Ludwig-
Maximilians-
Universität
München

LMU

Unified Modeling Language
UML 2.0

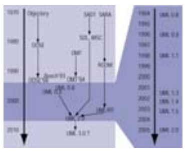
Prof. Dr. Harald Störrle

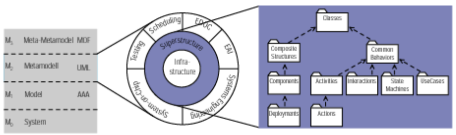
Unified Modeling Language 2.0

Part 1 – Introduction

Prof. Dr. Harald Störrle
University of Innsbruck
mgm technology partners

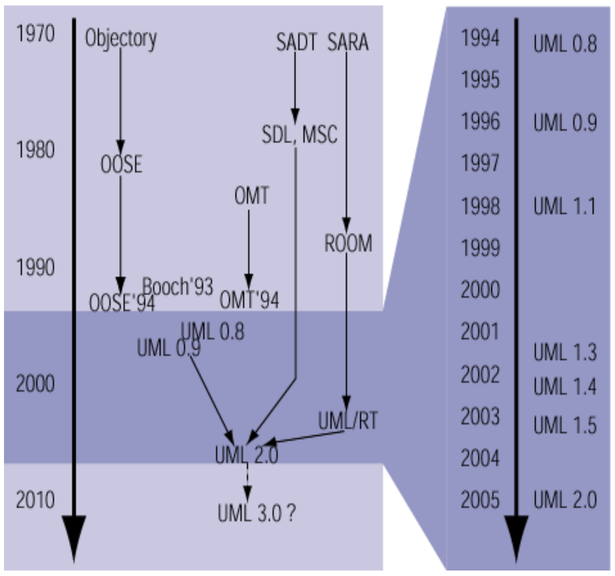
Dr. Alexander Knapp
University of Munich





1 - Introduction History and Predecessors

- **The UML is the "lingua franca" of software engineering.**
- **It subsumes, integrates and consolidates most predecessors.**
- **Through the network effect, UML has a much broader spread and much better support (tools, books, trainings etc.) than other notations.**
- **The transition from UML 1.x to UML 2.0 has**
  - **resolved a great number of issues;**
  - **introduced many new concepts and notations (often feebly defined);**
  - **overhauled and improved the internal structure completely.**
- **While UML 2.0 still has many problems, it is much better than what we ever had before.**



*current version ("the standard")
formal/05-07-04 of August '05*

1 - Introduction Usage Scenarios

- **UML has not been designed for specific, limited usages.**
- **There is currently no consensus on the role of the UML:**
  - **Some see UML only as tool for sketching class diagrams representing Java programs.**
  - **Some believe that UML is "*the prototype of the next generation of programming languages*".**
- **UML is a really a system of languages ("notations", "diagram types") each of which may be used in a number of different situations.**
- **UML is applicable for a multitude of purposes, during all phases of the software lifecycle, and for all sizes of systems - to varying degrees.**

1 - Introduction
Diagram types in UML 2

UML is a coherent system of languages rather than a single language.

Each language has its particular focus.

Structure		Class Diagram		static structure (generic/snapshot)
		Composite Structure Diagram		logical system structure
		Component Diagram		physical system structure
		Deployment Diagram		computing infrastructure / deployment
		Package Diagram		containment hierarchy
Behavior		Use Case Diagram		abstract functionality
		Activity Diagram		controlflow and dataflow
	Interaction	Sequence Diagram	interactions by message exchange	message exchange over time
		Communication Diagram		structure of interacting elements
		Timing Diagram		coordinated state change over time
		Interaction Overview Diagram		flows of interactions
		State Machine Diagram		event-triggered state change

Notationsübersicht (UML)

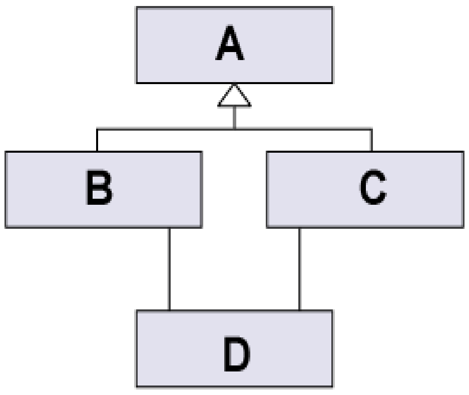
- **Die Unified Modeling Language (UML) ist der Industriestandard für Modellierungssprachen.**
- **Es ist hilfreich, diesen Standard zu kennen und einzuhalten:**
  - **es gibt zahlreiche Werkzeuge, Kurse, Bücher, Tutorials etc.**
  - **UML-Kenntnisse sind verfügbar bzw. auch in anderem Kontext nützlich.**
  - **UML-Modelle können zwischen Werkzeugen i.d.R. ausgetauscht werden*.**
- **Die UML ist als Allzwecksprache, also für alle Arten von Modellen konzipiert.**
- **Das bringt einiges an Komplexität**

Diagrammtyp

Struktur		Klassendiagramm
		Montagediagramm
		Komponentendiagramm
		Verteilungsdiagramm
		Paketdiagramm
Verhalten		Anwendungsfalldiagramm
		Aktivitätsdiagramm
	Interaktion	Sequenzdiagramm
		Kommunikationsdiagramm
		Zeitdiagramm
		Interaktionsübersichtsdiagramm
		Zustandsautomaten

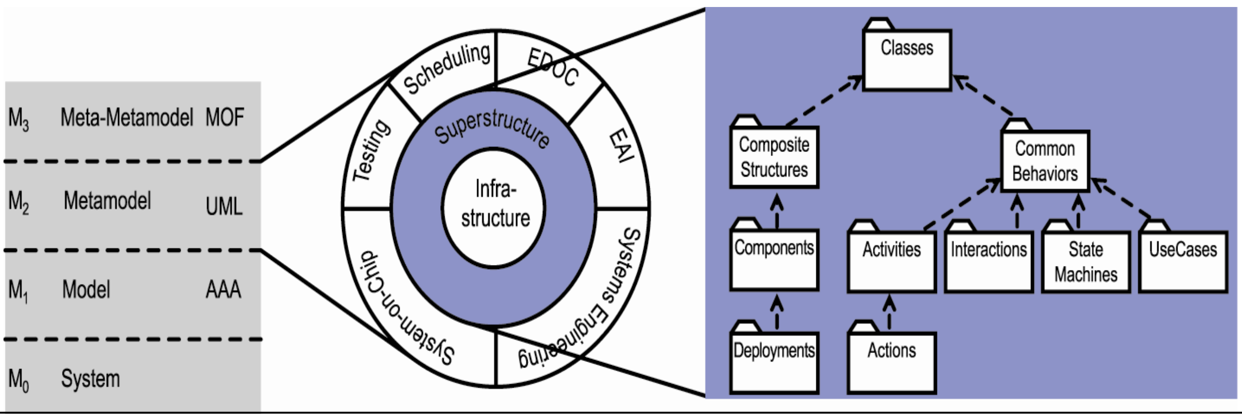
Diagram types also depend on their usage

- Each diagram type may be used in a multitude of settings, for each of which different rules and best practices may apply.
- For instance, class diagrams may be used during analysis as well as during implementation.
- During analysis, this class diagram is bad, or at least suspicious.
- During implementation, it is bad if and only if it does not correspond to the code (or other structure) it is used to represent.

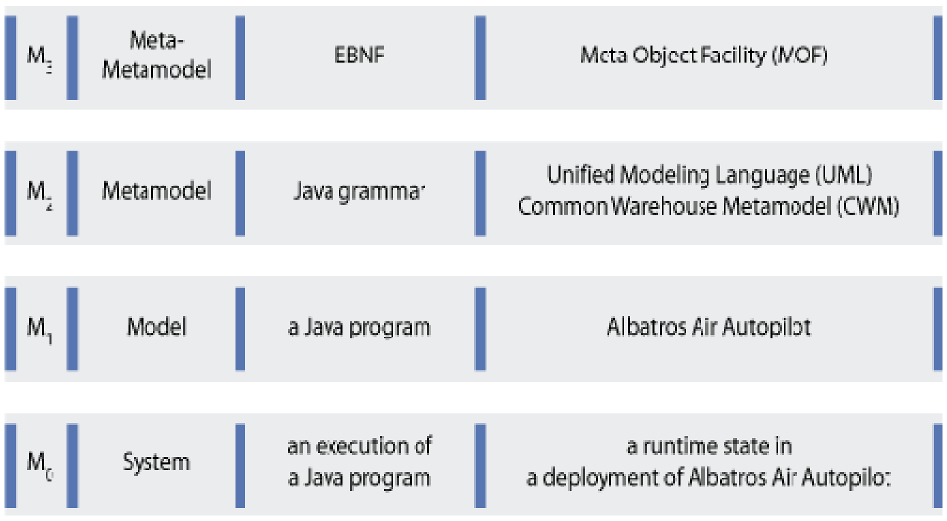


1 - Introduction
Internal Structure: Overview

- The UML is structured using a metamodeling approach with four layers.
- The M_2-layer is called metamodel.
- The metamodel is again structured into rings, one of which is called superstructure, this is the place where concepts are defined ("the metamodel" proper).
- The Superstructure is structured into a tree of packages in turn.

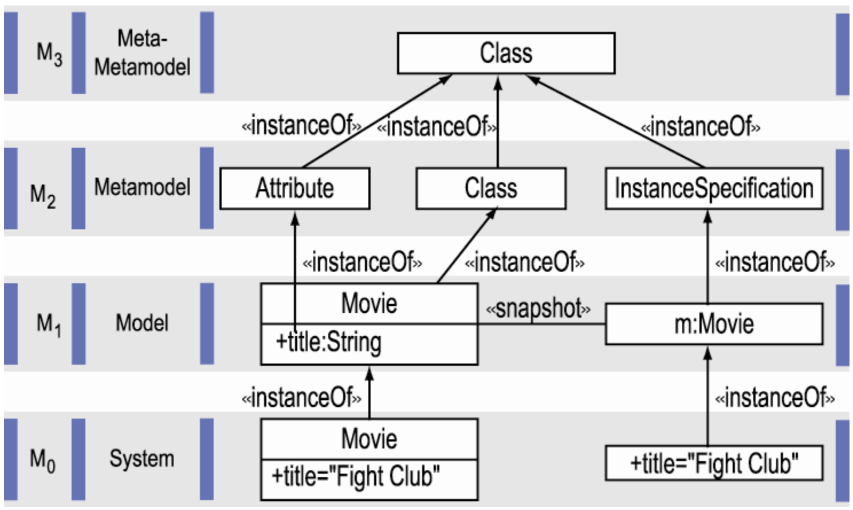


1 - Introduction
Internal Structure: Layers

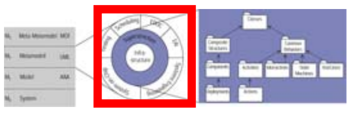


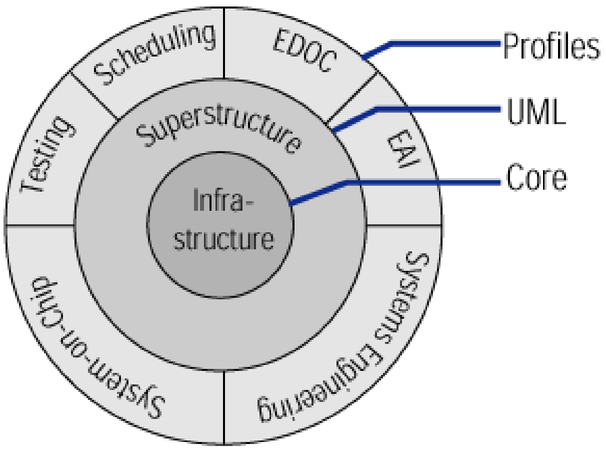
M_3	Meta-Metamodel	EBNF	Meta Object Facility (MOF)
M_2	Metamodel	Java grammar	Unified Modeling Language (UML) Common Warehouse Metamodel (CWM)
M_1	Model	a Java program	Albatros Air Autopilot
M_0	System	an execution of a Java program	a runtime state in a deployment of Albatros Air Autopilot

1 - Introduction
Internal Structure: Layers

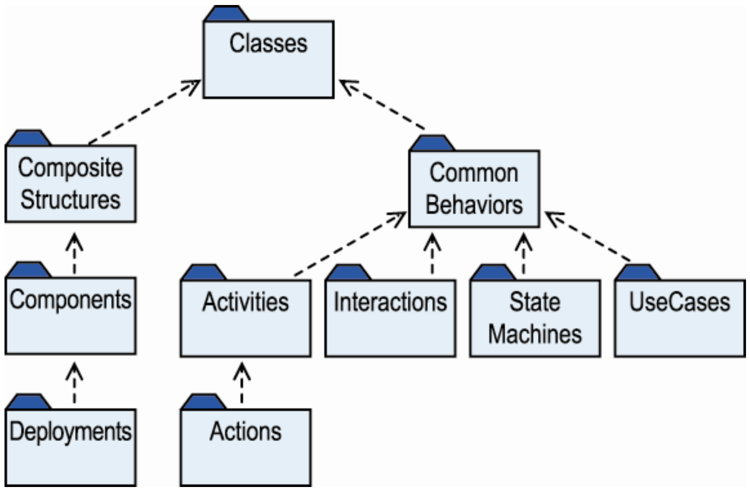


1 – Introduction
Internal Structure: Rings





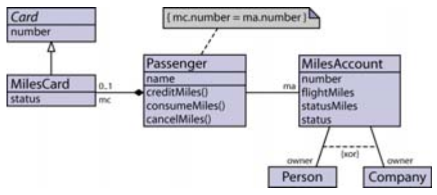
1 - Introduction
Internal Structure: Packages

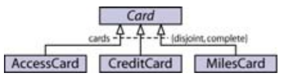


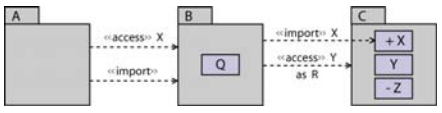
Unified Modeling Language 2.0
Part 2 – Classes and packages

Prof. Dr. Harald Störrle
University of Innsbruck
MGM technology partners

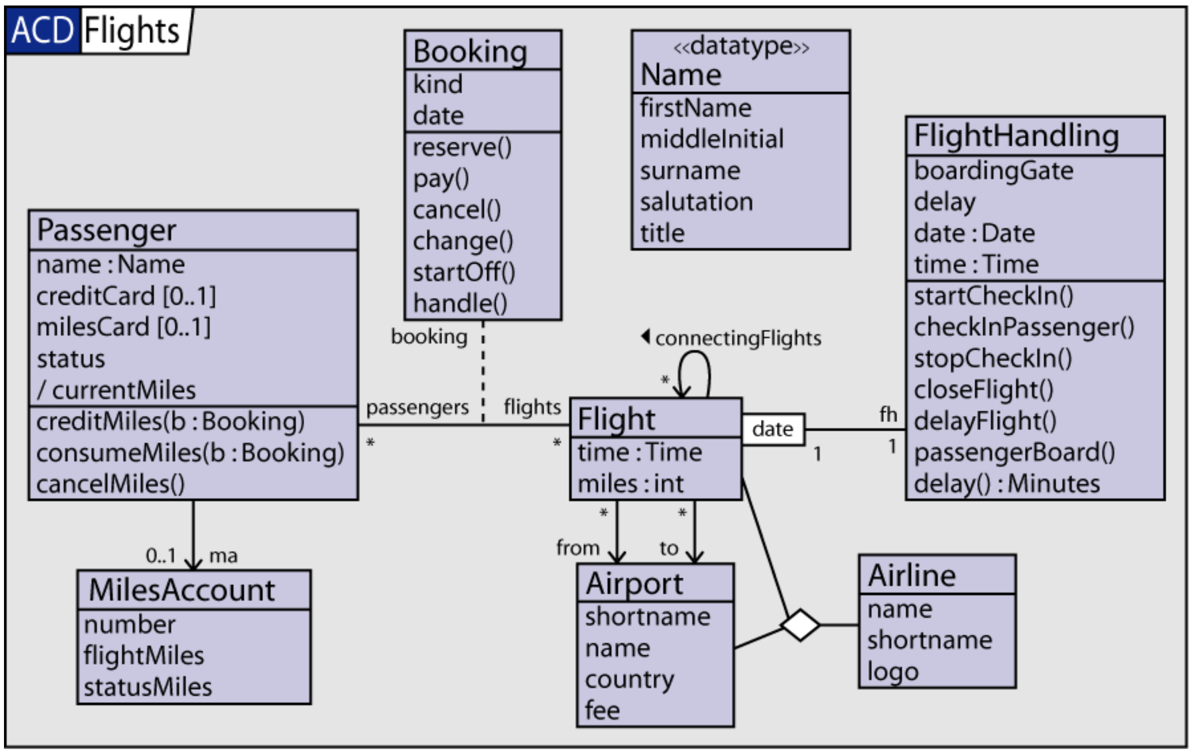
Dr. Alexander Knapp
University of Munich







2 – Classes and packages
A first glimpse



2 – Classes and packages
History and predecessors

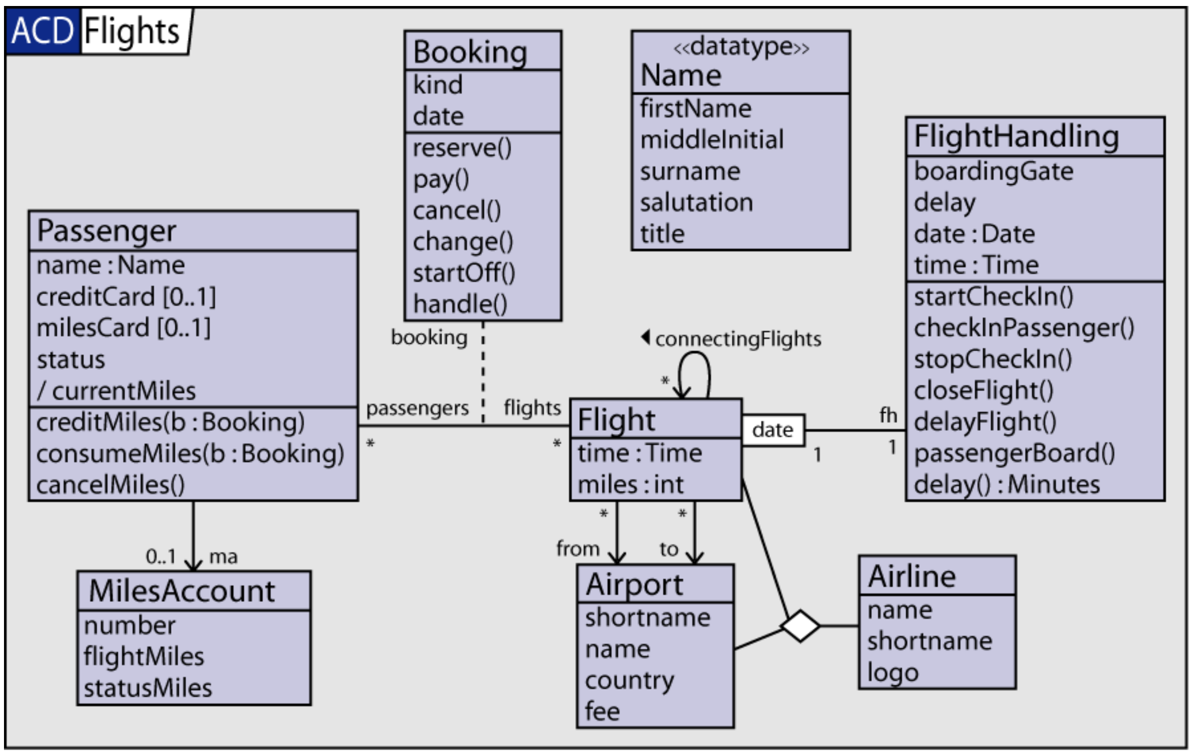
- **Structured analysis and design**
  - **Entity-Relationship (ER) diagrams (Chen 1976)**
- **Semantic nets**
  - **Conceptual structures in AI (Sowa 1984)**
- **Object-oriented analysis and design**
  - **Shlaer/Mellor (1988)**
  - **Coad/Yourdon (1990)**
  - **Wirfs-Brock/Wilkerson/Wiener (1990)**
  - **OMT (Rumbaugh 1991)**
  - **Booch (1991)**
  - **OOSE (Jacobson 1992)**

2 – Classes and packages
Usage scenarios

- **Classes and their relationships describe the vocabulary of a system.**
  - Analysis: **Ontology, taxonomy, data dictionary, …**
  - Design: **Static structure, patterns, …**
  - Implementation: **Code containers, database tables, …**
- **Classes may be used with different meaning in different software development phases.**
  - **meaning of generalizations varies with meaning of classes**

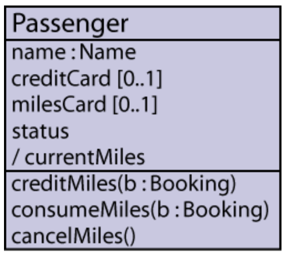
	Analysis	Design	Implementation
Concept	√		´
Type		√	√
Set of objects	´	√	√
Code	´		√

2 – Classes and packages
Analysis class diagram (1)



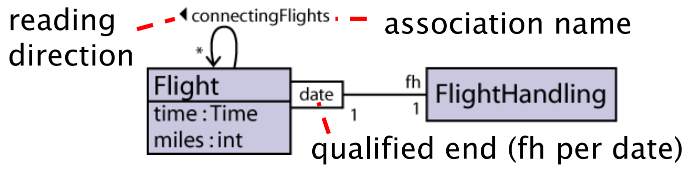
2 – Classes and packages
Classes

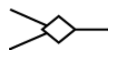
- **Classes describe a set of instances with common features (and semantics).**
  - **Classes induce types (representing a set of values).**
  - **Classes are namespaces (containing named elements).**
- **Structural features (are typed elements)**
  - **properties**
    - commonly known as attributes
    - describe the structure or state of class instances
    - may have multiplicities (e.g. **1, 0..1, 0..*, *, 2..5**)
      (default: 0..* = *, but 1 for association ends)
- **Behavioral features (have formal parameters)**



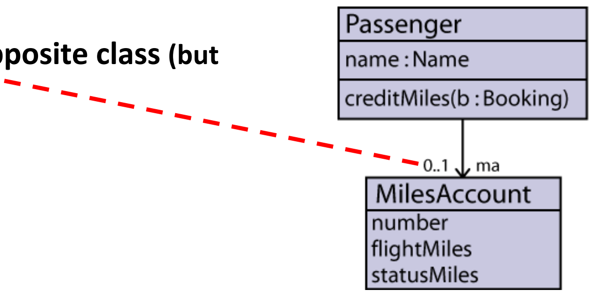
2 – Classes and packages
Associations

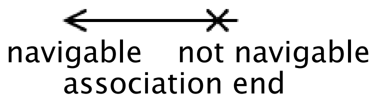
- **Associations describe sets of tuples whose values refer to typed instances.**
  - **In particular, structural relationship between classes**
  - **Instances of associations are called links.**




ternary association

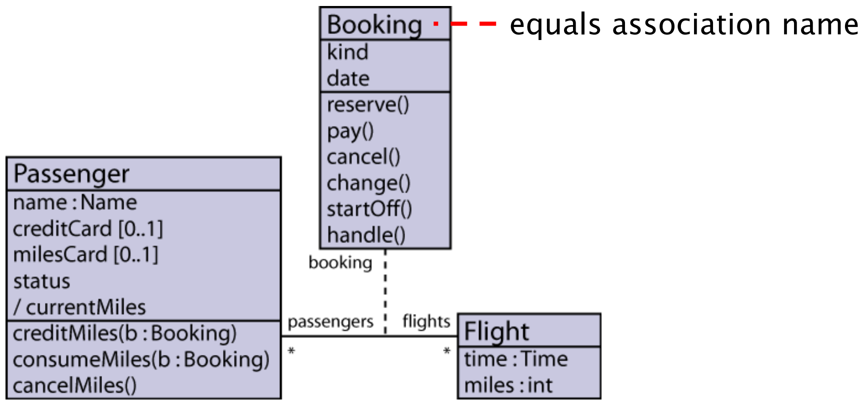
- **Association ends are properties.**
  - **correspond to properties of the opposite class (but default multiplicity is 1)**
- **Association ends may be navigable.**
  - **in contrast to general properties**





2 – Classes and packages
Association classes

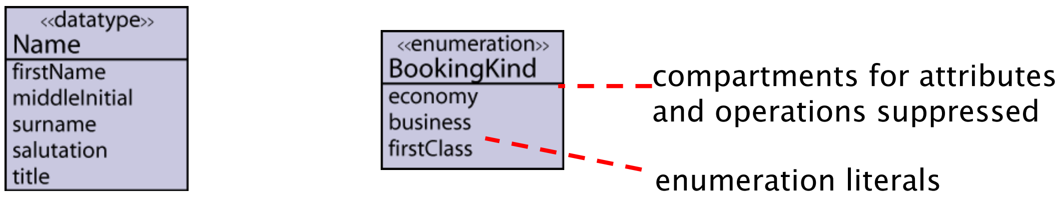
- **Association classes combine classes with associations.**
  - **not only connect a set of classifiers but also define a set of features that belong to the relationship itself and not to any of the classifiers**



- each instance of Booking has one passenger and one flight
- each link of Booking is one instance of Booking

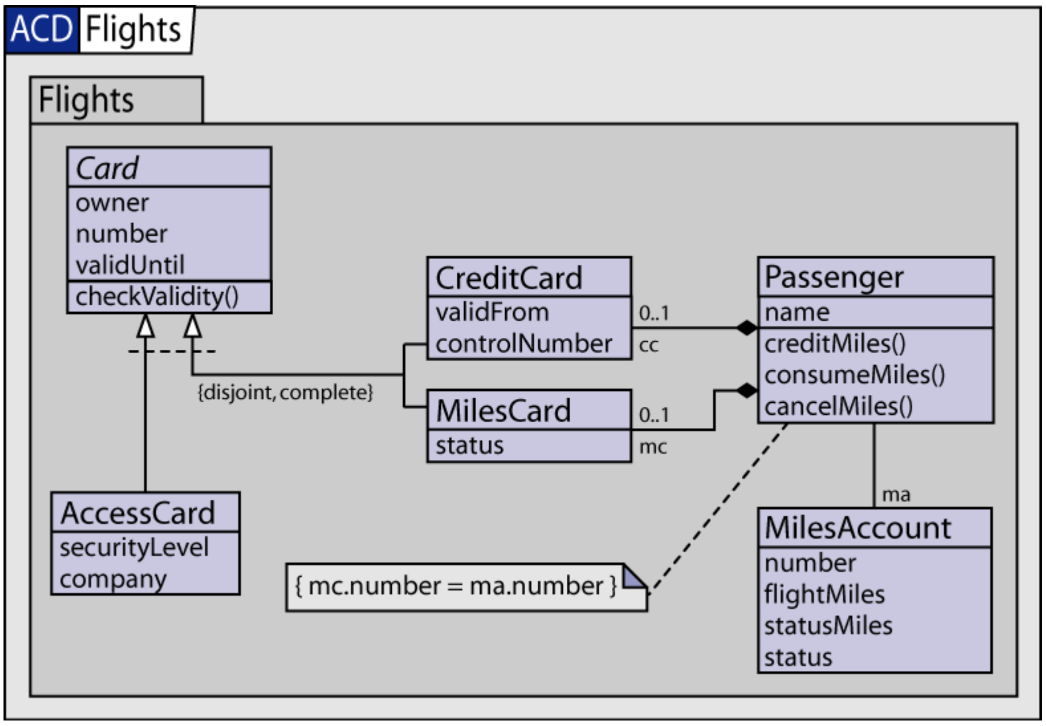
2 – Classes and packages
Data types and enumerations

- **Data types are types whose instances are identified by their value.**
  - **Instances of classes have an identity.**
  - **may show structural and behavioral features**



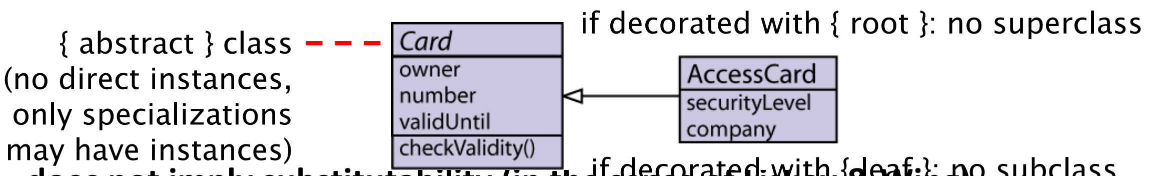
- **Enumerations are special data types.**
  - **instances defined by enumeration literals**
    - denoted by *Enumeration*::*EnumerationLiteral* or *#EnumerationLiteral*
  - **may show structural and behavioral features**

2 – Classes and packages
Analysis class diagram (2)

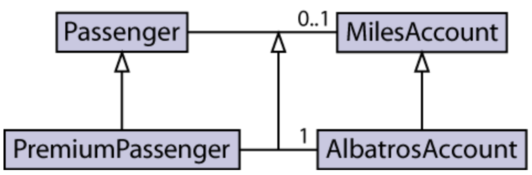


2 – Classes and packages
Inheritance (1)

- **Generalizations relate specific classes to more general classes.**
  - **instances of specific class also instances of the general class**
  - **features of general class also implicitly specified for specific class**
  - **does not imply substitutability (in the sense of Liskov & Wing)**
    - must be specified on specific class separately by { substitutable }

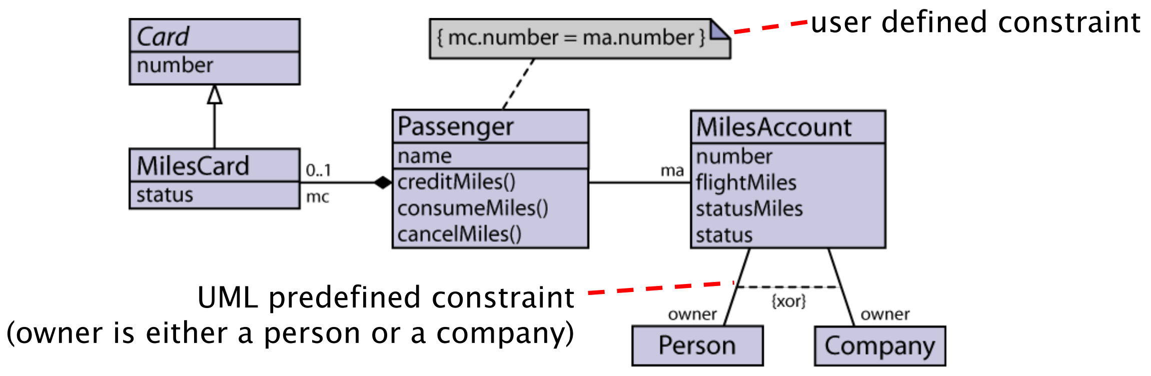


- **Generalizations also apply to associations.**
  - **as both are Classifiers**



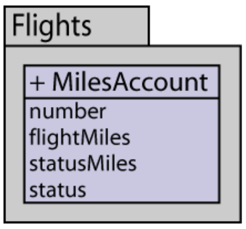
2 – Classes and packages
Constraints

- **Constraints restrict the semantics of model elements.**
  - **constraints may apply to one or more elements**
  - **no prescribed language**
    - OCL is used in the UML 2.0 specification
    - also natural language may be used

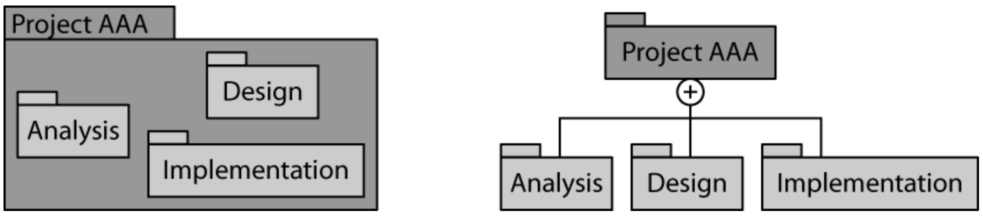


2 – Classes and packages
Packages (1)

- **Packages group elements.**
  - **Packages provide a namespace for its grouped elements.**
  - **Elements in a package may be**
    - public (+, visible from outside; default)
    - private (-, not visible from outside)
  - **Access to public elements by qualified names**
    - e.g., Flights::MilesAccount

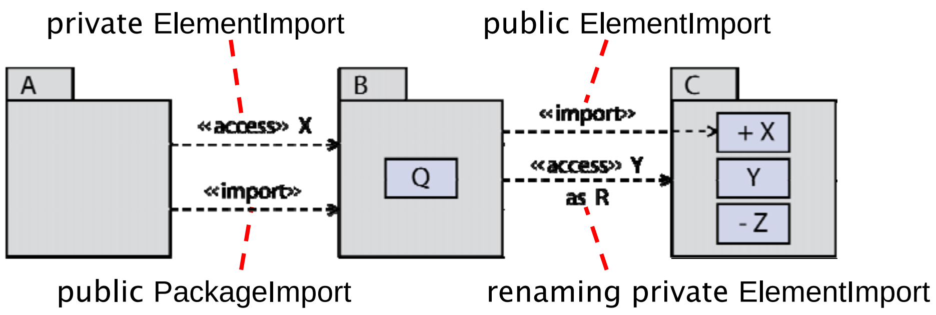


Notational variants



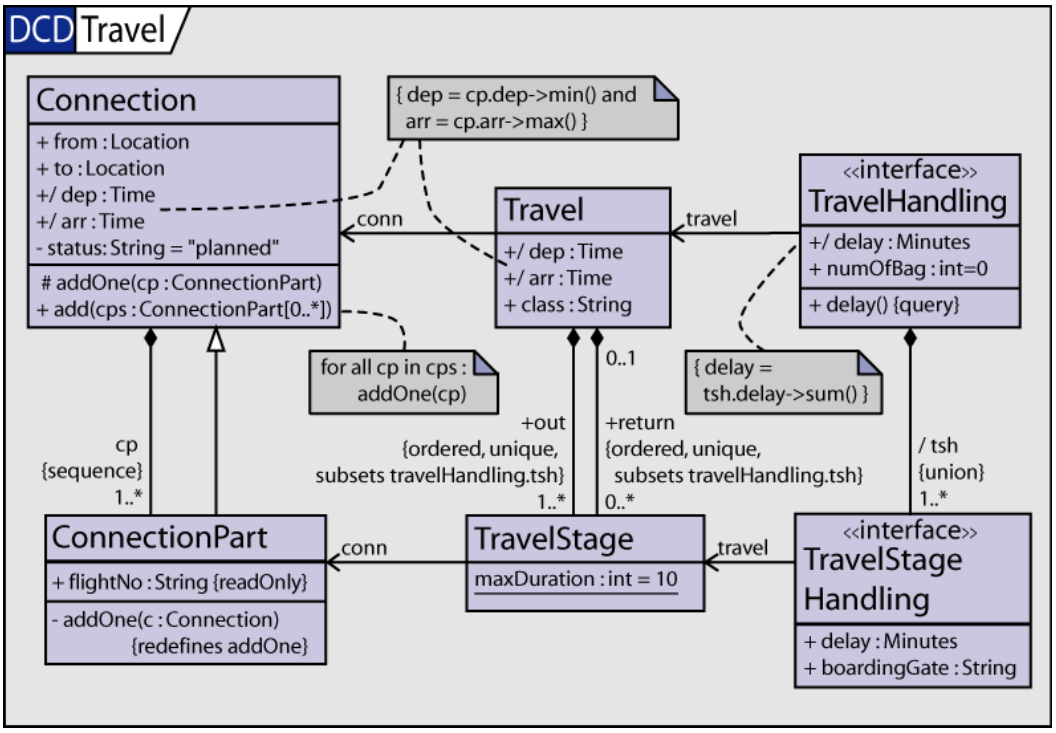
2 – Classes and packages
Packages (2)

- **Package imports simplify qualified names.**



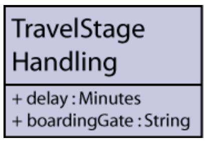
Package	Element	Visibility	
A	X	private	separate private element import (otherwise public overrides private)
A	Q	public	all remaining visible elements of B
B	X	public	public import
B	Q	public	default visibility
B	R	private	private import, renaming

2 – Classes and packages
Design class diagram


DCD Travel
Connection
+ from : Location
+ to : Location
+/ dep : Time
+/ arr : Time
- status: String = "planned"
# addOne(cp : ConnectionPart)
+ add(cps : ConnectionPart[0..*])
{ dep = cp.dep->min() and arr = cp.arr->max() }
Travel
+/ dep : Time
+/ arr : Time
+ class : String
«interface» TravelHandling
+/ delay : Minutes
+ numOfBag : int=0
+ delay() {query}
conn
travel
for all cp in cps : addOne(cp)
{ delay = tsh.delay->sum() }
0..1
cp {sequence} 1..*
+out {ordered, unique, subsets travelHandling.tsh} 1..*
+return {ordered, unique, subsets travelHandling.tsh} 0..*
/ tsh {union} 1..*
ConnectionPart
+ flightNo : String {readOnly}
- addOne(c : Connection) {redefines addOne}
conn
TravelStage
maxDuration : int = 10
travel
«interface» TravelStage Handling
+ delay : Minutes
+ boardingGate : String

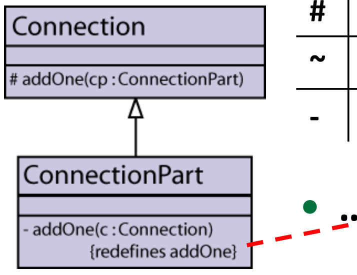
2 – Classes and packages
Features

- **… belong to a namespace (e.g., class or package)**

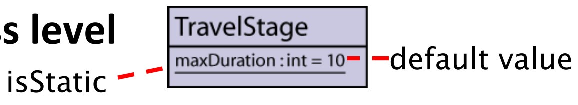


Visibility kinds (no default)

		visible to elements …
+	public	that can access owning namespace (by membership, import, or access)
#	protected	with generalization to owning namespace
~	package	in the same package as the owning namespace
-	private	in owning namespace only


Connection
# addOne(cp : ConnectionPart)
ConnectionPart
- addOne(c : Connection) {redefines addOne}

- **… are redefinable (unless decorated by { leaf })**
  - **in classes that specialize the context class**
- **… can be defined on instance or class level**

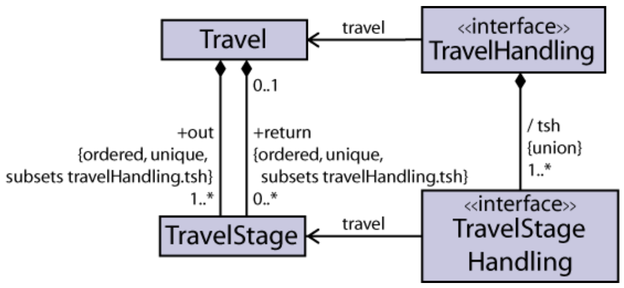


2 – Classes and packages
Properties

Aggregation kinds (default: *none*)

none		reference
shared	——	**undefined (!)**
composite	◇——	value
	◆——	

{ ordered }	{ unique }	**Collection type**
√	√	`OrderedSet`
√	´	`Sequence`
´	√	`Set` (default)
´	´	`Bag`


Travel
travel
«interface» TravelHandling
0..1
+out {ordered, unique, subsets travelHandling.tsh} 1..*
+return {ordered, unique, subsets travelHandling.tsh} 0..*
/ tsh {union} 1..*
TravelStage
travel
«interface» TravelStage Handling

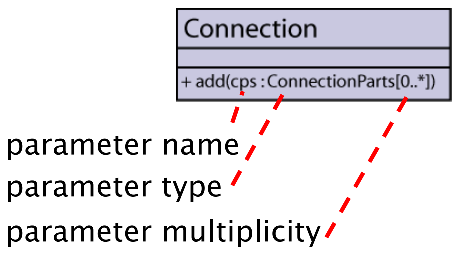
/ ({ derived })	can be computed from other information (default: false)
{ readOnly }	can only be read, not written (default: false = unrestricted)
{ union }	union of subset properties (implies derived)
{ subsets … }	which property this property is a subset of

2 – Classes and packages
Operations (1)

- **An operation specifies the name, return type, formal parameters, and constraints for invoking an associated behavior.**
  - **«pre» / «post»**
    - precondition constrains system state on operation invocation
    - postcondition constrains system state after operation is completed
  - **{ query }: invocation has no side effects**
    - «body»: body condition describes return values
  - **{ ordered, unique } as for properties, but for return values**
  - **exceptions that may be thrown can be declared**

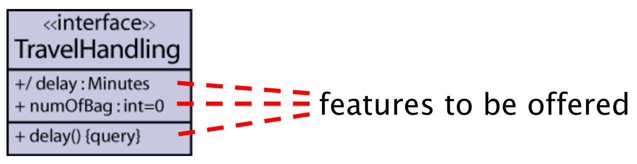
Parameter direction kinds (default: in)

in	*one way from caller*
out	*one way from callee*
inout	*both ways*
return	*return from callee* (at most 1)

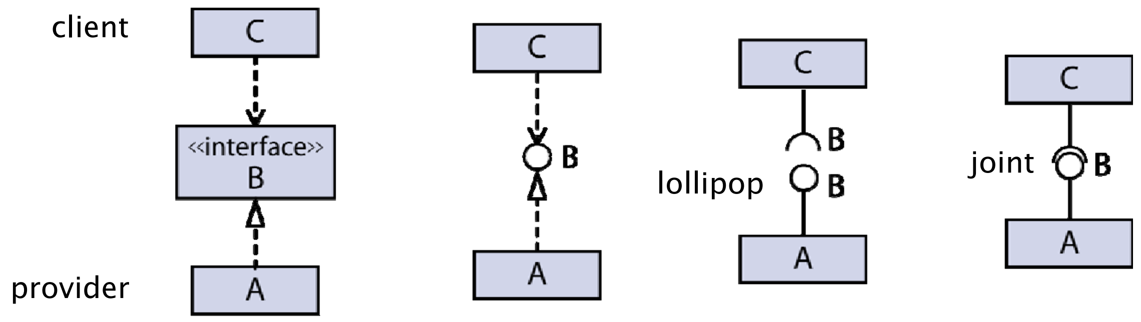


2 – Classes and packages
Interfaces

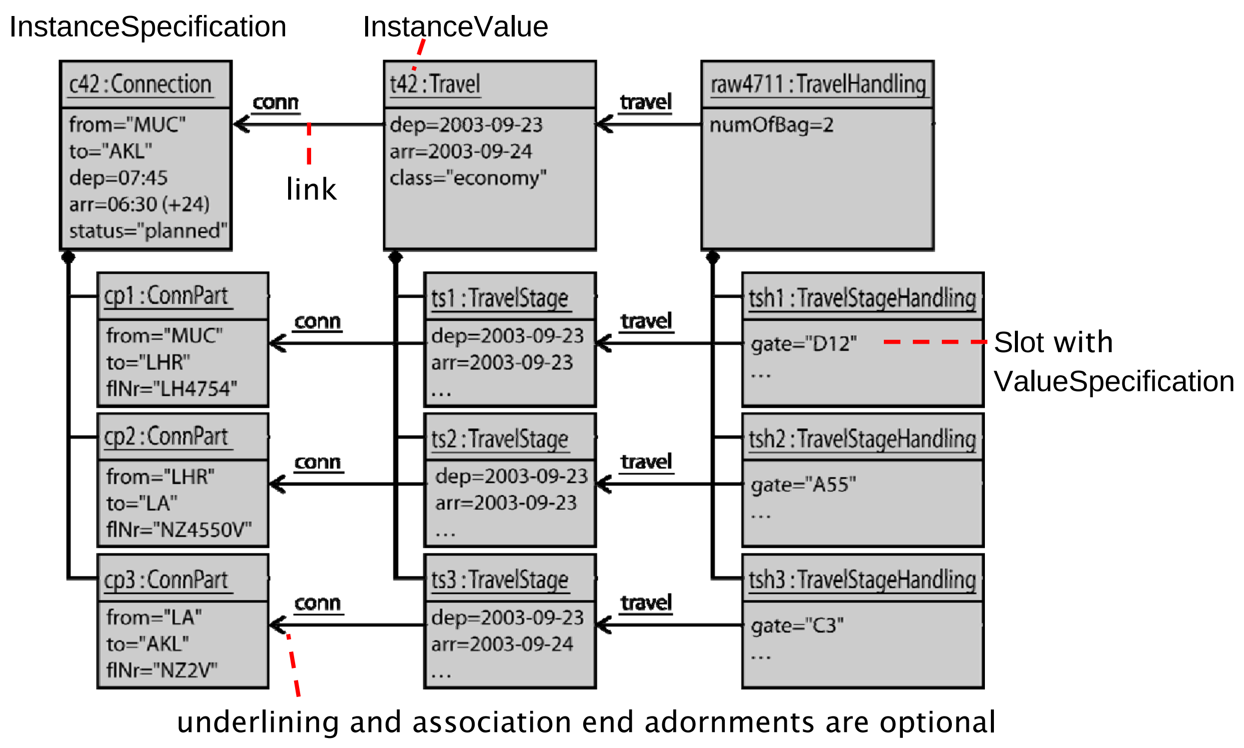
- **Interfaces declare a set of coherent public features and obligations.**
  - **i.e., specify a contract for implementers (realizers)**



Several notations for client/provider relationship

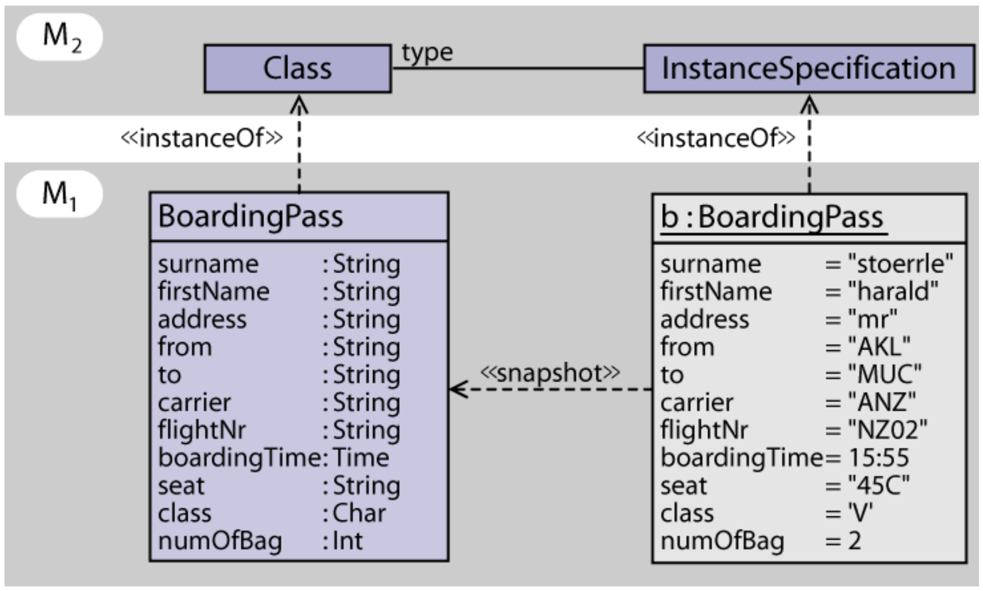


2 – Classes and packages
Object diagram



2 – Classes and packages
Instances specifications

UML metamodel



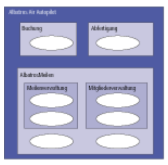
user model

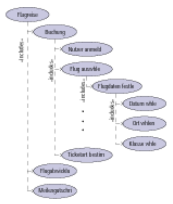
Unified Modeling Language 2.0

Part 3 – Use Cases

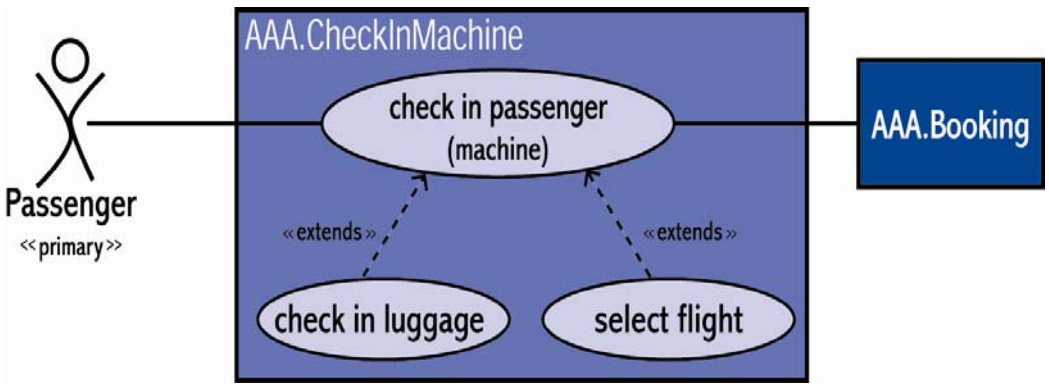
Prof. Dr. Harald Störrle
University of Innsbruck
MGM technology partners

Dr. Alexander Knapp
University of Munich





3 - Use Cases
A first glimpse



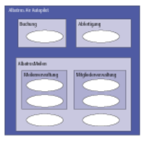
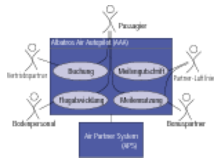
- **Displayed aspects**
    - **System boundary and context of system**
    - **Users and neighbor systems**
    - **Functionalities**
    - **Relationships between functionalities (calling/dependency, taxonomy)**
    - **Functional requirements**
    - **Some non-functional ("quality") requirements as comments/annotations**

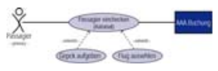
3 - Use Cases
History and predecessors

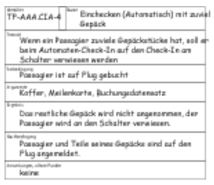
- **1970's**
    - **Structured methods (SADT etc.) use top-level DFD as context diagram**
    - **Structured methods use function trees**
- **1980's**
    - **Jacobson (Objectory) introduces the concept of use case as an aid for communicating with domain experts**
- **1997**
    - **UML 1.3 encompasses Use Cases**

3 - Use Cases
Usage scenarios

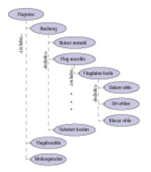
- **Use case inventory/ domain architecture**
  - **complete catalog of all subdomains and (groups of) business processes and business functions**
  - **overview of system's (domain) capabilities**
- **"Classical" use cases**
  - **illustrate context of individual functionality**
  - **useful in design/documentation of business processes (i.e. analysis phase and reengineering)**
- **Use Case / Test case table**
  - **schematic detail description of business process/function/test case**
- **Function tree**
  - **describe functional decomposition of system behavior**
  - **useful for non-OO construction and for re-architecting pre-OO systems**





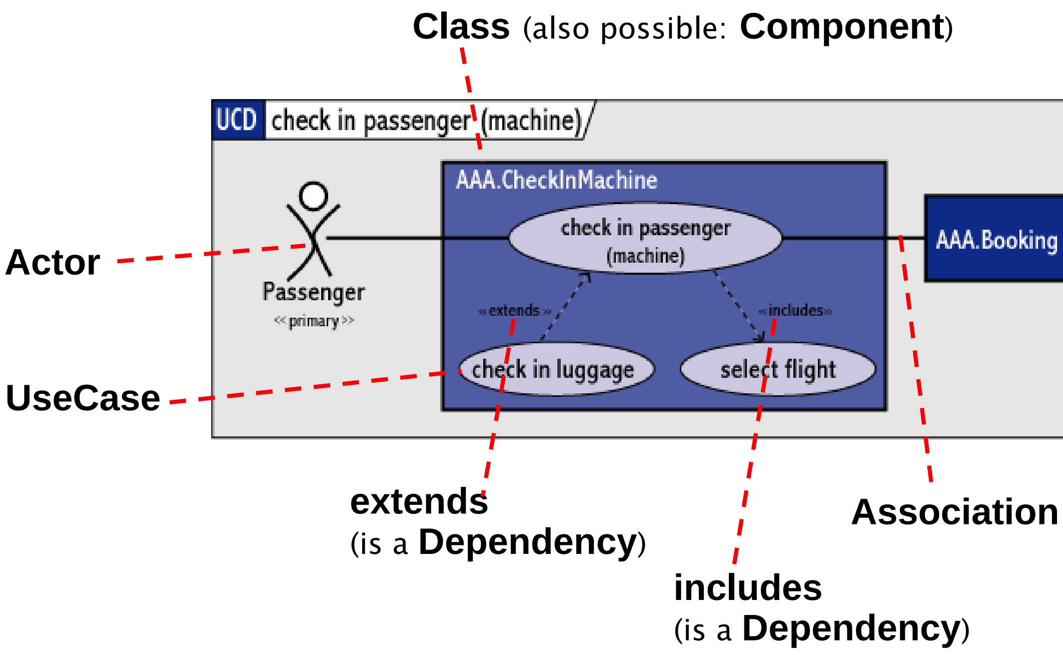
TP-AAA.CIA-4	Einchecken (Automatisch) mit zuviel Gepäck
Wenn ein Passagier zuviele Gepäckstücke hat, soll er beim Automaten-Check-In auf den Check-In am Schalter verwiesen werden	
Passagier ist auf Flug gebucht	
Koffer, Meilenkarte, Buchungsdatensatz	
Das restliche Gepäck wird nicht angenommen, der Passagier wird an den Schalter verwiesen.	
Passagier und Teile seines Gepäcks sind auf den Flug angemeldet.	
keine	



3 – Use Cases
Types of use cases

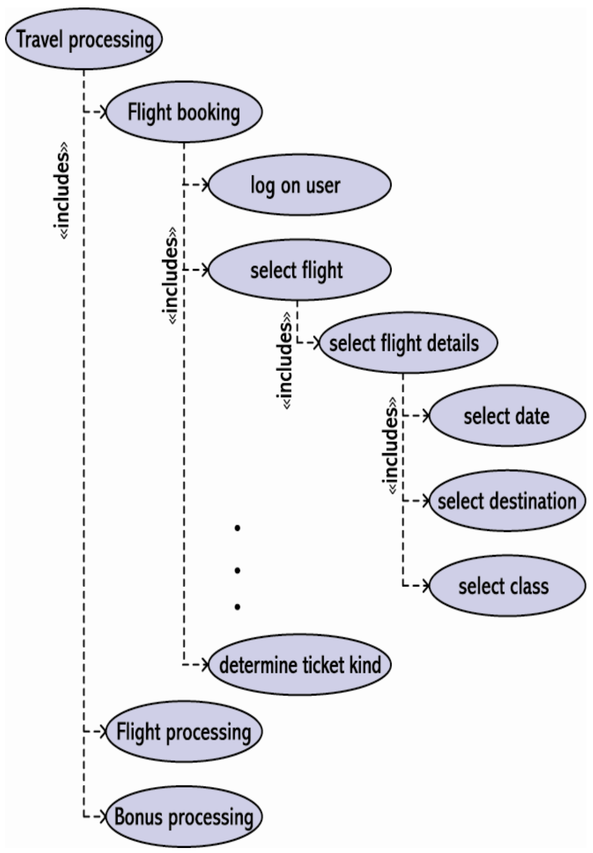
- **The UML provides only the concept of use case. In many applications, however, there are two fundamentally different kinds of use cases:**
  - **business processes ("processes")**
    - white box, large scale, long running (suspendable), customized processes
    - either dialogue or batch processes
    - directly support the business or domain of the system, create or destroy value
    - are subject to rearrangement when business changes
    - may contain some manual steps and business functions
  - **business functions ("services")**
    - black box, small(er) scale, short(er) running, atomic, reusable function
    - small recurring functionality, plausibility, user dialogue, interface call, . .

3 - Use Cases
Main concepts (concrete syntax)



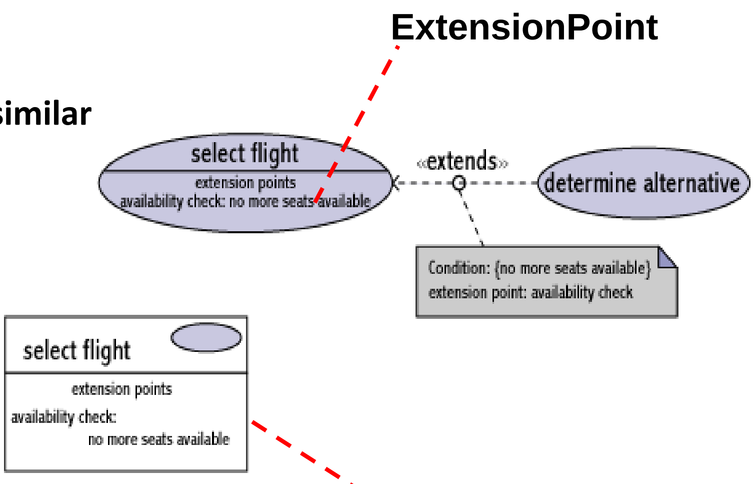
3 - Use Cases
Inclusion & extension

- **Inclusion**
  - **plain old call**
  - **directed from caller to callee**
  - **may occur once or many times**
- **Extension**
  - **covers variant or exceptional behavior**
  - **relationship is directed from exception to standard case**
  - **may or may not occur**
  - **occurs at most once**



3 - Use Cases
Extension points

- **An extension occurs at a (named) ExtensionPoint, when a specific condition is satisfied.**
- **In a way, ExtensionPoints are similar to *user exits* or *hooks*.**



UseCase with ExtensionPoint, alternative syntax suitable for large numbers of ExtensionPoints

- **In real world systems, there are *many* ExtensionPoints, most of which are poorly documented.**

UseCase-Template

ID	
Name	
System/Subsystem	
Akteure	
Beschreibung	
Vorbedingungen	
Nachbedingungen	
Parameter	
Ergebnisse	
Auslöser	
Ablauf	
Varianten	
NFA	
Offene Fragen	
Anmerkungen	