algebraic specification languages, like
    - OBJ3, Maude, Larch

- Similar approaches in programming languages
  - ESC, JML

- **History**
  - developed by IBM as an easy-to-use formal annotation language
  - used in UML metamodel specification since UML 1.1
  - current version: OCL 2.3.1
    - specification: formal/2012-01-01

Usage scenarios

- Constraints on implementations of a model
  - invariants on classes
  - pre-/post-conditions for operations
    - cf. protocol state machines
  - body of operations
  - restrictions on associations, template parameters, …
- Formalization of side conditions
  - derived attributes
- Guards
  - in state machines, activity diagrams
- Queries
  - query operations
- **Model-driven architecture (MDA)/query-view-transformation (QVT)**

Language characteristics

- Integration with UML
  - access to classifiers, attributes, states, …
  - navigation through attributes, associations, …
  - limited reflective capabilities
  - model extensions by derived attributes
- **Side-effect free**
  - *not* an action language
  - only possibly describing effects
- **Statically typed**
  - inherits and extends type hierarchy from UML model
- Abstract and concrete syntax
  - precise definition new in OCL 2

Simple types

- Predefined primitive types
  - `Boolean` — `true`, `false`
  - `Integer` — `-17, 0, 3`
  - `Real` — `-17.89, 0.0, 3.14`
  - `String` — `“Hello”`
- Types induced by UML model
  - Classifier types, like
    - Passenger — no denotation of objects, only in `context`
  - Enumeration types, like
    - Status — `Status::Albatros, #Albatros`
  - Model element types
    - `OclModelElement`, `OclType`, `OclState`
- Additional types
  - `OclInvalid` — invalid (`OclUndefined`)
  - `OclVoid` — null
  - `OclAny` — top type of primitives and classifiers

Parameterized types

- Collection types
  - `Set(`T`)` sets
  - `OrderedSet(`T`)` like `Sequence` without duplicates
  - `Bag(`T`)` multi-sets
  - `Sequence(`T`)` lists
  - `Collection(`T`)` abstract
- Tuple types (records)
  - `Tuple(`a_1 `:` T_1`, …,` a_n `:` T_n`)`
- Message type
  - `OclMessage` for operations and signals

Examples

- `Set{Set{ 1 }, Set{ 2, 3 }} : Set(Set(Integer))`
- `Bag{1, 2.0, 2, 3.0, 3.0, 3} : Bag(Real)`
- `Tuple{x = 5, y = false} : Tuple(x : Integer, y : Boolean)`

Type hierarchy

- Type conformance (reflexive, transitive relation ≤)
  - `OclVoid` $\leq T$ — for all types T but `OclInvalid`
  - `OclInvalid` $\leq T$ — for all types T
  - `Integer` ≤ `Real`
  - $T \leq T' \Rightarrow C(T) \leq C(T')$ — for collection type C
  - $C(T) \leq$ `Collection(`T`)` — for collection type C
  - generalization hierarchy from UML model
  - $B \leq$ `OclAny` — for all primitives and classifiers B

Counterexample

  - ¬(`Set(OclAny)` ≤ `OclAny`)

- Casting
  - v`.oclAsType(`T`)` if $v : T'$ and ($T \leq T'$ or $T' \leq T$)
  - upcast necessary for accessing overridden properties

Expressions

- Local variable bindings

```
let x = 1 in x+2
```

- Iteration

  `c->iterate(` $i : T; a : T' = e' \mid e$ `)`

source collection

iteration variable
(bound to current value in *c*)

accumulator with initial value *e'*
(gathers result, returned after iteration)

iteration expression
(using variables *i* and *a*)

Example:

```
Set{1, 2}->iterate(i : Integer; a : Integer = 0 | a+i) = 3
```

- Many operations on collections are **reduced** to `iterate`

Expressions: Standard library (1)

- Operations on primitive types (written: v.op(...))
  - operations without () are mixfix

OclAny	=, <>, oclIsTypeOf(T), oclIsKindOf(T), ...
Boolean	**and**, **or**, **xor**, **implies**, **not**
Integer	+, -, *, /, div(i), mod(i), ...
Real	+, -, *, /, floor(), round(), ...
String	size(), concat(s), substring(l, u), ...

- Operations on collection types (written: v->op(...))

Collection	size(), includes(v), isEmpty(), ...
Set	union(s), including(v), flatten(), asBag(), ...
OrderedSet	append(s), first(), at(i), ...
Bag	union(b), including(v), flatten(), asSet(), ...
Sequence	append(s), first(), at(i), asOrderedSet(), ...

Expressions: Standard library (2)

- **Finite quantification**
  - c`->forAll(`i : T | e`)` = c`->iterate(`i : T; a : `Boolean = true` | a **and** e`)`
  - c`->exists(`i : T | e`)` = c`->iterate(`i : T; a : `Boolean = false` | a **or** e`)`
- Selecting values
  - c`->any(`i : T | e`)` — some element of c satisfying e
  - c`->select(`i : T | e`)` — all elements of c satisfying e
- Collecting values
  - c`->collect(`i : T | e`)` — collection of elements with e applied to each element of c
  - c`.`p — collection of elements $v.p$ for each v in c (short-hand for `collect`)

C`.allInstances()`	all current instances of classifier C
o`.oclIsInState(`s`)`	is o currently in state machine state s?
v`.oclIsUndefined()`	is value v null or invalid?
v`.oclIsInvalid()`	is value v invalid?

Evaluation

- Strict evaluation with some exceptions
  - `(if (1/0 = 0) then 0.0 else 0.0 endif).oclIsInvalid() = true`
  - `(1/0).oclIsInvalid() = true`
- Short-cut evaluation for **and**, **or**, **implies**
  - `(1/0 = 0.0) and false = false`
  - `true or (1/0 = 0.0) = true`
  - `false implies (1/0 = 0.0) = true`
  - `(1/0 = 0.0) implies true = true`
  - `if (0 = 0) then 0.0 else 1/0 endif = 0.0`

- In general, OCL expressions are evaluated over a system state.

e.g., represented
by an object diagram