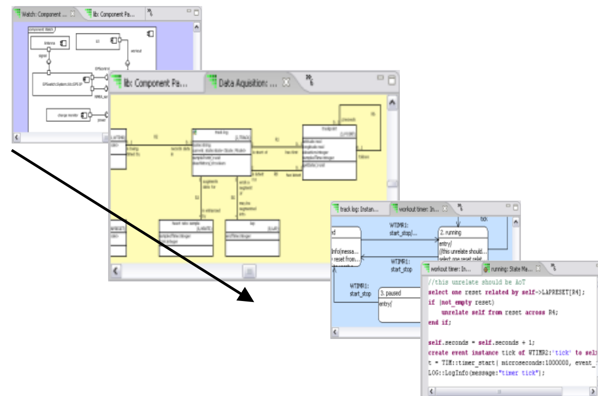
The xtUML method - Building Class Diagrams

- **Analysis** – questioning, thinking, sketching...
  - Descriptive UML diagrams
    - use case, sequence, ...
- **Executable Modeling** – formalizing the analysis:
  - Component Diagrams (partitioning/interfaces)
  - **Class Diagrams (data)**
  - State Machines (control)
  - Activities (processing)
- **Verification**
  - Interpretive Model Execution
- **Code generation**
  - Template and Rule-Based Translation



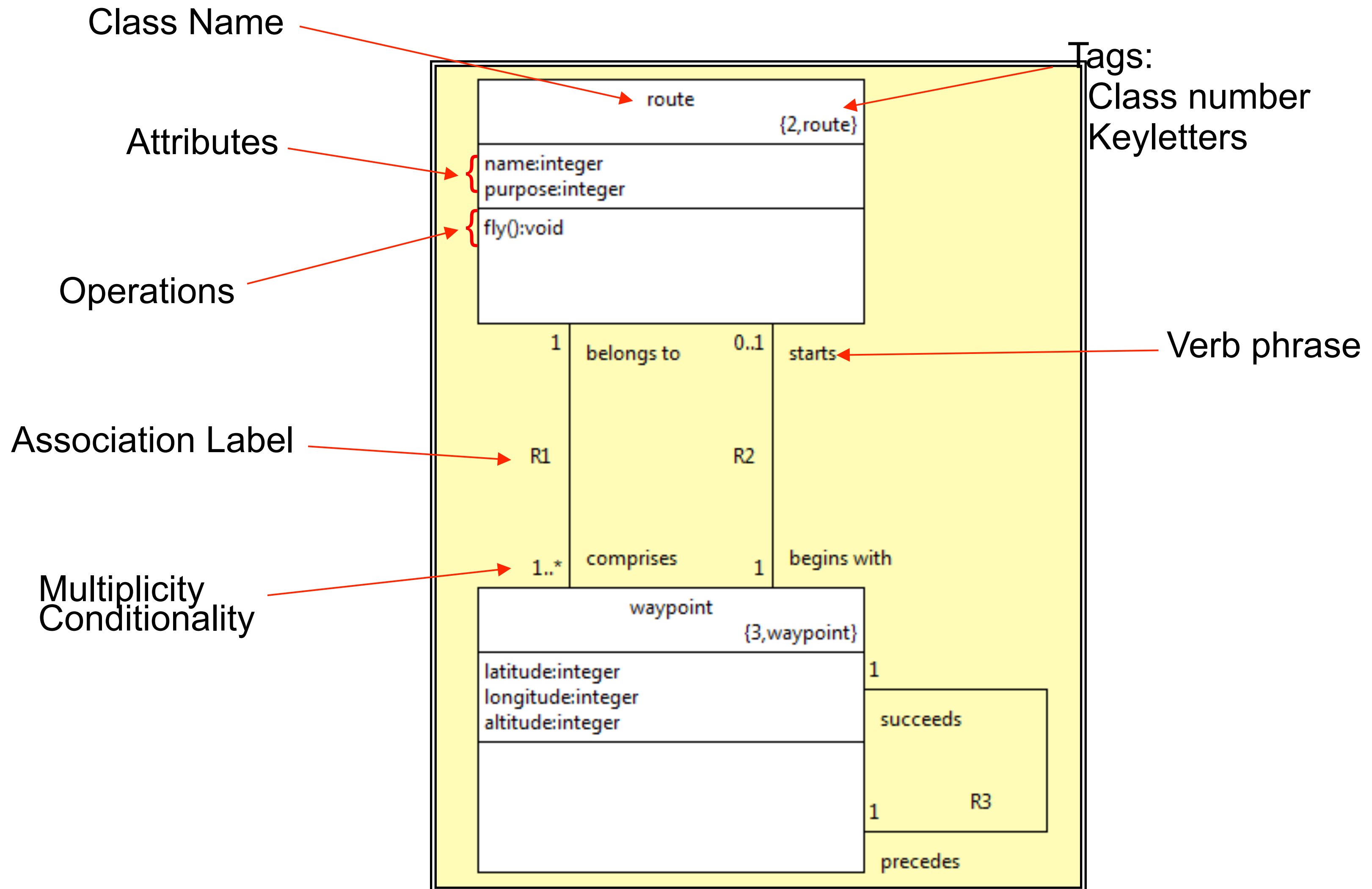
Class Diagrams

- **Identify the types of object the component is concerned with and draw them as classes.**
- **Abstract the characteristics that define the classes; these are the class attributes.**
- **The choice of classes and attributes depends on the purpose of the component.**
- **Draw associations to represent real world relationships that exist between objects.**
- **During execution, instances of these classes and associations will be created as necessary to represent the real world.**

Some Common ‘Types' of Classes

- Tangible objects
  - Laser, Mirror, Motor...
- Roles:
  - User, Customer, Peripheral,...
- Discovered classes:
  - Account, Packet header, Proxy,...
- Incident:
  - Button press, Data sampling operation...
- Interaction – between other classes:
  - Channel assignment, Process step,...
- Specification – shared characteristics:
  - Common data shared by multiple instances of other classes

Class Diagram Elements



Class Blitz – Exercise

- Build a class diagram of a clock
  - Consider what discrete parts are in a clock
  - Capture relationships between parts of the clock
  - Place attributes in each class

Associations

- **An association is a domain relationship between classes.**
- **Associations can be navigated to locate particular class instance.**
- **Associations can be created in a class diagram as lines between classes.**
- **An association relating to a class instance of the same type is called a Reflexive Association.**

Representing data relationships

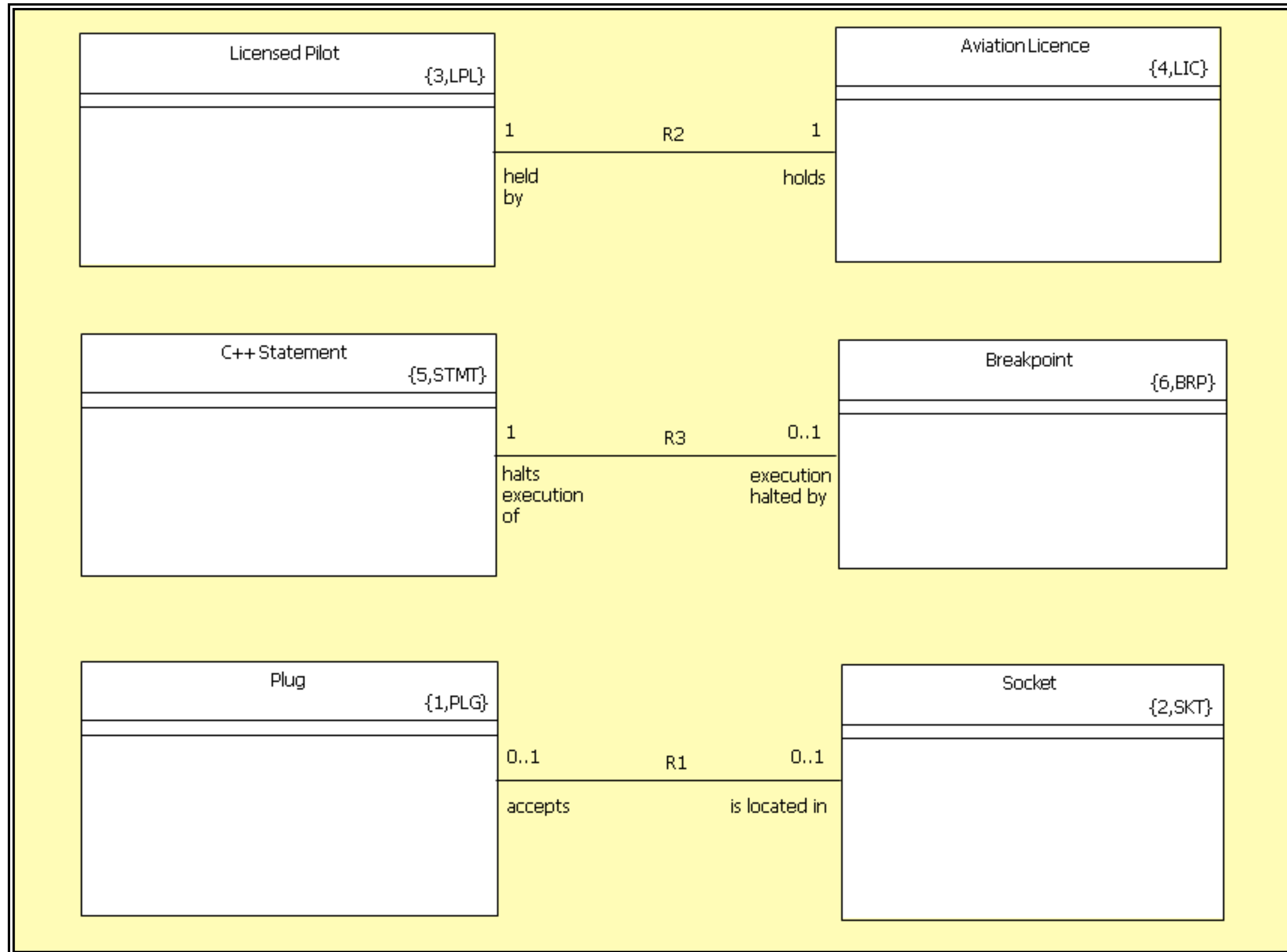
- A representation of a relationship between real world things
- Binary associations – between two participant classes
  - One-to-one [1:1]
  - One-to-many [1:M]
  - Many-to-many [M:M]
- Unconditional, conditional, bi-conditional
  - Either or both of participant classes may not always be related.
- Association is 'reflexive' when each 'end' is the same class
  - Navigation must then use verb phrase to distinguish 'direction'.
- Any binary association may have its own characteristics
  - Add an association class with attributes, associations, state model...
- Generalization, specialization use a special association
  - No verb phrases needed: 'is a' assumed

Multiplicity & Conditionality

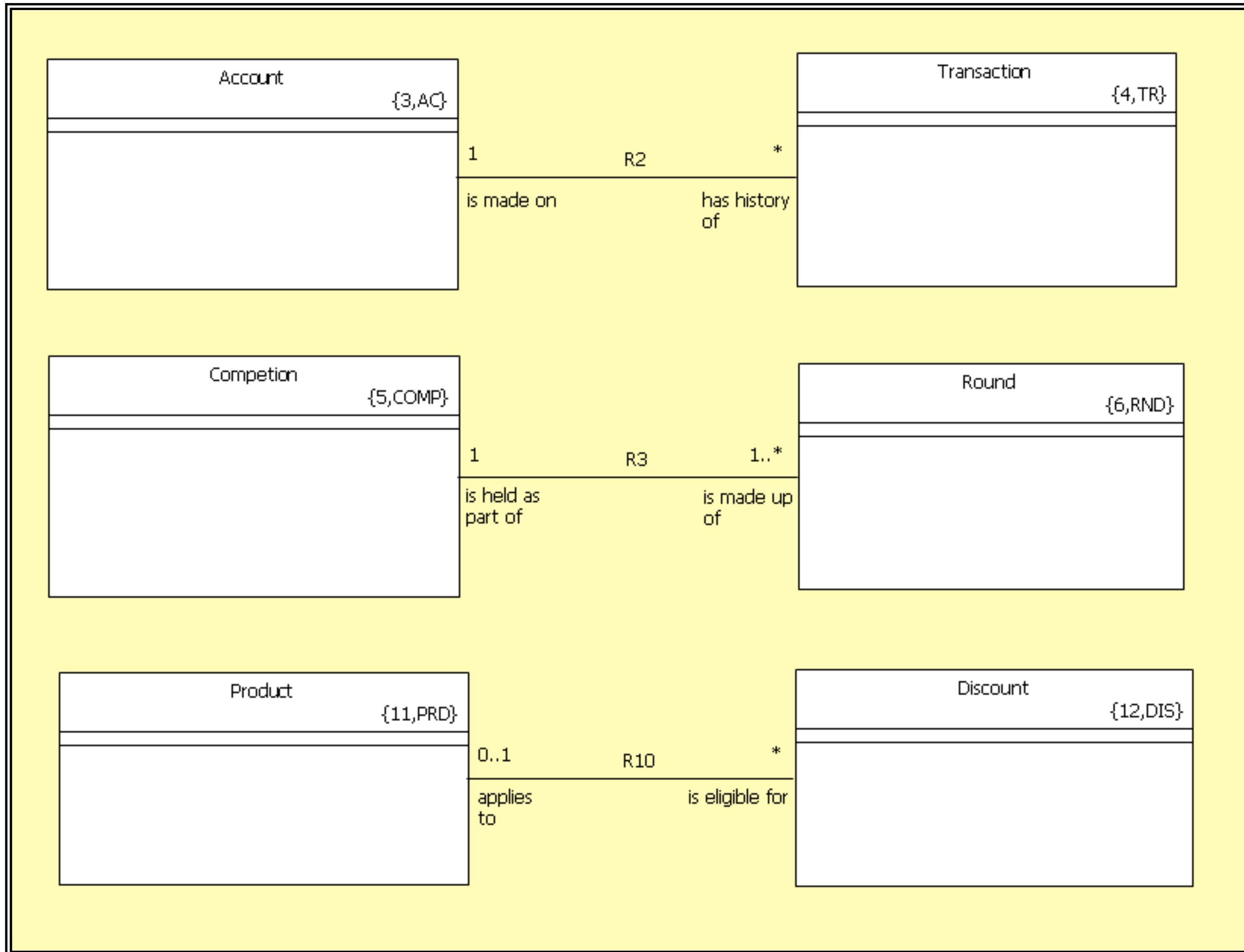
- **Multiplicity – Can there be more than one?**
- **Conditionality – Must there be one at all?**

Conditionality	Multiplicity	Nomenclature
conditional	one	0..1
conditional	many	*
unconditional	one	1
unconditional	many	1..*

Examples: One

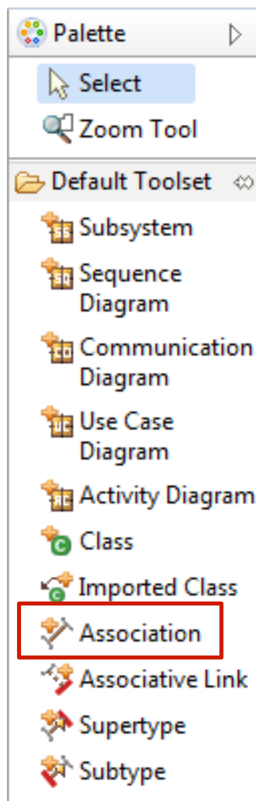


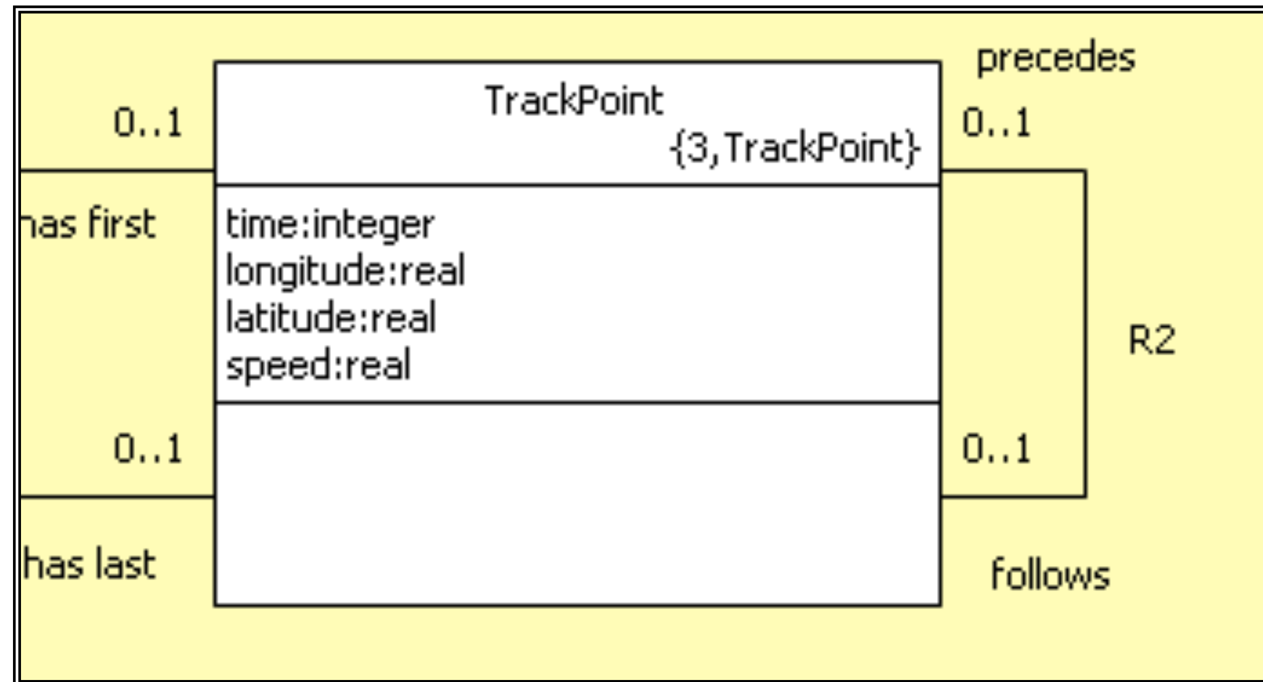
Examples: Many



Reflexive associations

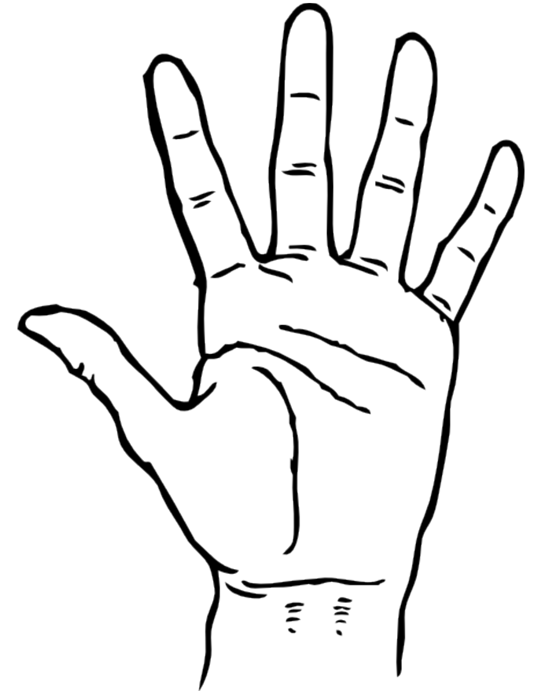
- Associations between a pair of instances of the same class
- Generally conditional with multiplicity of 1





Reflexive Associations: Exercise

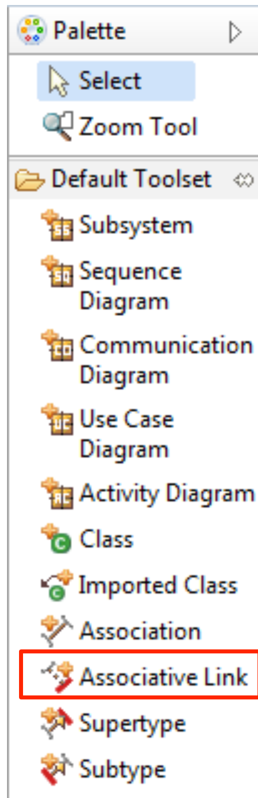
- Create a class diagram of a hand
- Classes to use: palm, wrist, finger
- Consider conditionality and multiplicity of associations
- Use at least one reflexive association

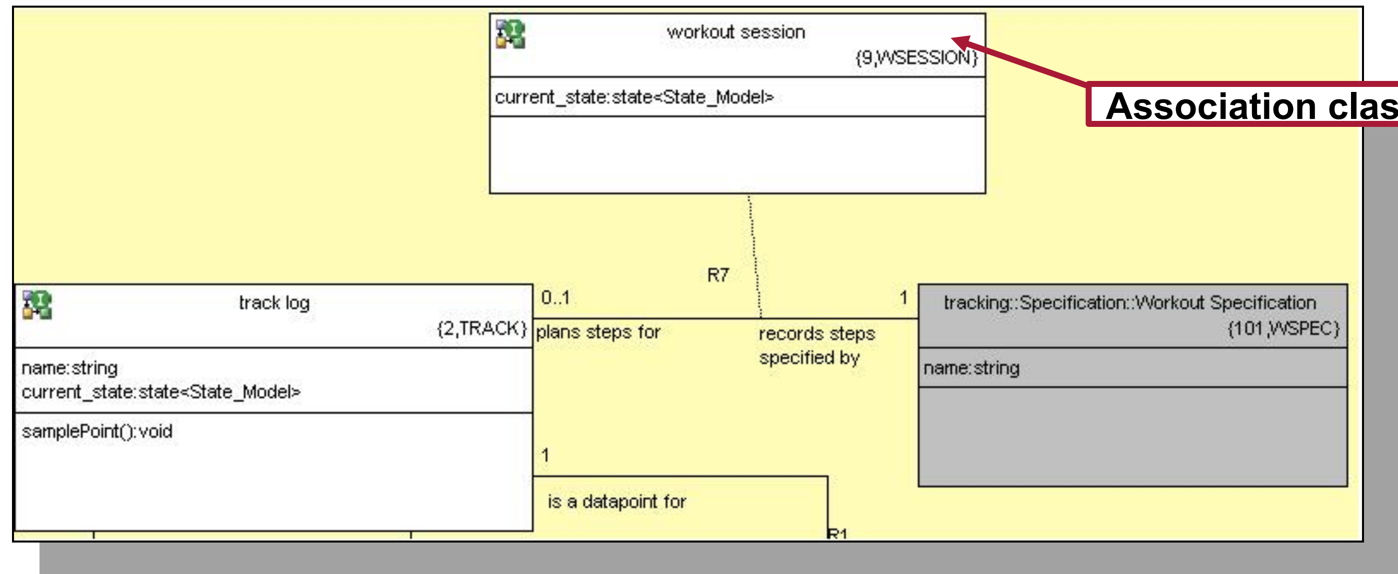


Lab 1: Exercise 4

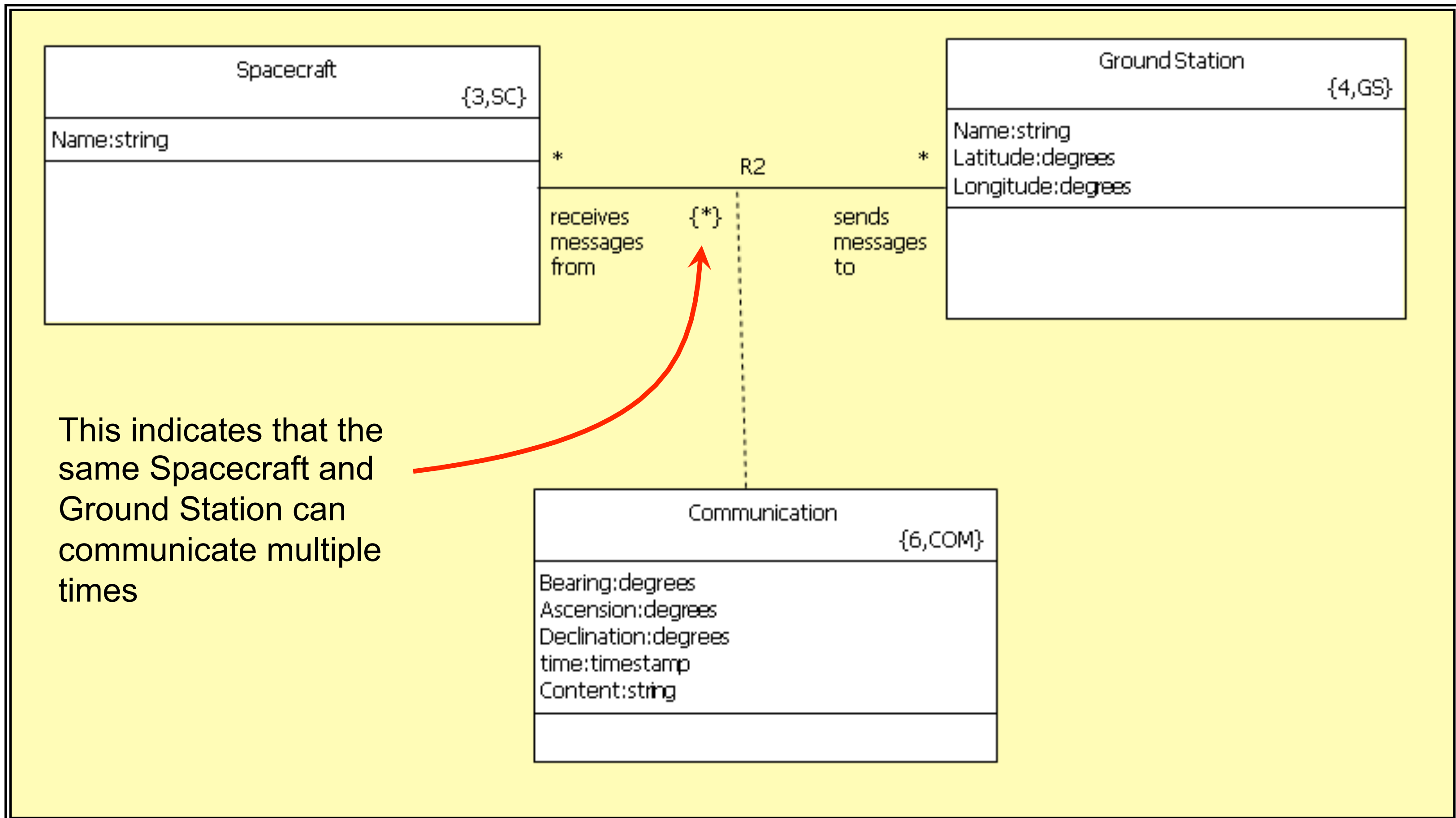
- **Defining the functionality of our GPS Watch System**

Association classes





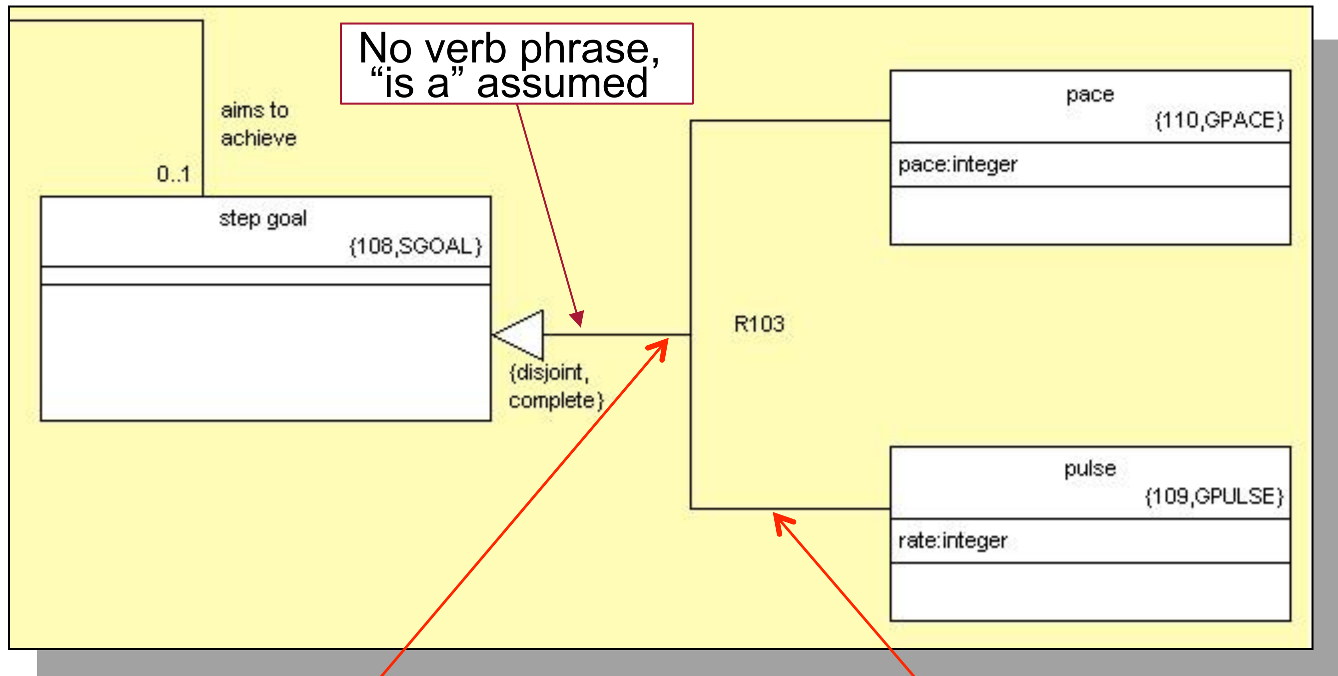
Another Associative Example

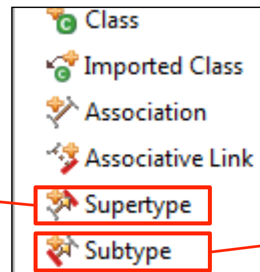


Generalization

- Not all languages support inheritance
- Multiple instances – allows reclassification/subtype migration
- Much like 1:1 association
- No polymorphic operation – only events
- Use when there are interesting associations on subtypes
- Model compiler can make may implement it with inheritance (where available) or even decide to flatten the structure
- No calls to super
- Future – inherited properties: operations and attributes

Generalization Example





Draw from parent type to intermediate point

Draw from child type to intermediate point

Construction Quality Class Diagrams

- **For associations, consider whether the relationship always holds true – conditionality – at each end.**
- **Consider whether there may be more than one participant class at each end - multiplicity.**
- **Choose meaningful names for classes; meaningful phrases for associations.**
- **Write descriptions as you go! They may cause you to think again about your analysis.**

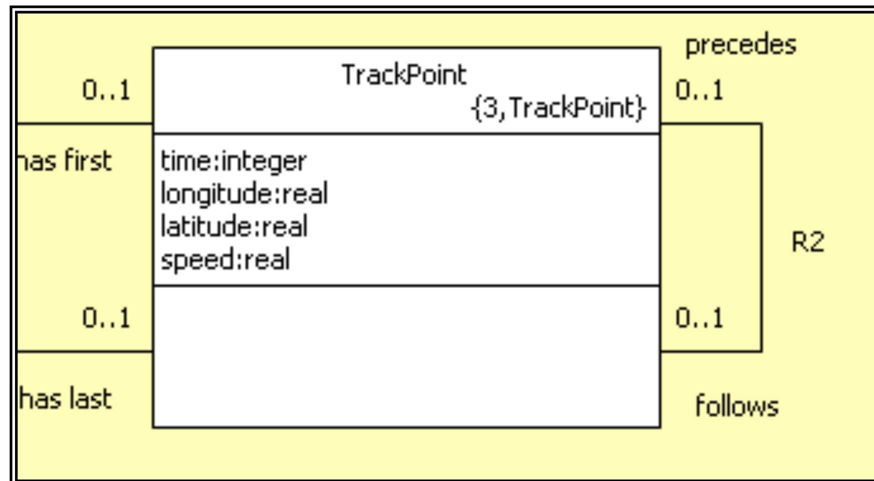
Attributes

- **An attribute represents a characteristic shared by all of the instances of a class**
- **Descriptive attributes represent inherent characteristics**

  Trackpoint has latitude, longitude, elevation.

- **Naming attributes represent arbitrary labeling of instances**
- **Which characteristics are considered relevant depends on the point of view – the "purpose" - of the application**

  Trackpoint also requires time, but not air temperature, windspeed...



Quality Attributes

- Each instance has exactly one value for each attribute
  - Lots of “Not Applicable” may indicate the class should be split
  - BP supports arrays as attributes, but these are intended to make interfacing with legacy code easier
- All attributes are Atomic
  - A need to 'parse' values suggests using separate attributes
- Every attribute should characterize the entire instance
  - Attributes related to other concepts should be in a separate class
  - Attributes related to only a subset of the instance population suggests that some generalization is needed.

Identifying Attributes

- Some attributes abstracted for a class represent values that may be used to uniquely find an instance, e.g.
  - License Number
  - WayPoint ID
  - Ticket Number
- Highlighting these attributes helps understanding of the subject matter being modeled
- In xtUML we call such attributes 'Identifying Attributes'

Identifiers

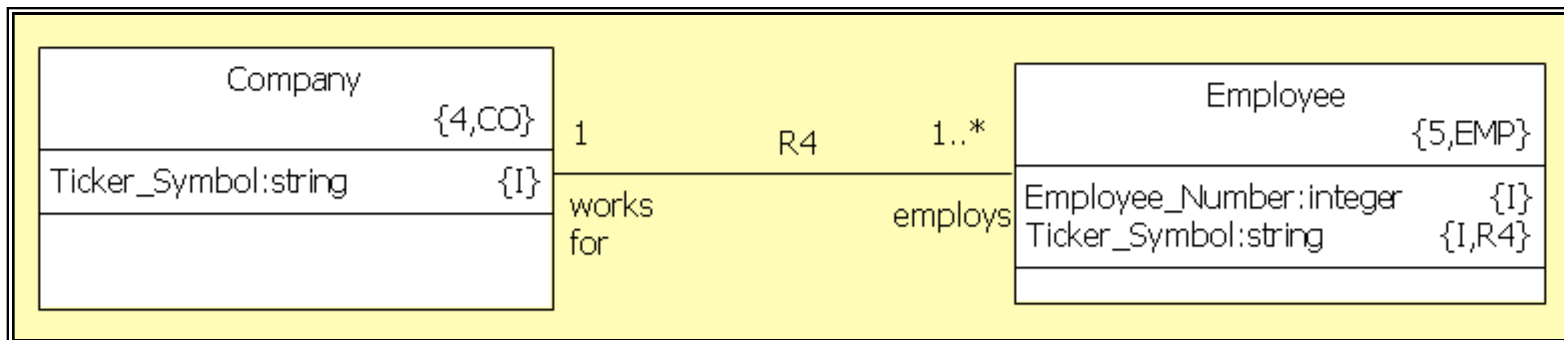
- **Sometimes, the value of more than one attribute is required to uniquely identify an instance**
- **A group of one or more attributes required for instance identification is known as an ‘Identifier’**
- **A class may have more than one Identifier as required (BridgePoint supports up to three Identifiers)**

Association Formalization

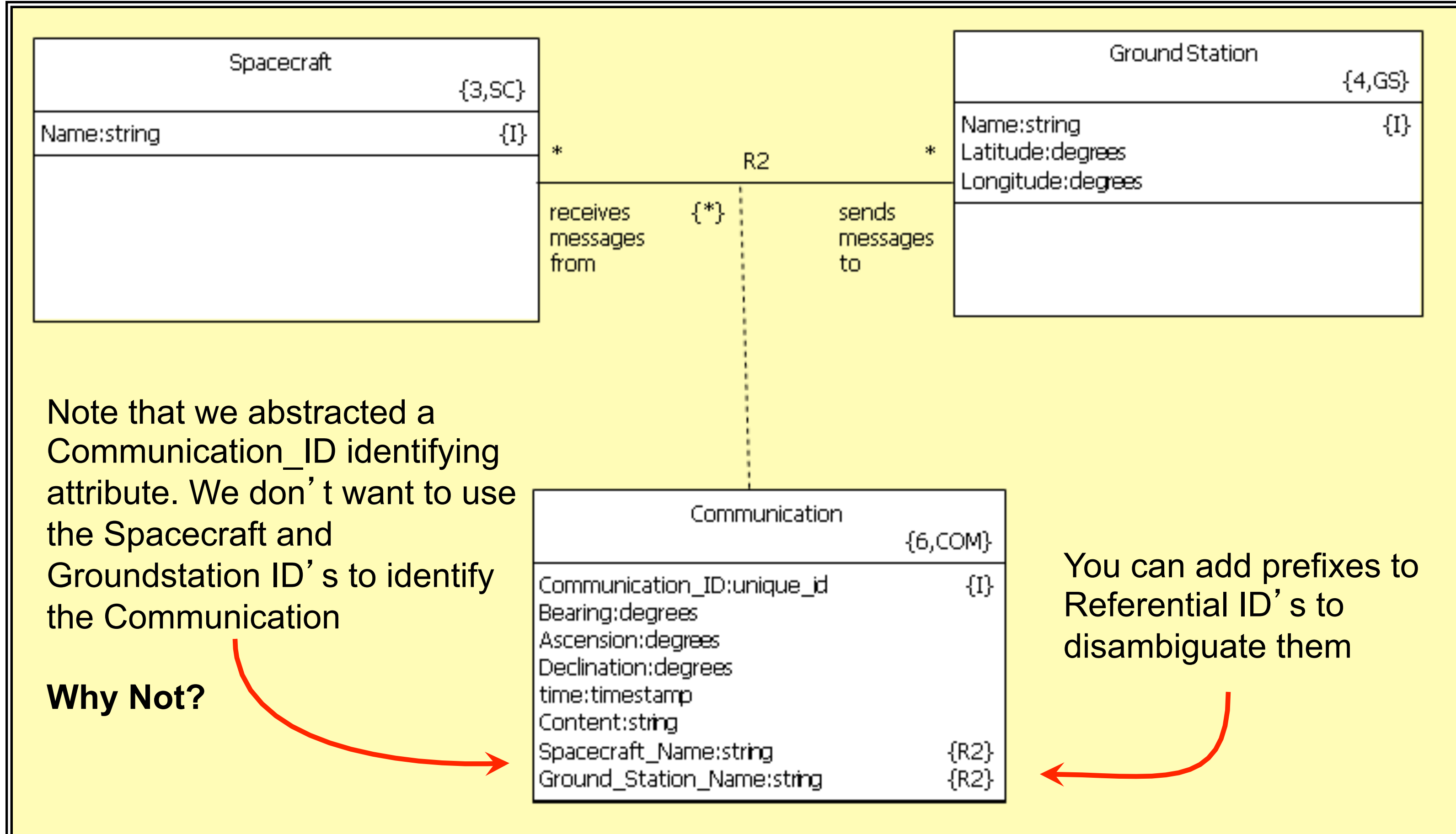
- Once a class has an identifier, it is possible formalize associations it participates in using its identifier

- When this is done, a ‘copy’ of the attribute set is added to the attributes of class at the other end of the association being formalized

- Attributes migrated in this way are called ‘Referential Attributes’



Formalized Spacecraft Example



High Precision Modeling: Referential Combination

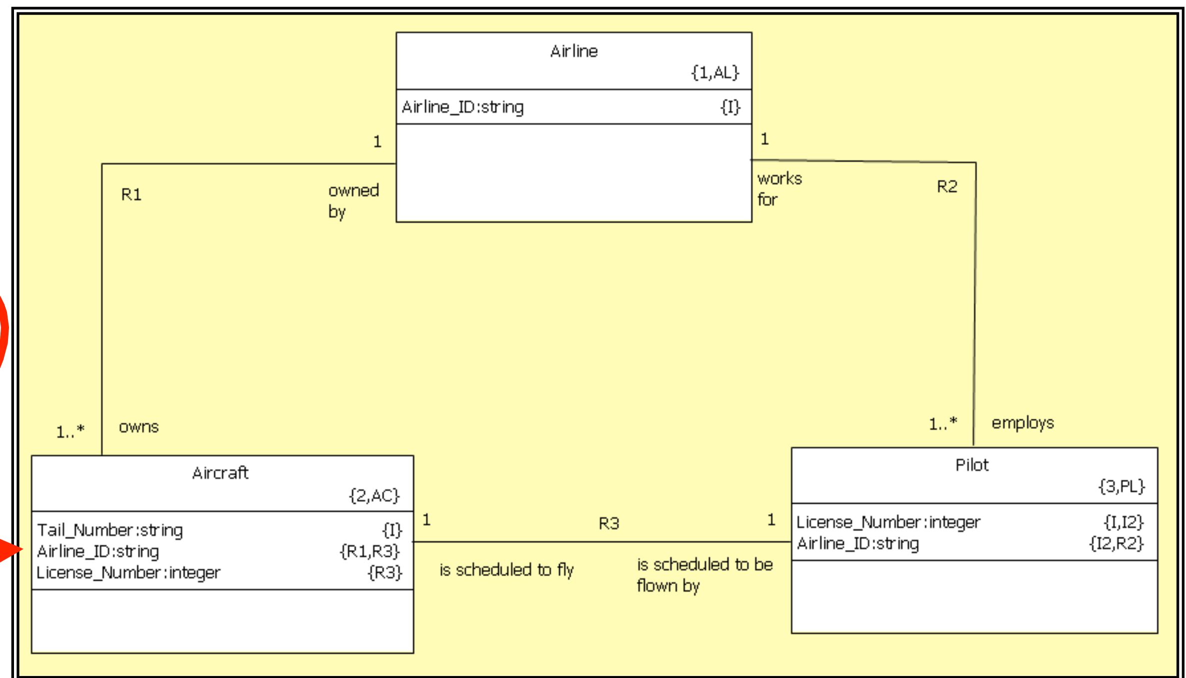
- Where required, referential attributes may be combined

- At first glance, it looks like this saves storage space. However, this is not the reason to do it. Combining referential attributes means something very precise

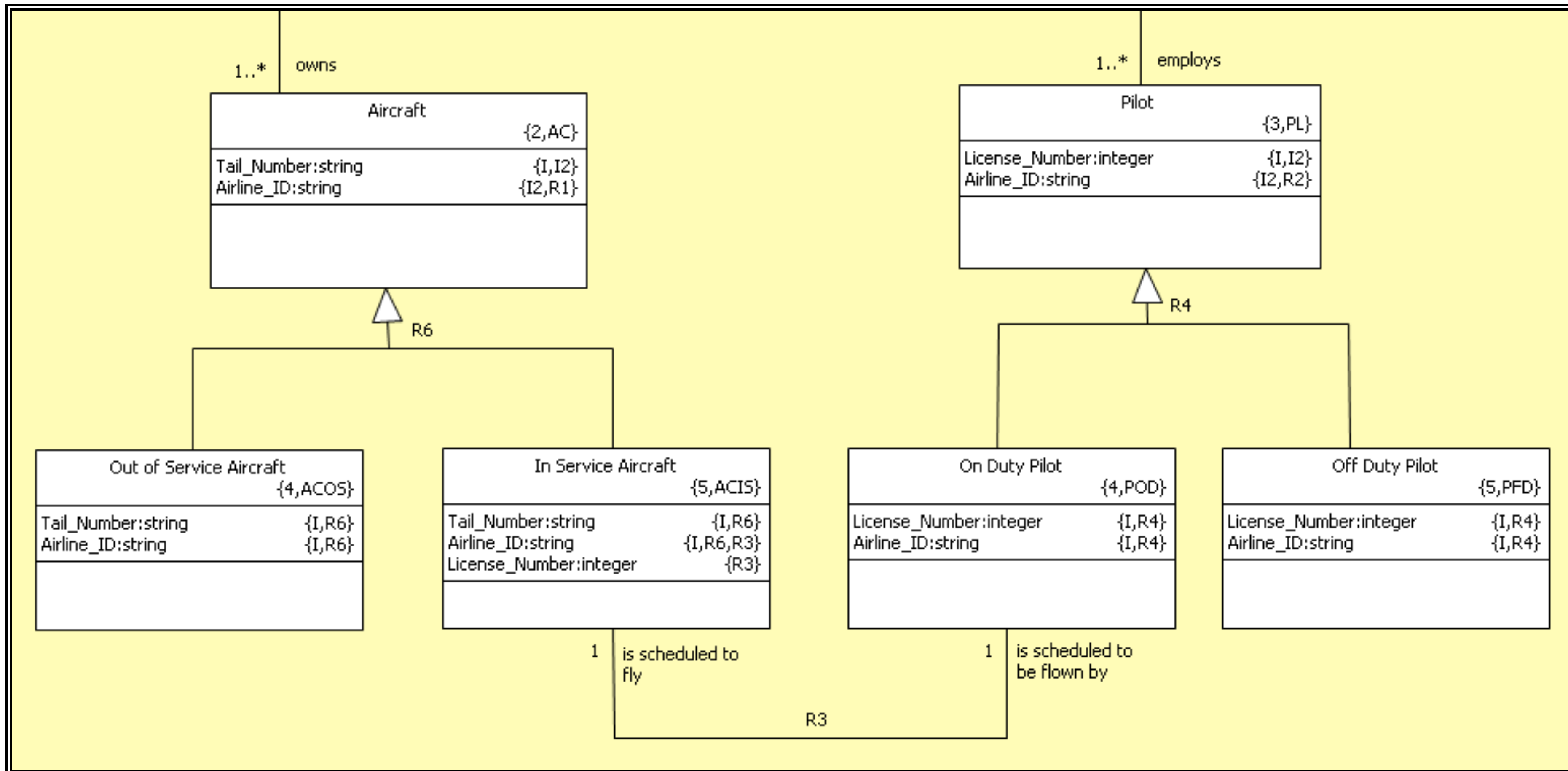
- Example:

Airline_ID is combined here.

According to the rules of attribution covered earlier, Airline_ID may only take one value, this combined referential mandates that a pilot may only fly aircraft that his or her airline owns



Leverage Generalization



Here, we have documented the constraint that only In Service Aircraft and On Duty Pilots will be doing any flying.

Paper Exercise: Putting it all together

- The application is a Electronic schematic drawing and PCB layout tool.
  - We are required to analyse the Library for it. This Library is to contain Integrated Circuits (for the schematic editor) and their associated physical Packages (for the PCB layout editor).
  - Integrated circuits come in a variety of different packages to support different deployments, prototyping, large scale production, hardened applications (military space etc.).
  - Every integrated circuit has connections for power, digital input and output etc.
  - Every Package has physical pins that carry these connections to the outside world.
  - For any given package, we must know which pin carries which connection.
  - Pins may be left unconnected, but every connection must get to the outside or else the Package cannot function.

Paper Exercise: Solution