Chapter 11

Software Design

Software Development Stages

- **Analysis**
  - Definition of system objectives.
  - Definition of system requirements.
- **Design**
  - Decomposition to modules and classes.
  - Specification of module and class contents.
  - Specification of data structures and algorithms.
- **Implementation**
  - Coding of software.
- **Testing**
  - Validation.
  - Verification.
- **Maintenance**
  - Bug fixing.
  - Extensions.
  - Guidance and consulting users.

The Object Oriented Paradigm

- About 70% of software projects fail
- In many cases the tool works but is useless
  - Does not meet customers' needs
  - Too late for the market
- The OO paradigm is addressed at overcoming software complexity
  - Make the tool easy to comprehend (user & developer)
  - Adhere to the *open-close* principle at each and every level (from class-level and up)
  - The above two result in significantly ease modifications and extensions
- Object Orientation is about *process*
  - It is not about *design* or about *programming*
  - It is not about *classes* or about *packages*
  - It is not about *UML* or about *C++*

Requirement Analysis

- Detailed interviews and dialogues with customers
  - Customers may have conflicting and/or contradictory expectations
- Textual descriptions of all various usage modes are created
  - These describe in details communication among participants
  - These are called *scripts*
  - Participants are called *actors*
- These descriptions are called

  Use Cases

Unified Modeling Language

- Based on previously used practices
- A Collection of various modeling techniques
- Allow modeling of different aspects
- They were modified to have a common style

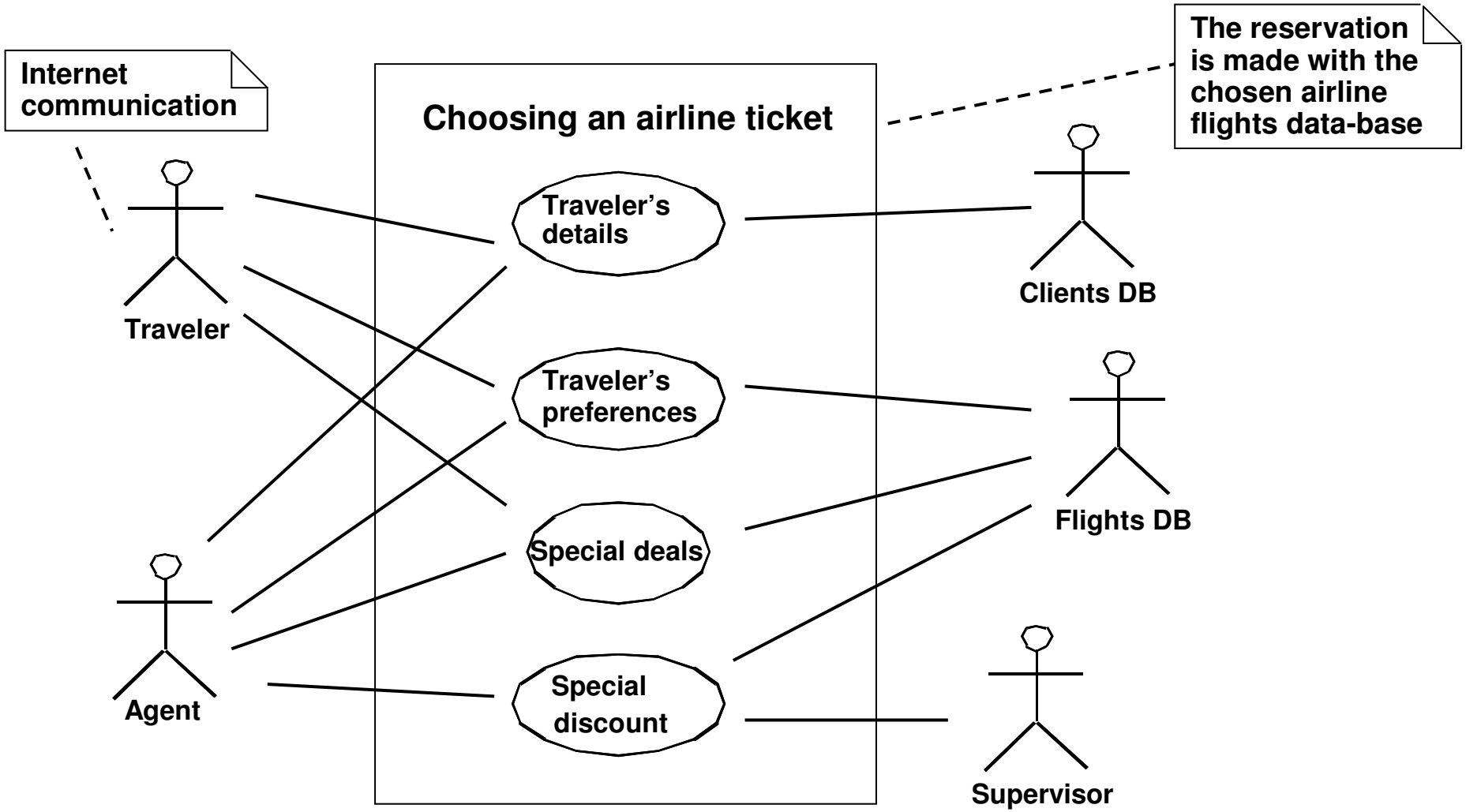
Use Cases

- A use case is the basis for
  - Use case diagrams
  - Collaboration diagrams
  - Sequence diagrams
- Eventually, use cases are the bases for the software
  - Nouns become classes
  - Verbs become messages
  - Relationships become relations
- Relationships between use cases
  - << include >> *(common to several)*
  - << extend >> *(a possible extension)*
  - generalization *(of a more specific)*

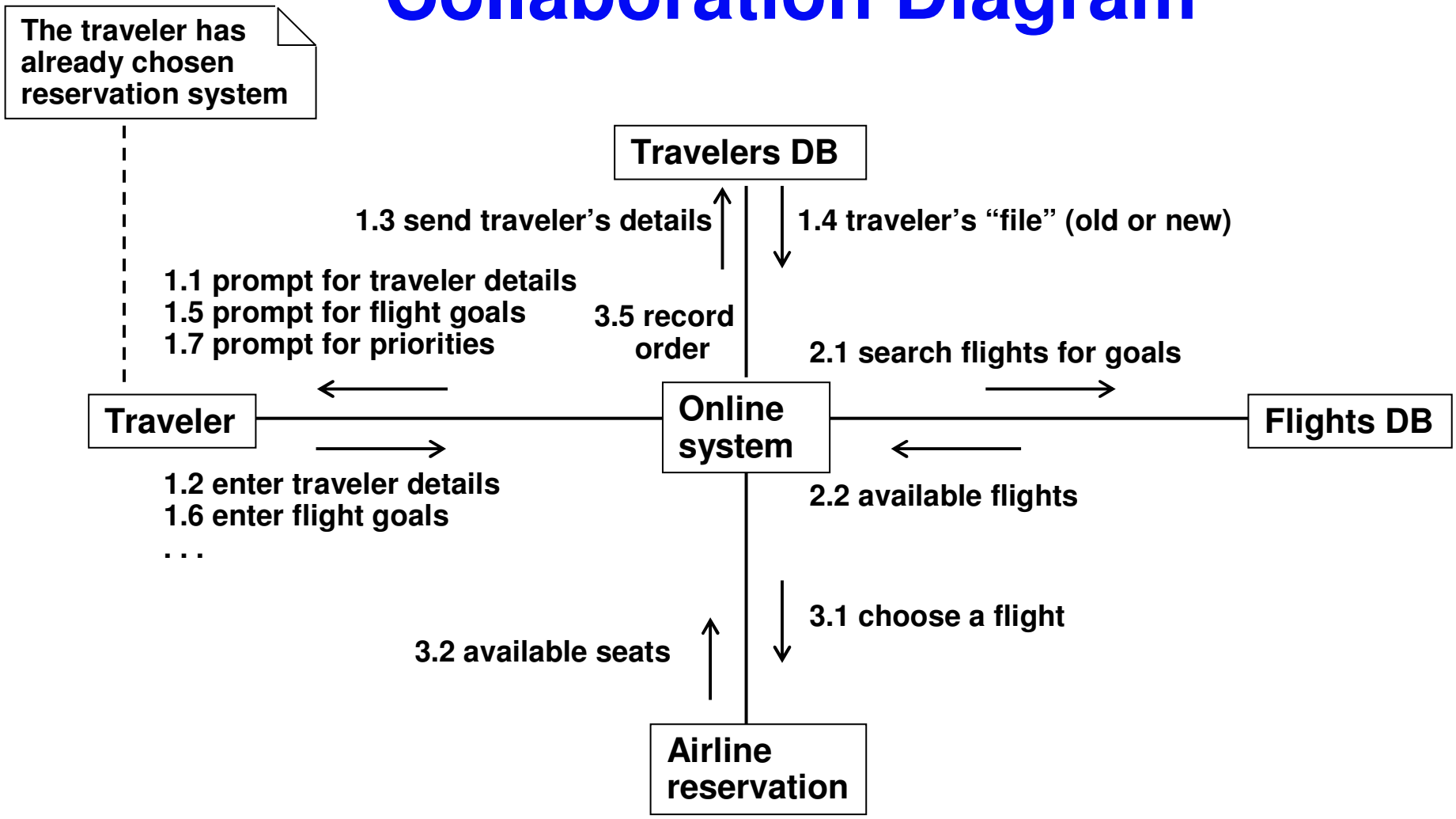
A Use Case Description:
a traveler orders an airline ticket

- **Actors:** traveler, travel agent, supervisor, flights DB (customer DB)
- **Preconditions:** cummunication channels established
- **Primary course**
  - Traveler presents personal details *(optional with an agent)*
  - Traveler presents goals: date, destination, price, airline
  - Traveler prioritize goals and presents constraints
  - Agent checks for special deals: student, frequent-flier . . .
  - Agent presents 3 best matches, then 10 best matches
  - Traveler negotiates: price-difference, departure time, . . .
  - . . . . .
- **Secondary courses**
  - Agent asks supervisor for a special discount
  - Usecase needs no agent if traveler connects to DB via Internet
    - » No special discount is available in this course
    - » Traveler sees less DB-information on screen

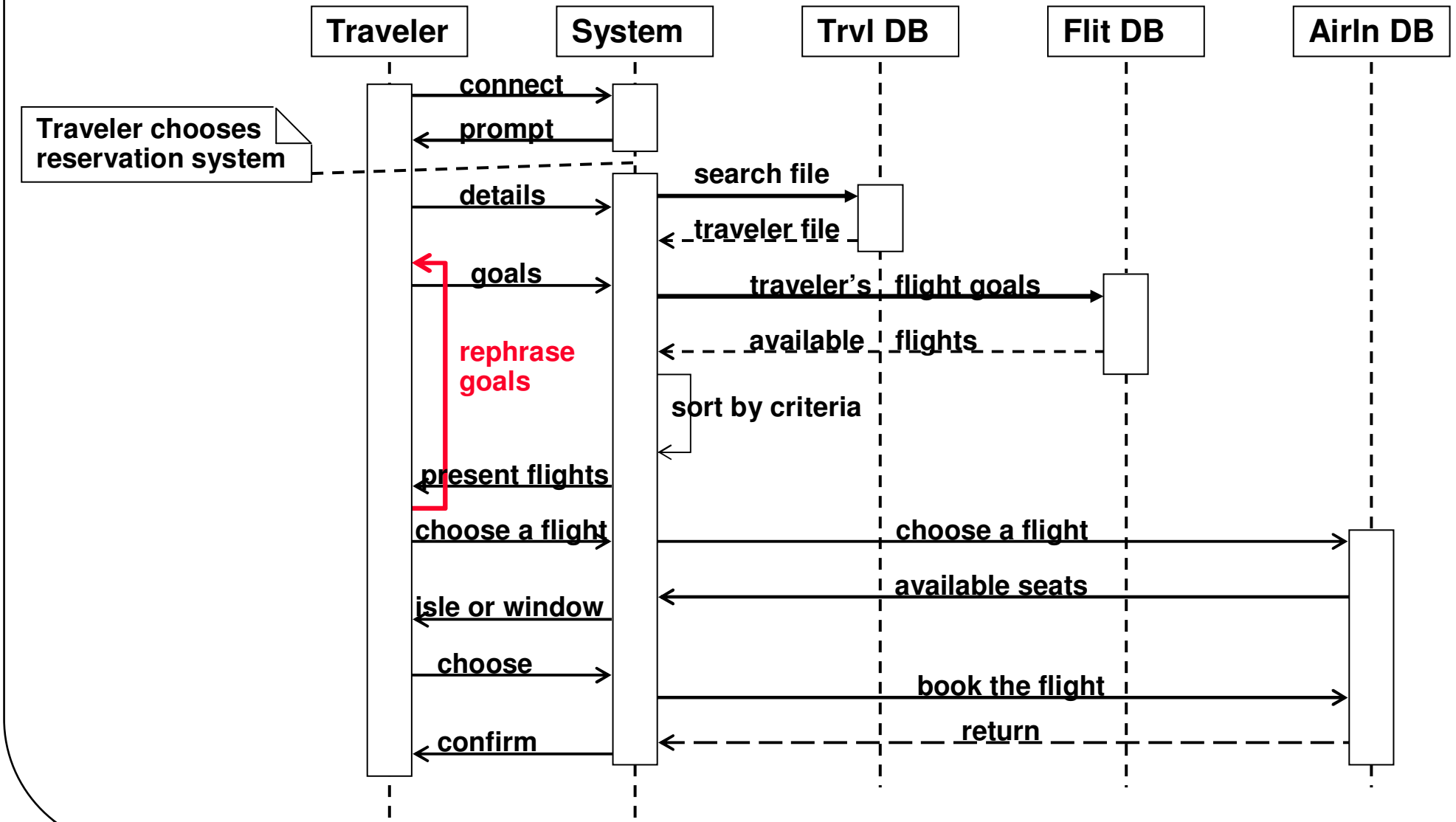
Use Case Diagram



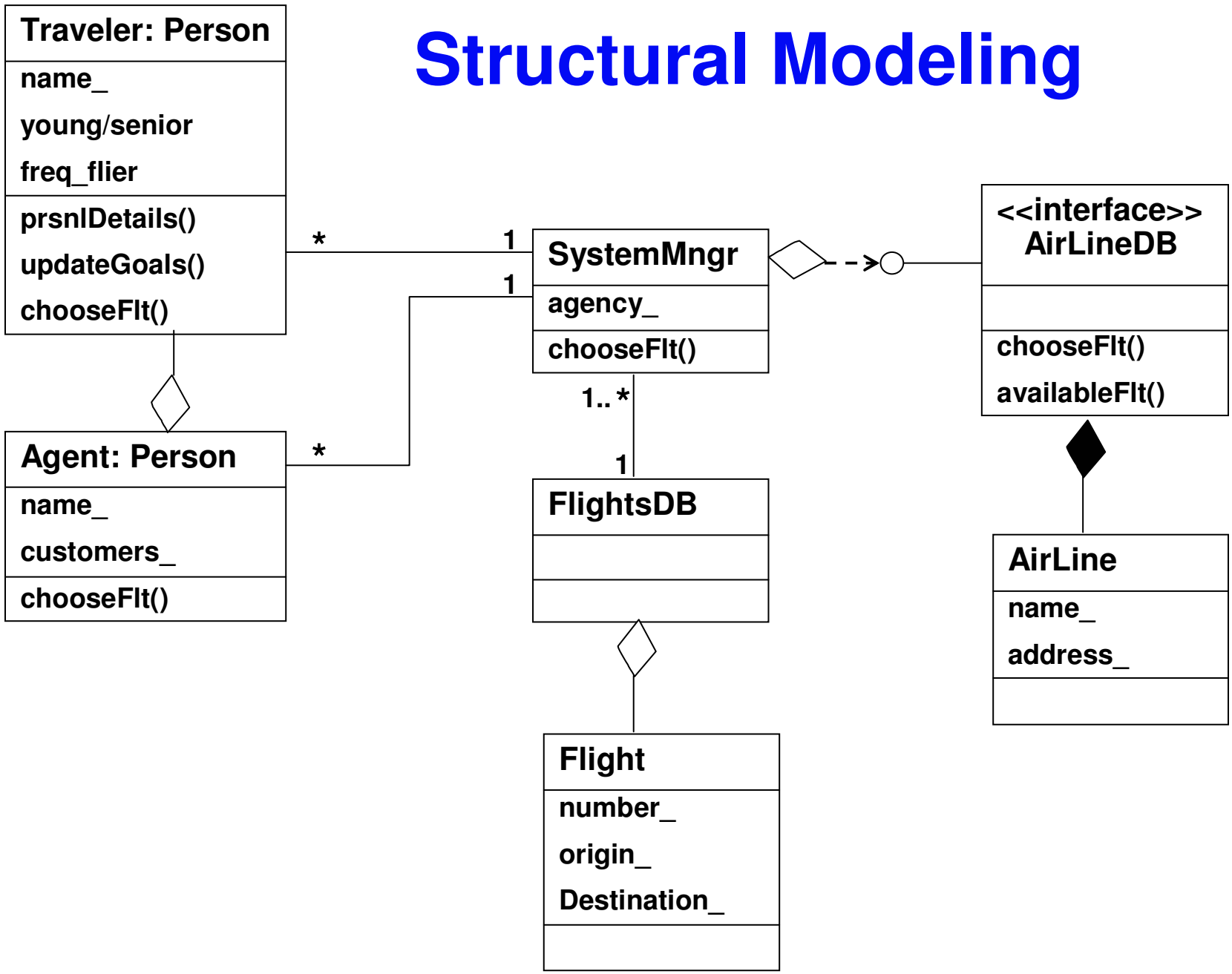
Collaboration Diagram



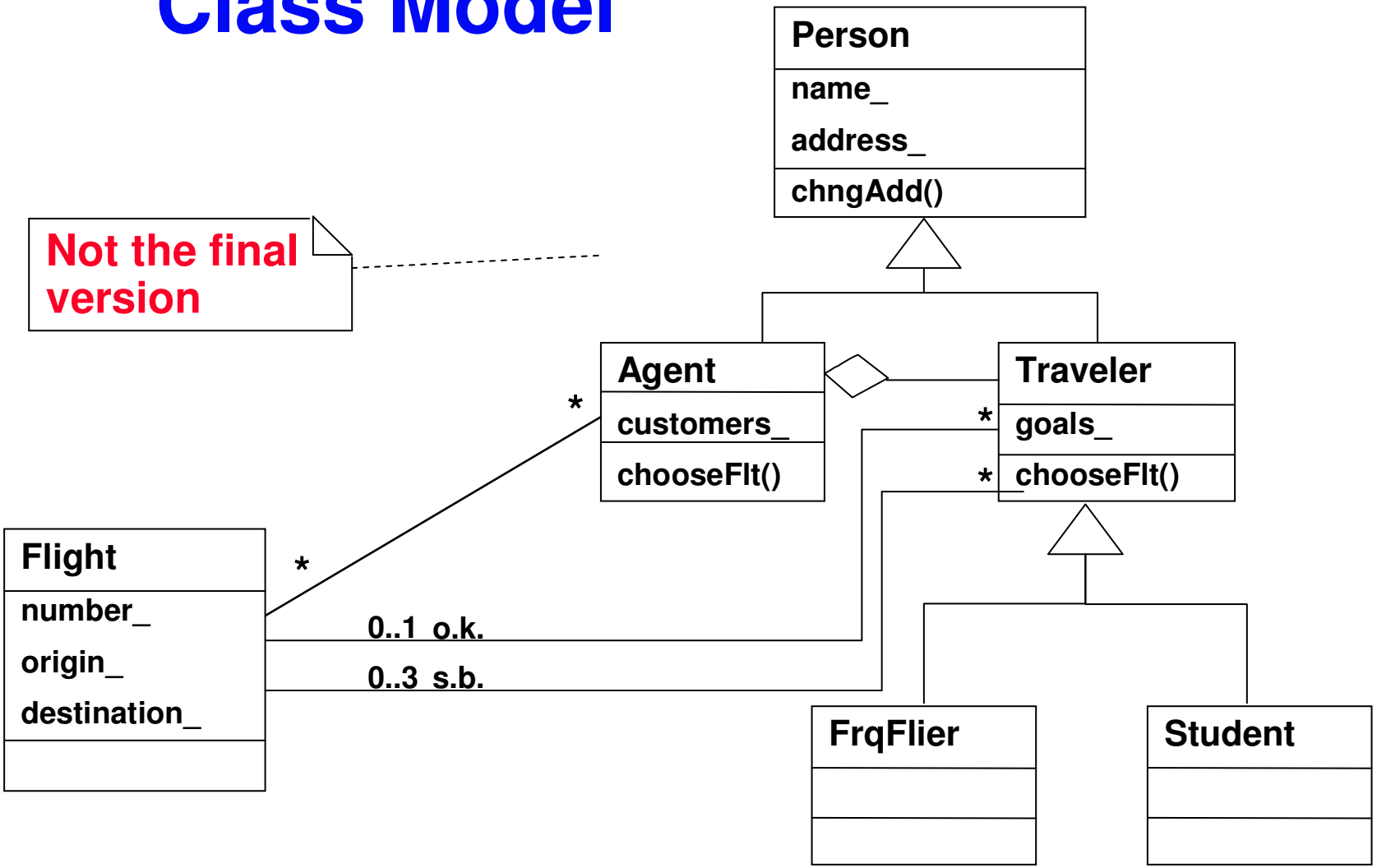
Sequence Diagram



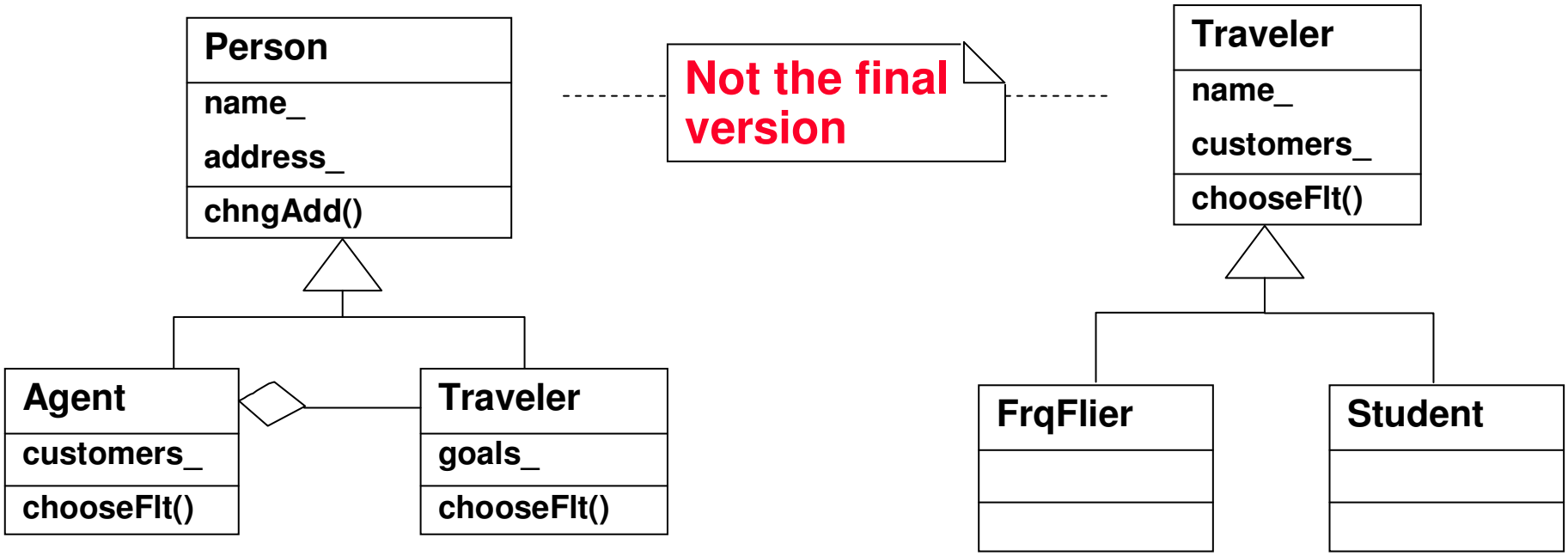
Structural Modeling



Class Model



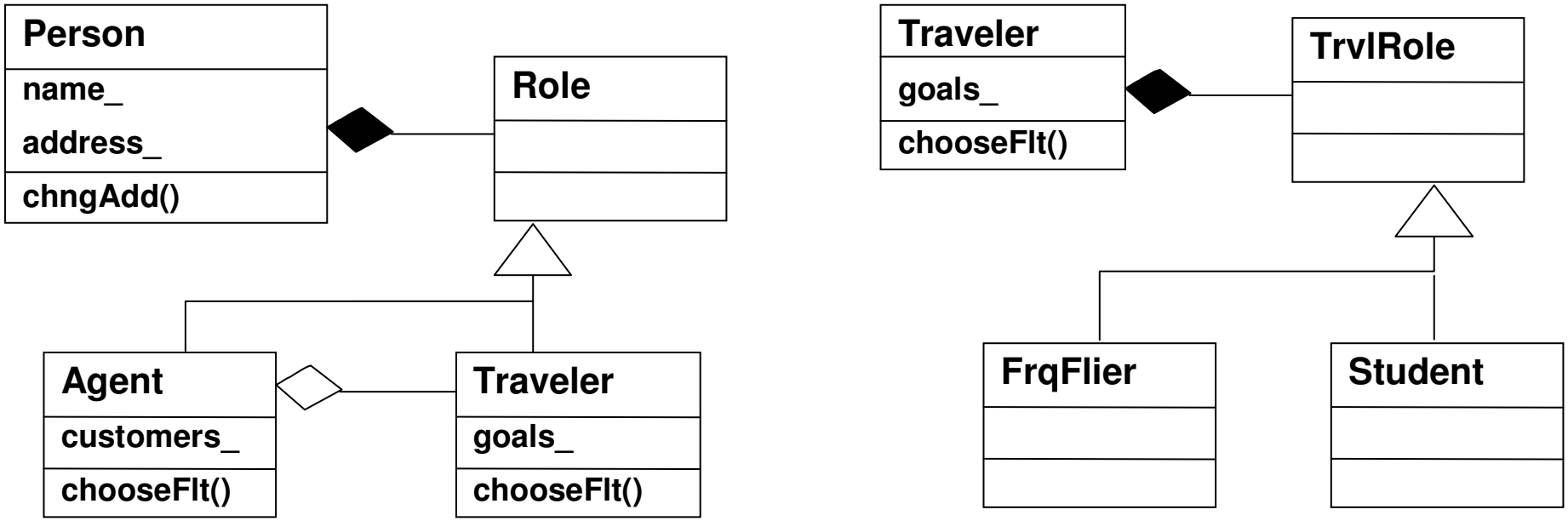
Class Model (cont.)



How can an agent become a traveler? become both?

How can a student become a FrqFlier? become both?

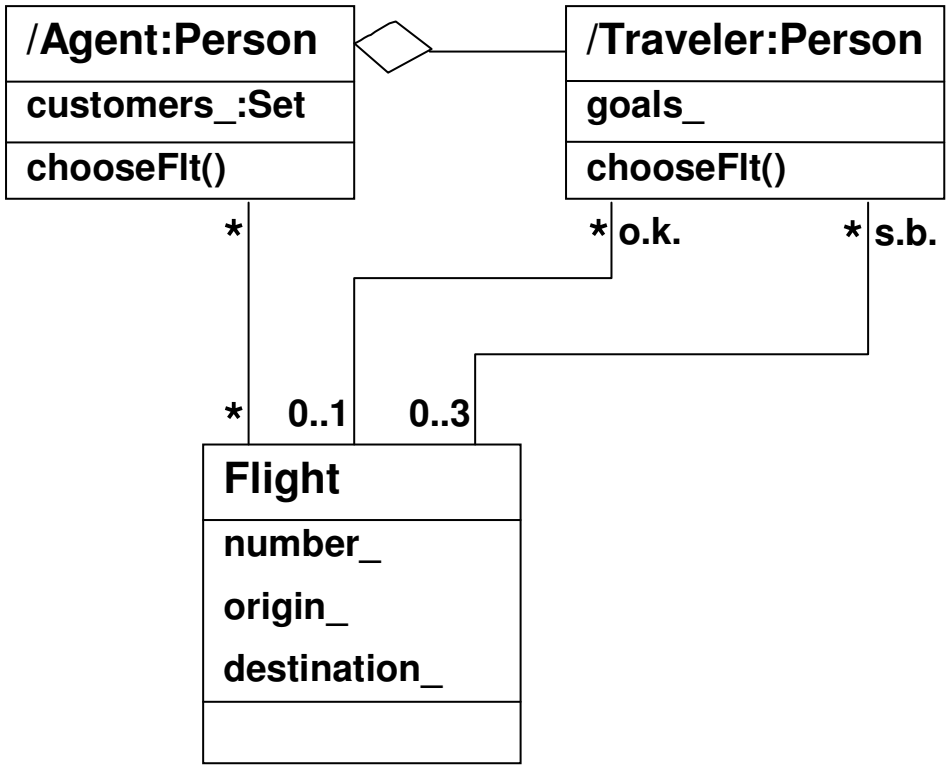
Class Model (cont.)

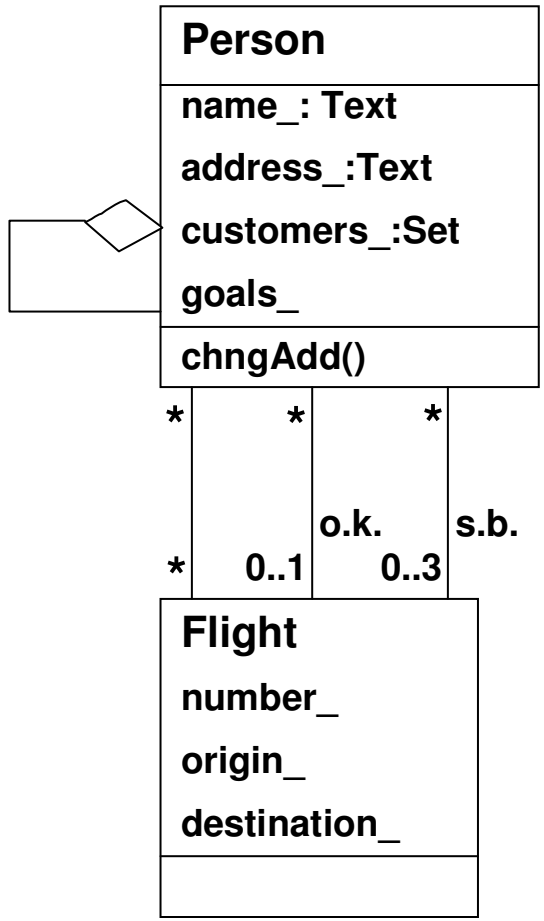


Here we touch the domain of Design Patterns

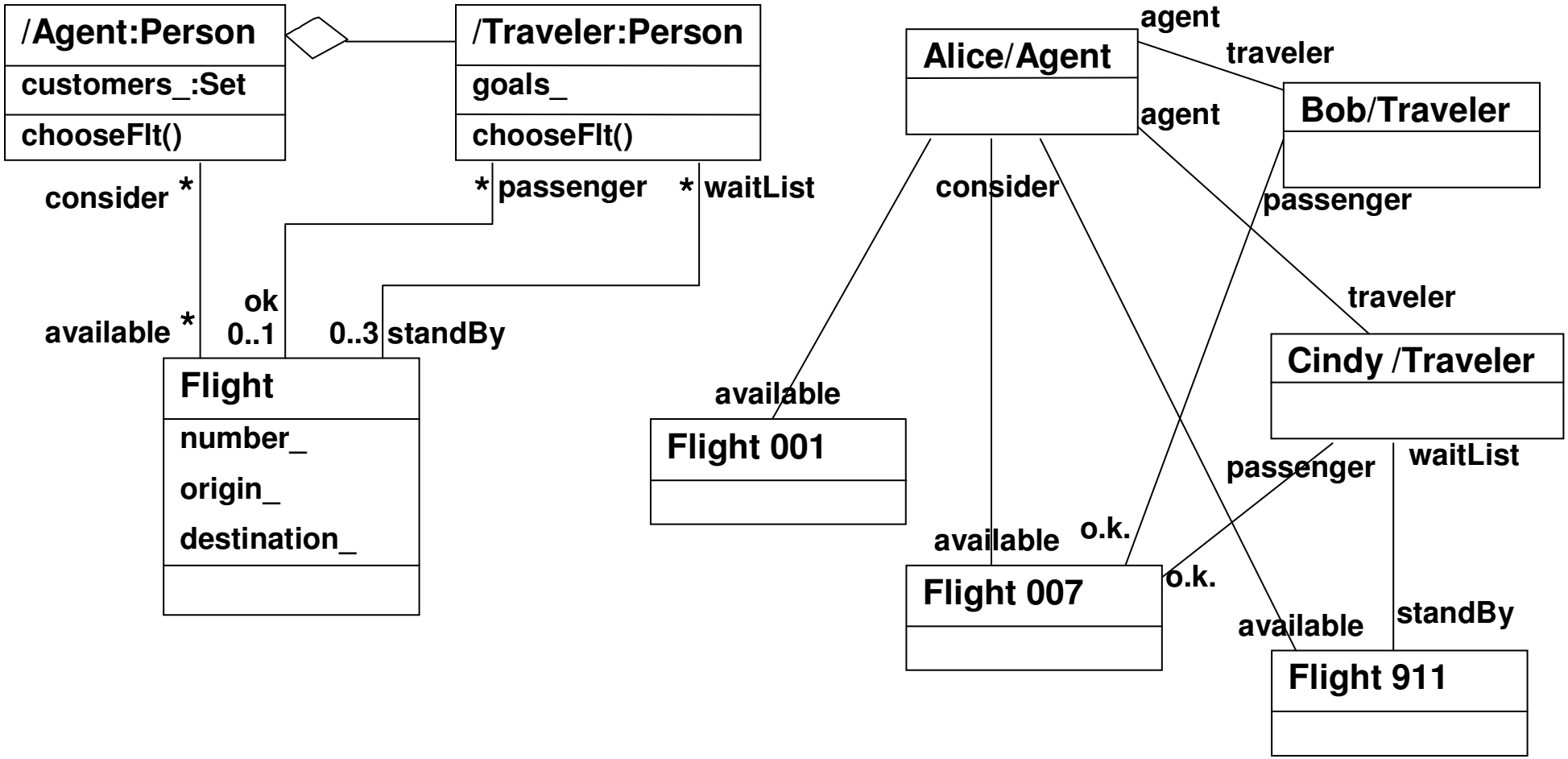
A *pattern* is a solution to an abstract problem

Role Model vs. Class Model

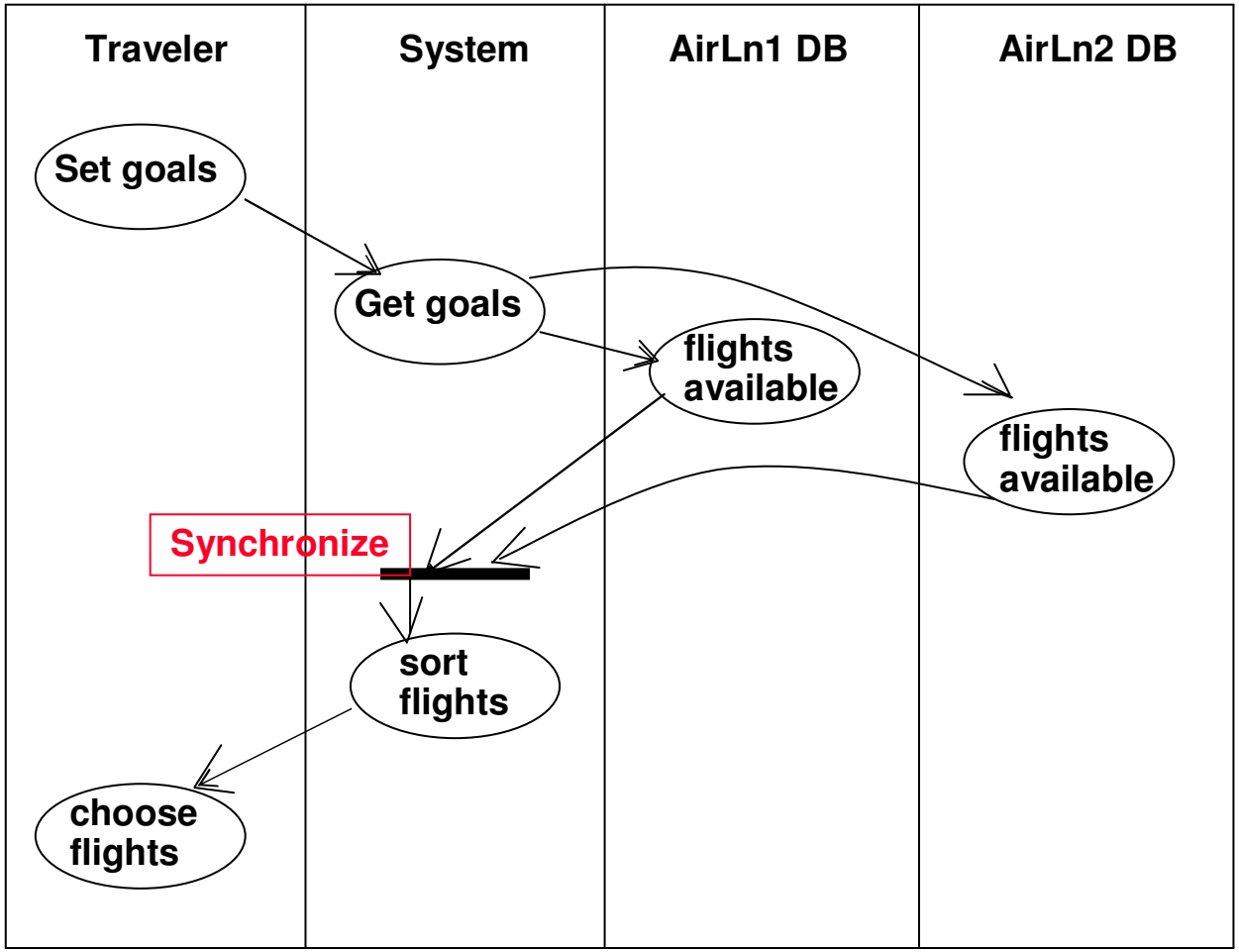




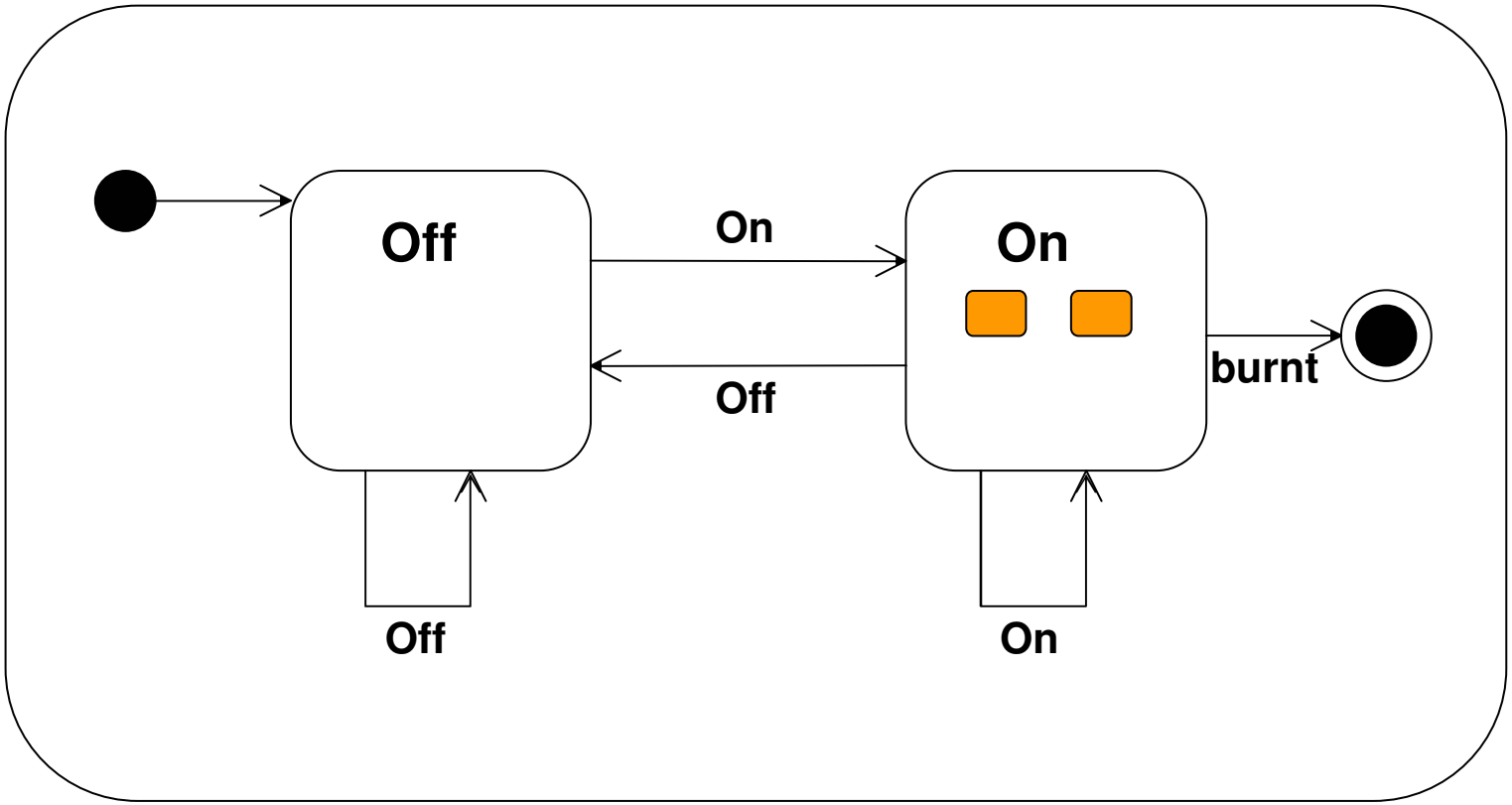
Collaboration Diagram
Specification Level vs. Instance Level



Activity Diagram

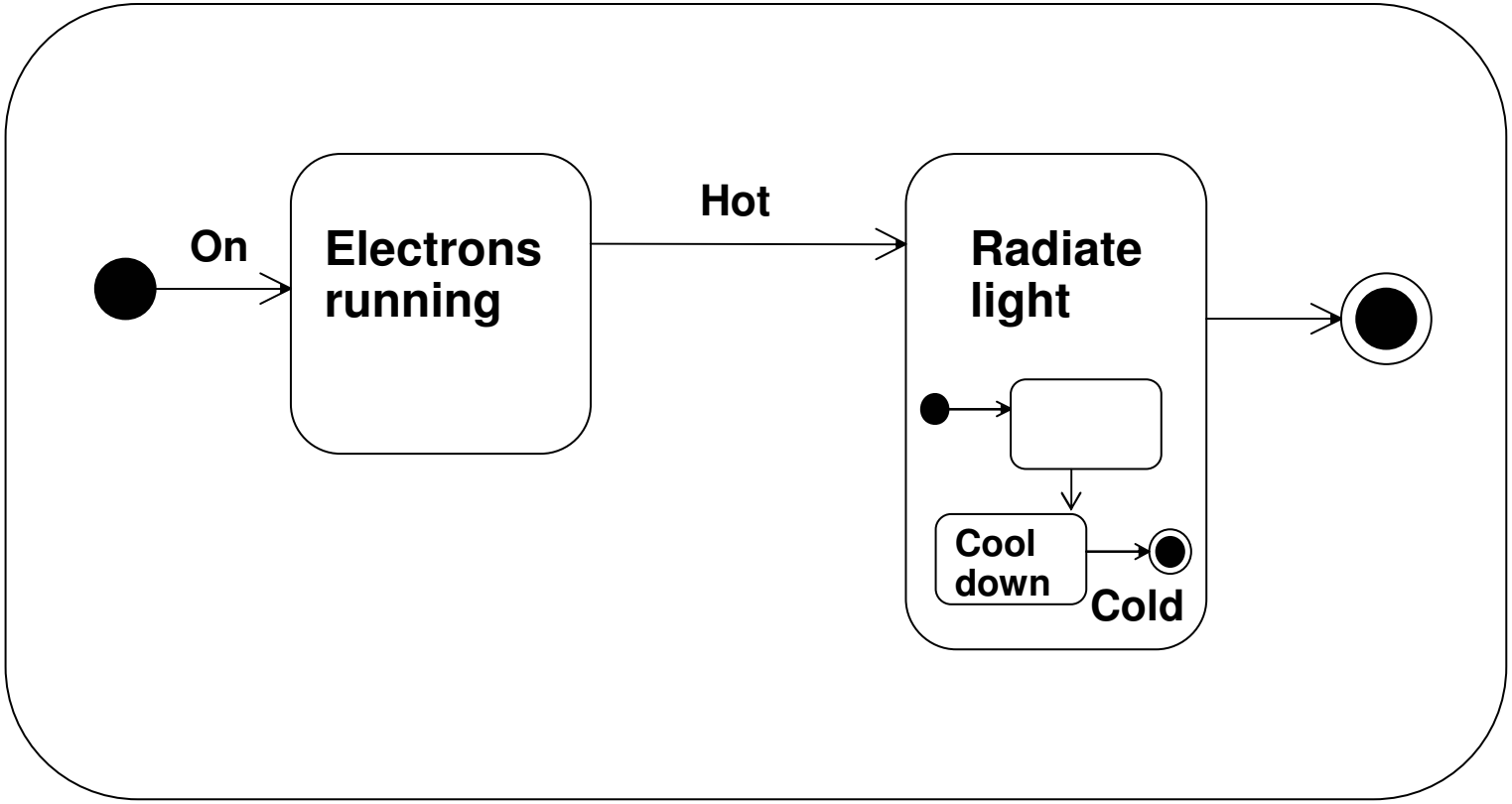


Statecharts



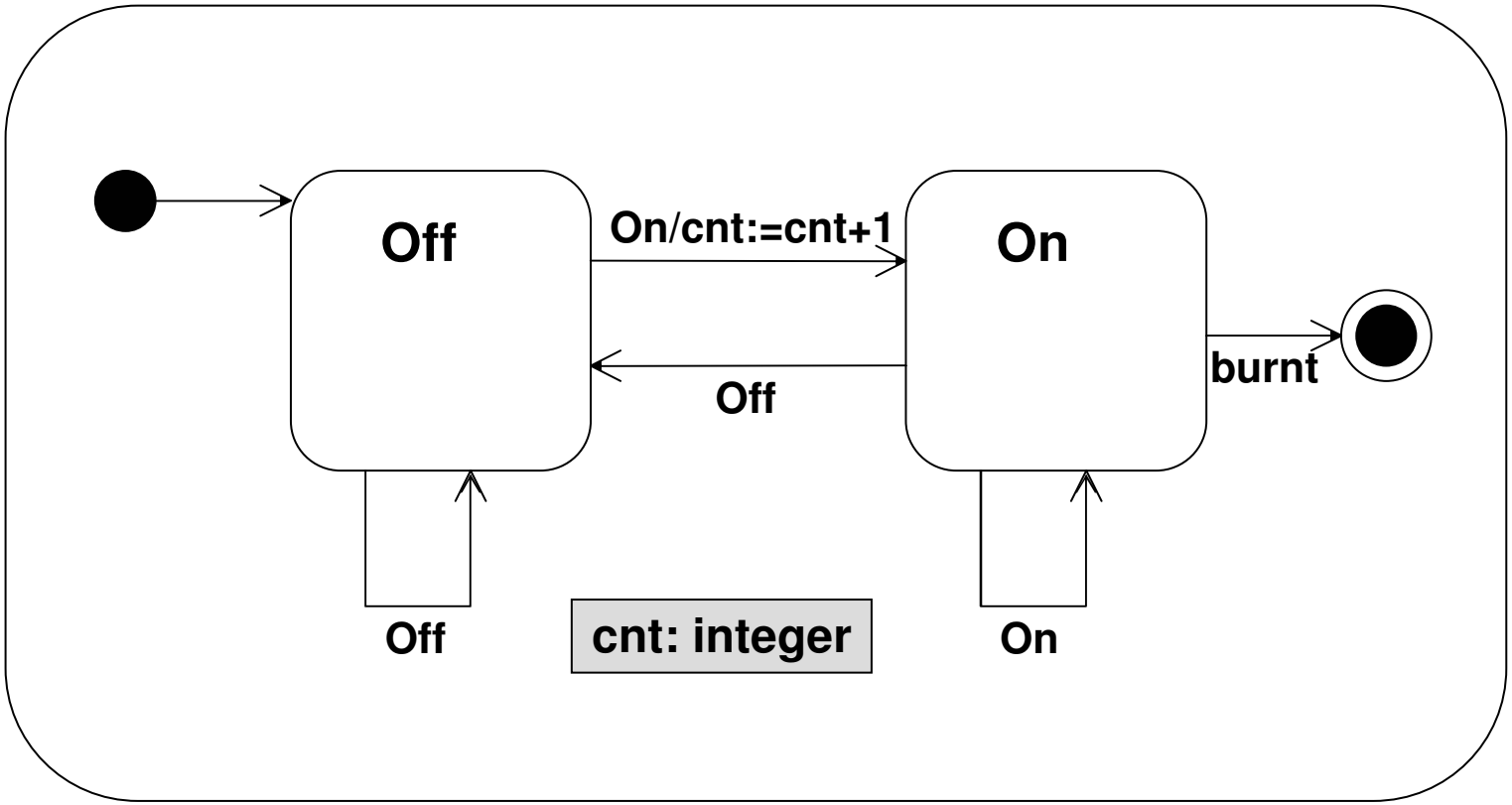
A light bulb

Statecharts (cont.)



On (A light bulb)

Statecharts



A light bulb with a variable

More Construct

- A few examples
- An old example

Associations

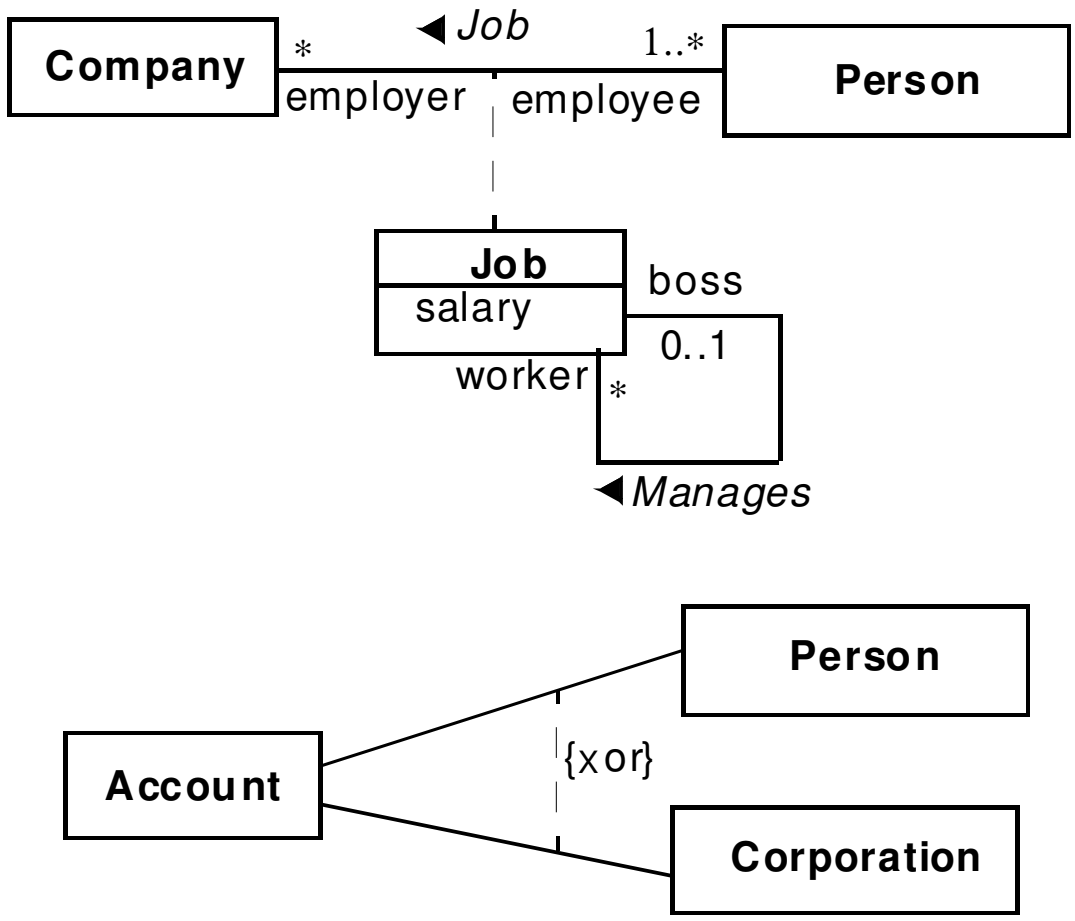


Fig. 3-31, *UML Notation Guide*

Association Ends

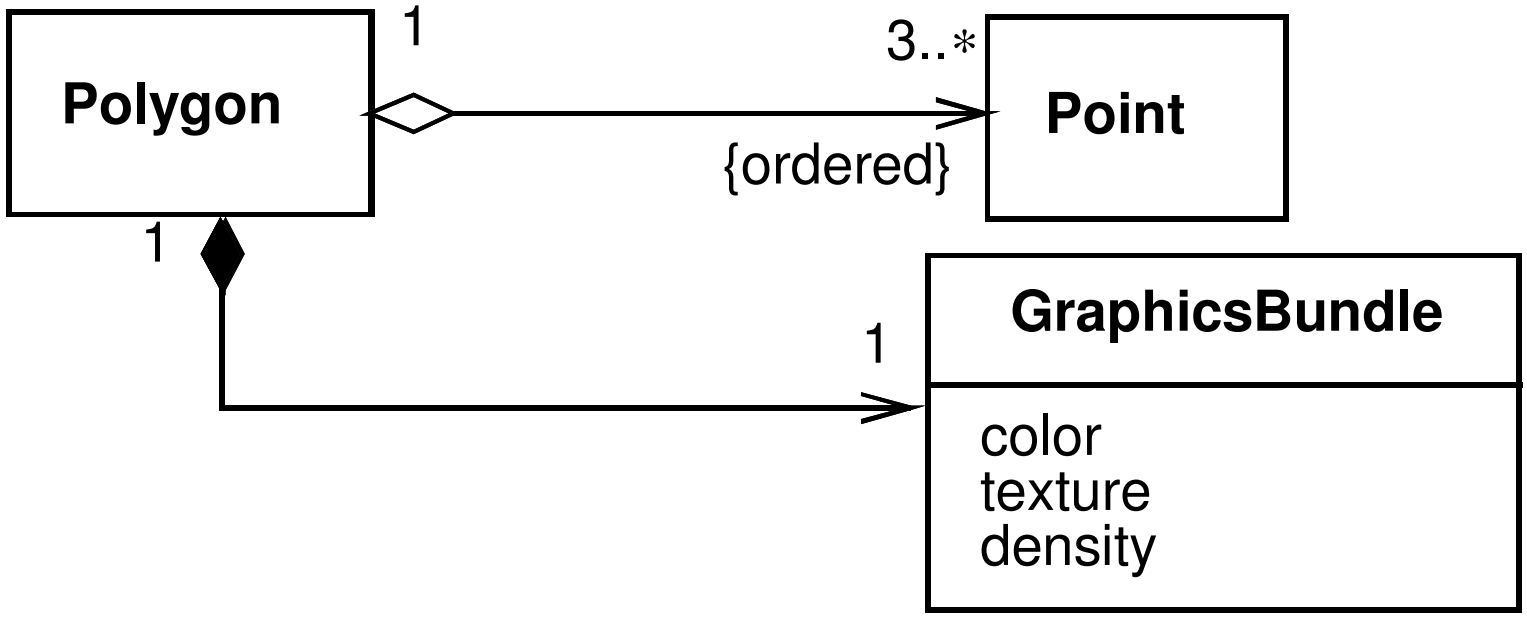


Fig. 3-32, *UML Notation Guide*

Composition 1, 2

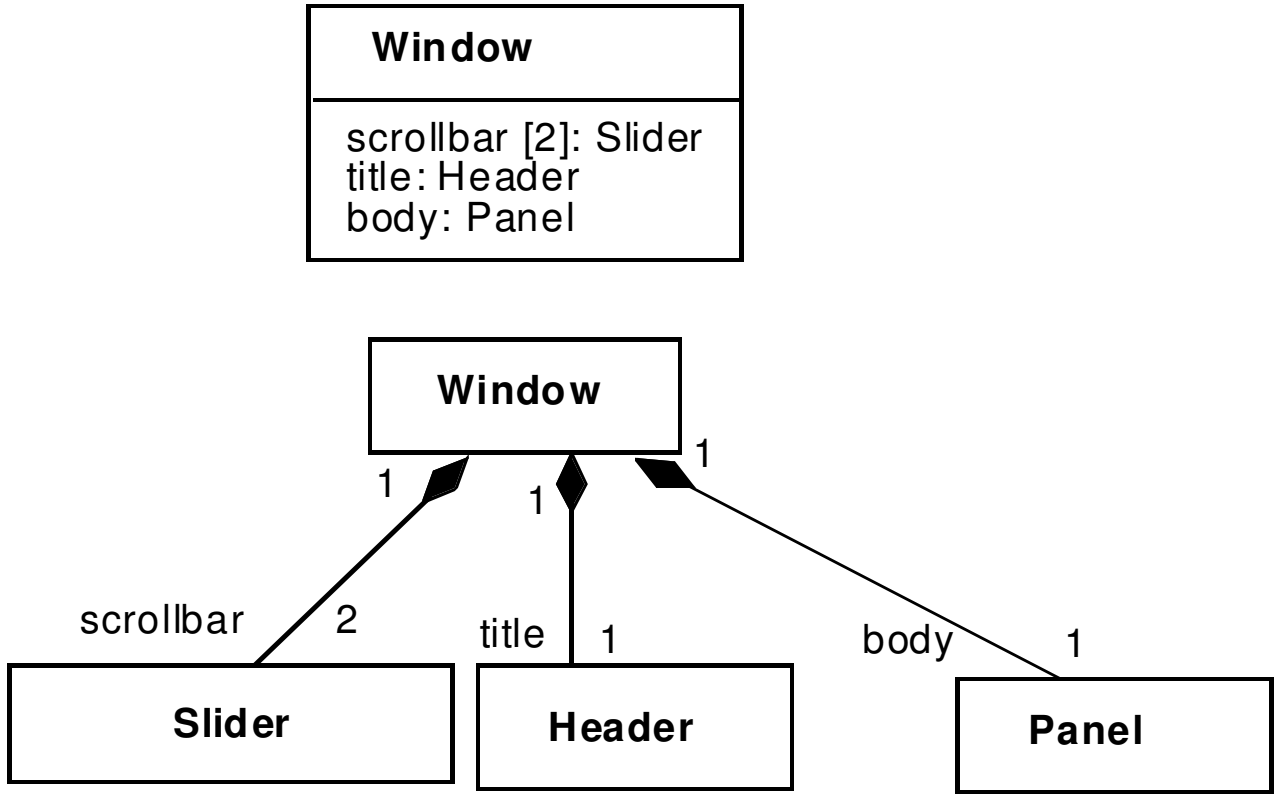


Fig. 3-36, *UML Notation Guide*

Composition 3

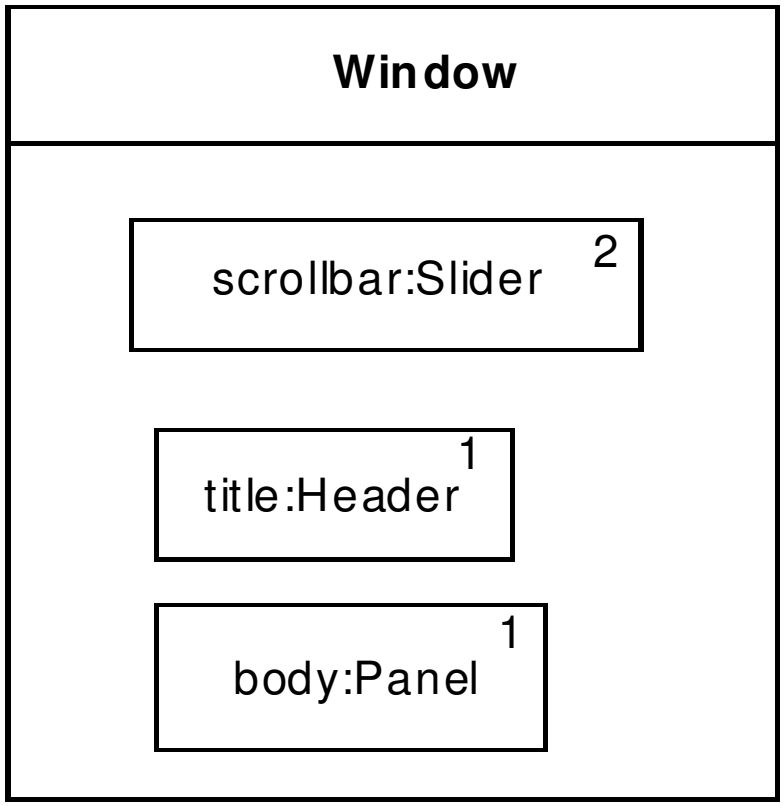
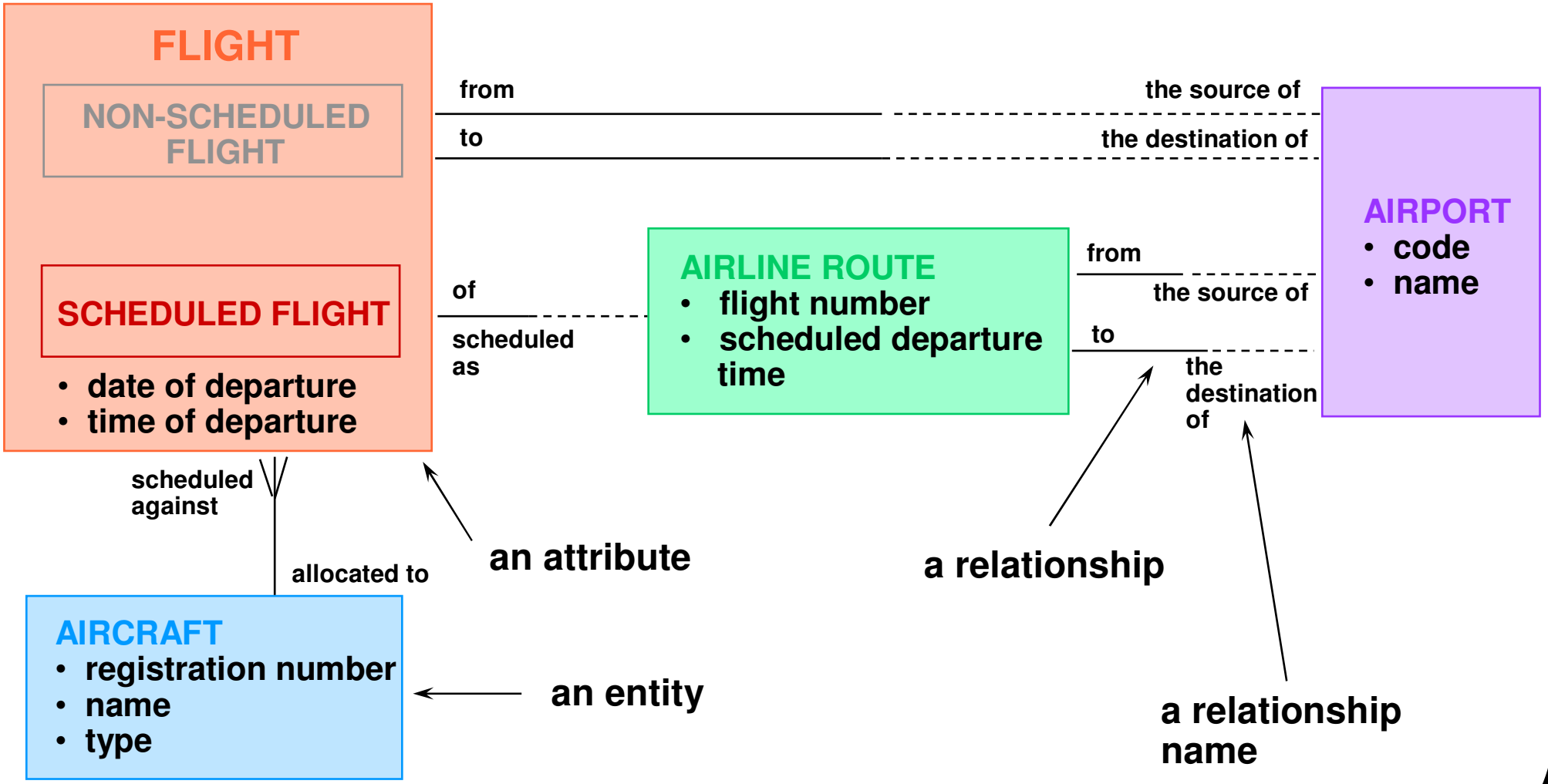


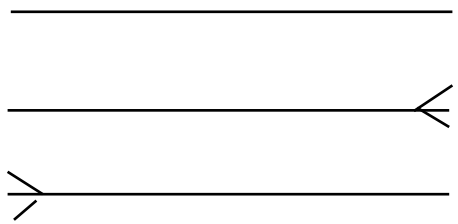
Fig. 3-36, *UML Notation Guide*

Entity-Relationship Diagram (pre-UML)



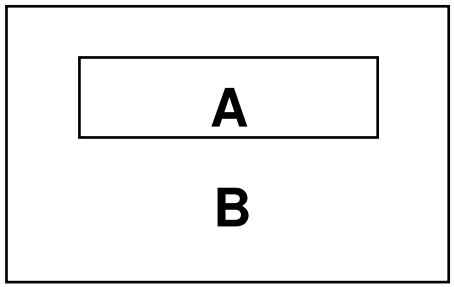
Types of Relationships

- One to one.
- One to many.
- Many to one.



- Mandatory
- Optional

- IS-A



Interested ?

Subsequent courses are

Software-Engineering

Object-Oriented-Programming