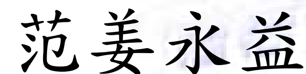
Analysis: Object Domain Analysis

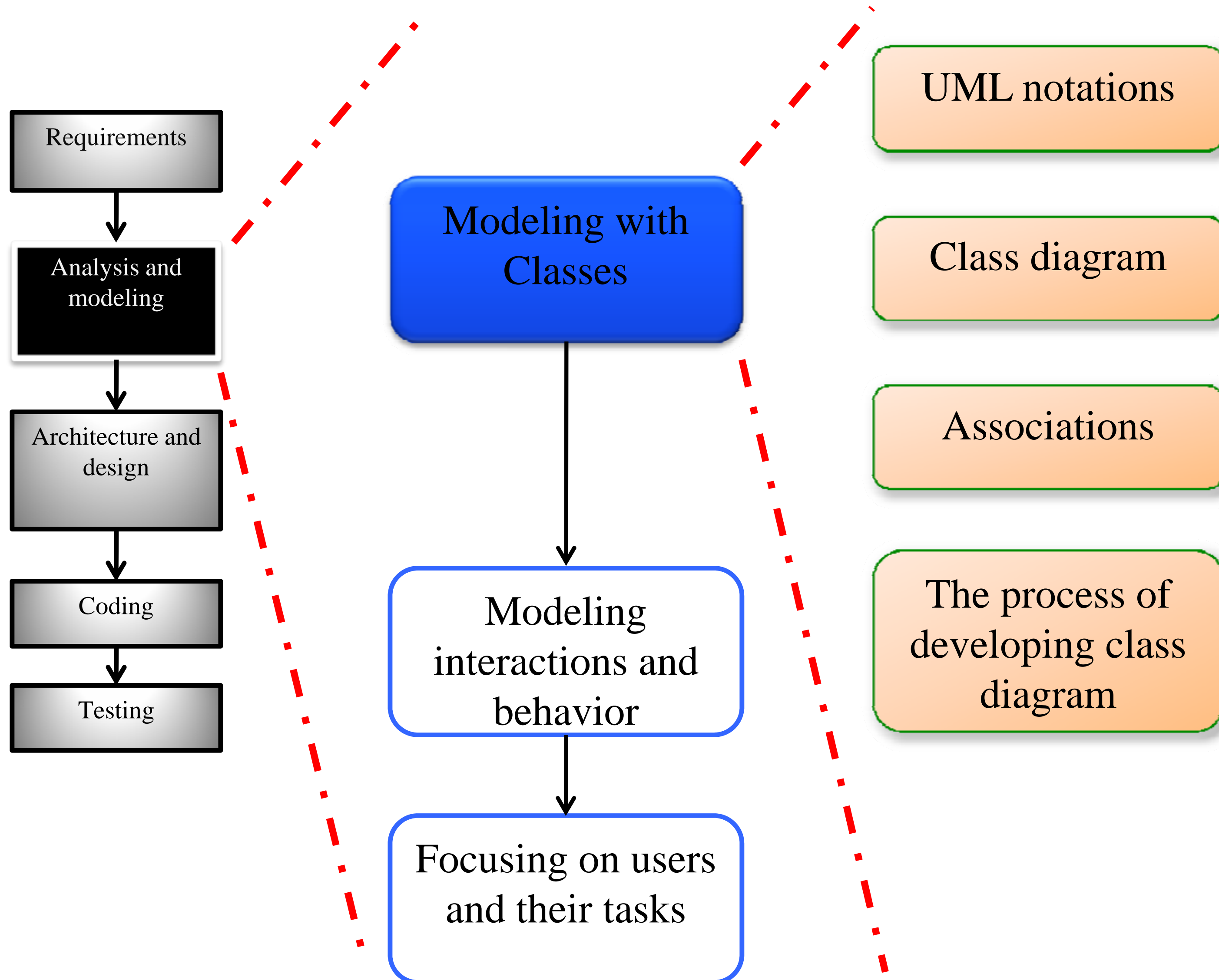
課程：嵌入式系統與軟體工程

開發學校：輔仁大學資工系

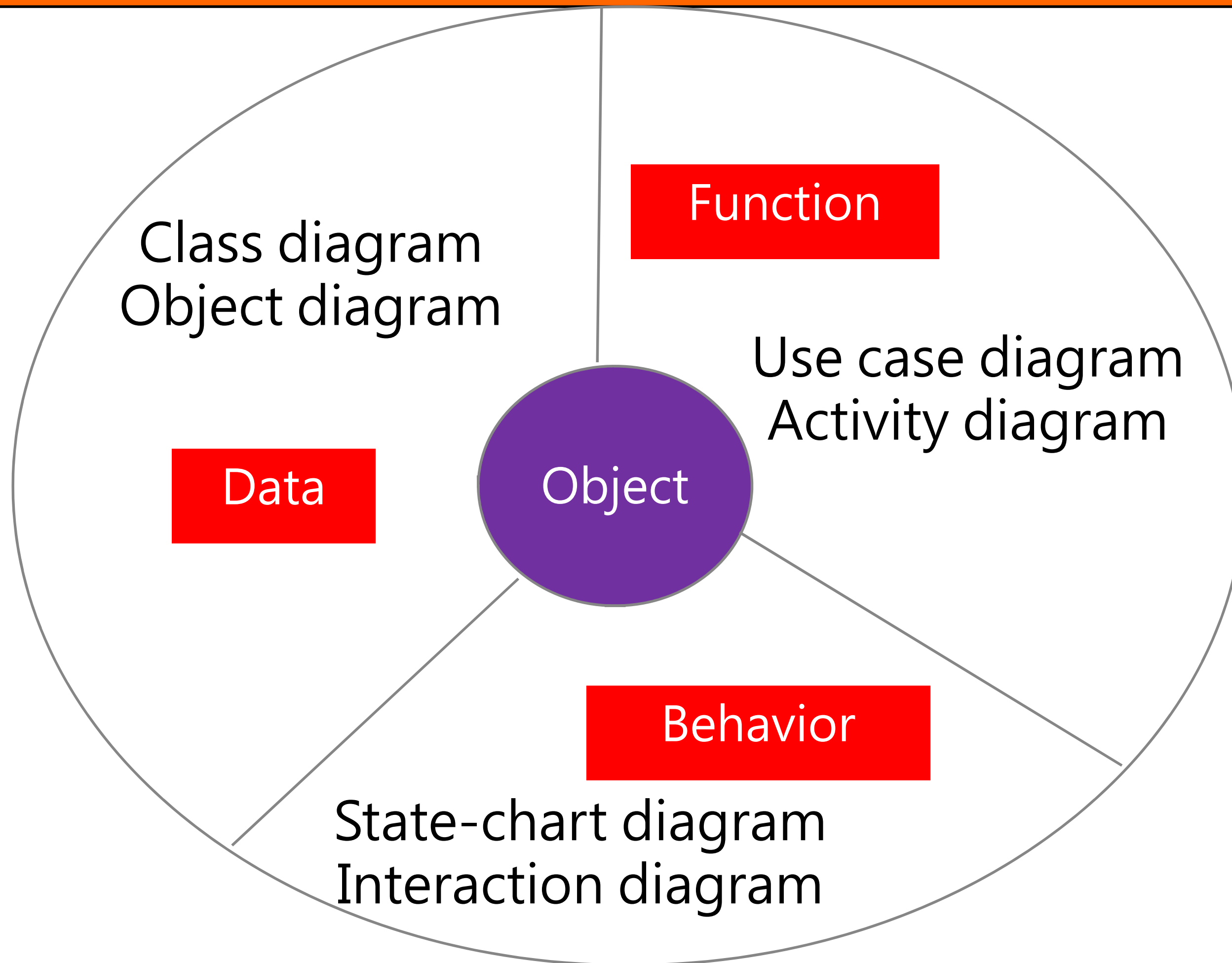


范姜永益

Process for Software Modeling



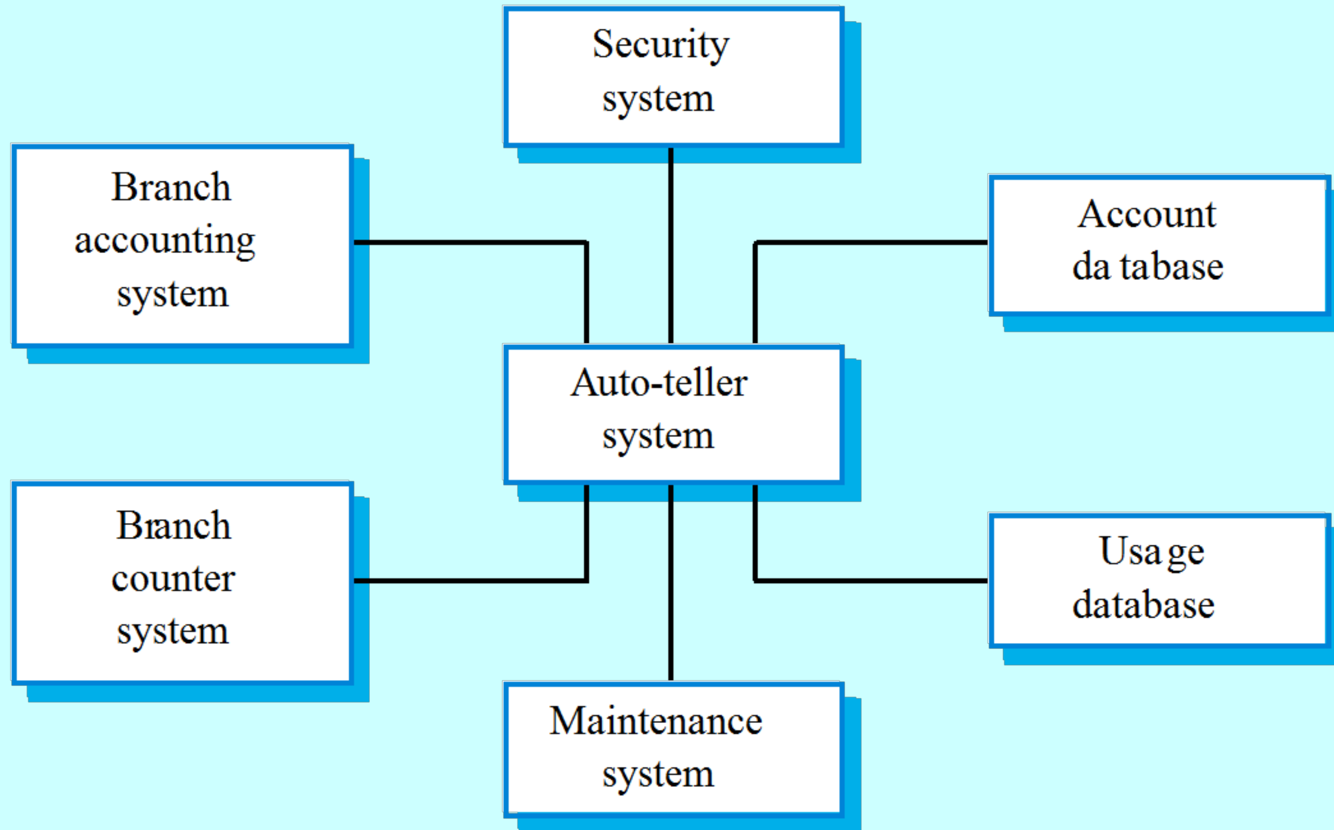
Analysis Model - UML



Context models

- **Context models** are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

The context of an ATM system

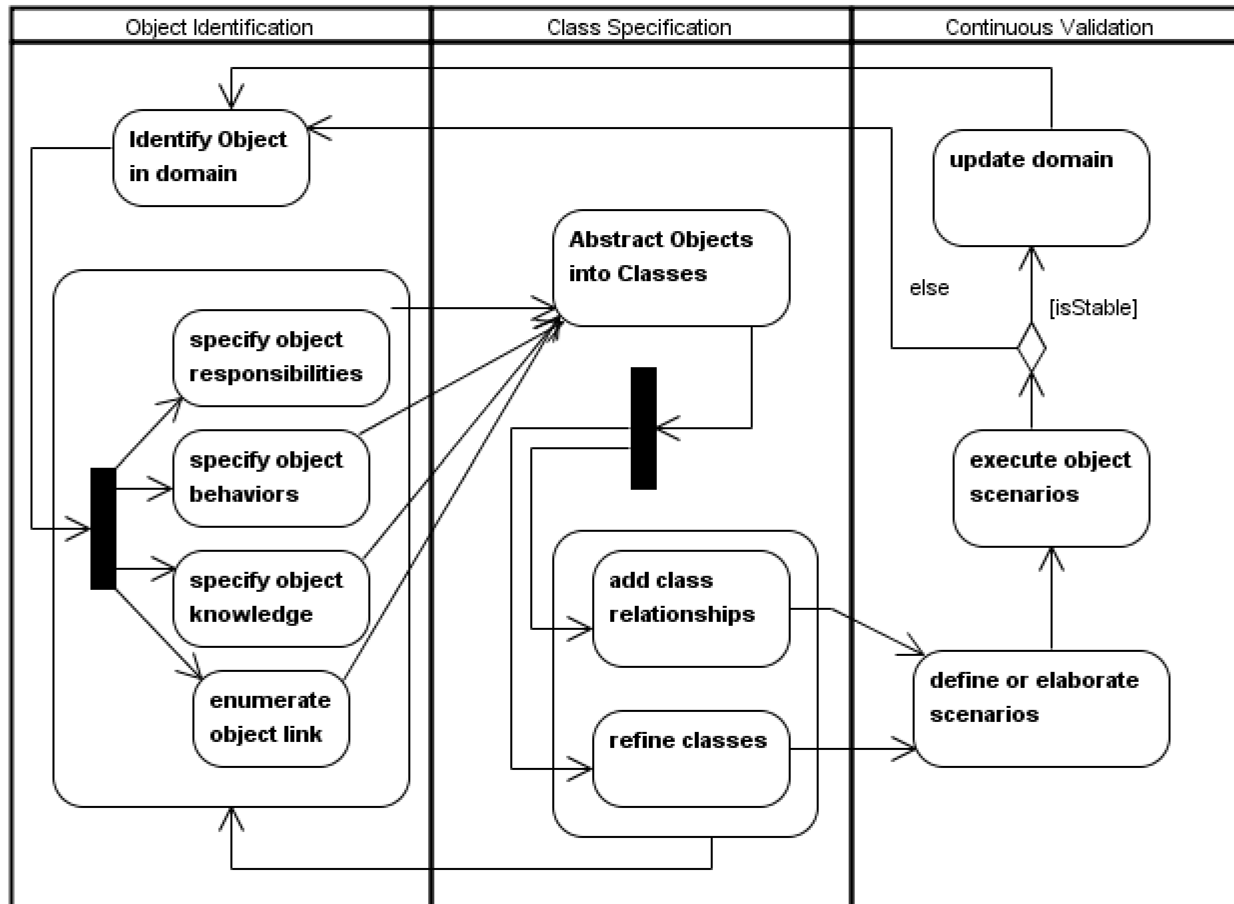


The Object/Class Discovery Process

- Object and class identification and how to infer relationships and associations among them.

- Steps
  - Object identification
  - Class specification
  - Continuous Validation

Object/Class Discovery Process



The Process of Developing Class Diagrams

You can create UML models at different stages and with different purposes and levels of details

- **Exploratory domain model**
  - Developed in domain analysis to learn about the domain
- **System domain model**
  - Models aspects of the domain represented by the system
- **System model**
  - Includes also classes used to build the user interface and system architecture

SYSTEM DOMAIN MODEL vs. SYSTEM MODEL

Type of model	*Contains elements that represent things in the domain*	*Models only things that will actually be implemented*	*Contains elements that do not represent things in the domain, but are needed to build a complete system*
Exploratory domain model: developed in domain analysis to learn about the domain	Yes	No	No
System domain model: models those aspects of the domain represented by the system	Yes	Yes	No
System model: includes classes used to build the user interface and system architecture	Yes	Yes	Yes

SYSTEM DOMAIN MODEL vs. SYSTEM MODEL

- The ***system domain model*** omits many classes that are needed to build a complete system
  - Can contain less than half the classes of the system.
  - Should be developed to be used independently of particular sets of
    - user interface classes
    - architectural classes

- The complete ***system model*** includes
  - The system domain model
  - User interface classes
  - Architectural classes
  - Utility classes

SUGGESTED SEQUENCE OF ACTIVITIES

- Identify a first set of candidate **classes**
- Add **associations** and **attributes**
- Find **generalizations**
- List the main **responsibilities** of each class
- Decide on specific **operations**
- **Iterate** over the entire process until the model is satisfactory
  - Add or delete classes, associations, attributes, generalizations, responsibilities or operations
  - Identify interfaces
  - Apply design patterns

Identifying classes

- When developing a domain model you tend to ***discover*** classes
- When you work on the user interface or the system architecture, you tend to ***invent*** classes
  - Needed to solve a particular design problem
  - (*Inventing may also occur when creating a domain model*)
- **Reuse should always be a concern**
  - Frameworks
  - System extensions
  - Similar systems

A simple technique for discovering domain classes

- Look at a source material such as a description of requirements
- Extract the ***nouns*** and ***noun phrases***
- Eliminate nouns that:
  - are redundant
  - represent instances
  - are vague or highly general
  - not needed in the application
- Pay attention to classes in a domain model that represent *types of users* or other actors

Identifying associations and attributes

- Start with classes you think are most **central** and important
- Decide on the clear and obvious data it must contain and its relationships to other classes.
- Work outwards towards the classes that are less important.
- Avoid adding many associations and attributes to a class
  - A system is simpler if it manipulates less information

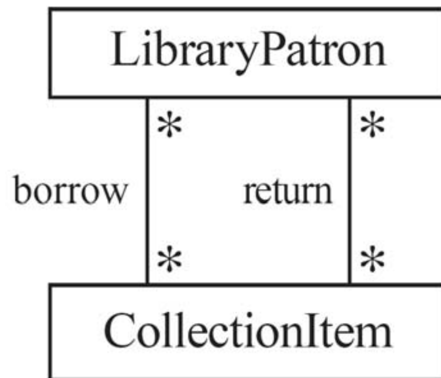
Tips about identifying and specifying valid associations

- An association should exist if a class
  - *possesses*
  - *controls*
  - *is connected to*
  - *is related to*
  - *is a part of*
  - *has as parts*
  - *is a member of*, or
  - *has as members*

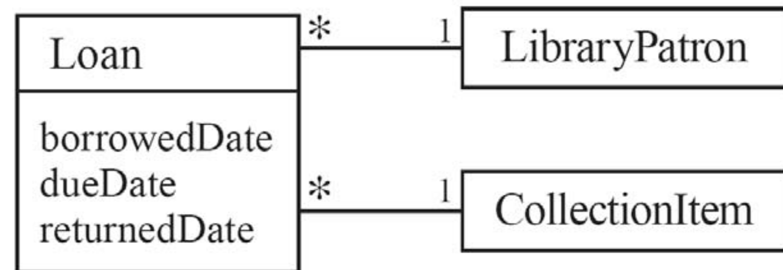
  some other class in your model
- Specify the multiplicity at both ends
- Label it clearly.

Actions versus associations

- A common mistake is to represent *actions* as if they were associations



Bad, due to the use of associations that are actions



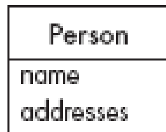
Better: The `borrow` operation creates a `Loan`, and the `return` operation sets the `returnedDate` attribute

Identifying attributes

- Look for information that must be maintained about each class
- Several nouns rejected as classes, may now become attributes
- An attribute should generally contain a simple value
  - e.g. string, number

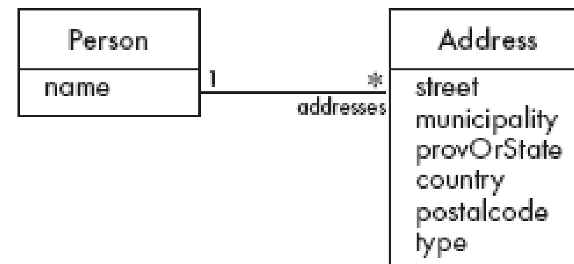
Tips about identifying and specifying valid attributes

- It is not good to have many duplicate attributes
- If a subset of a class's attributes form a coherent group, then create a distinct class containing these attributes



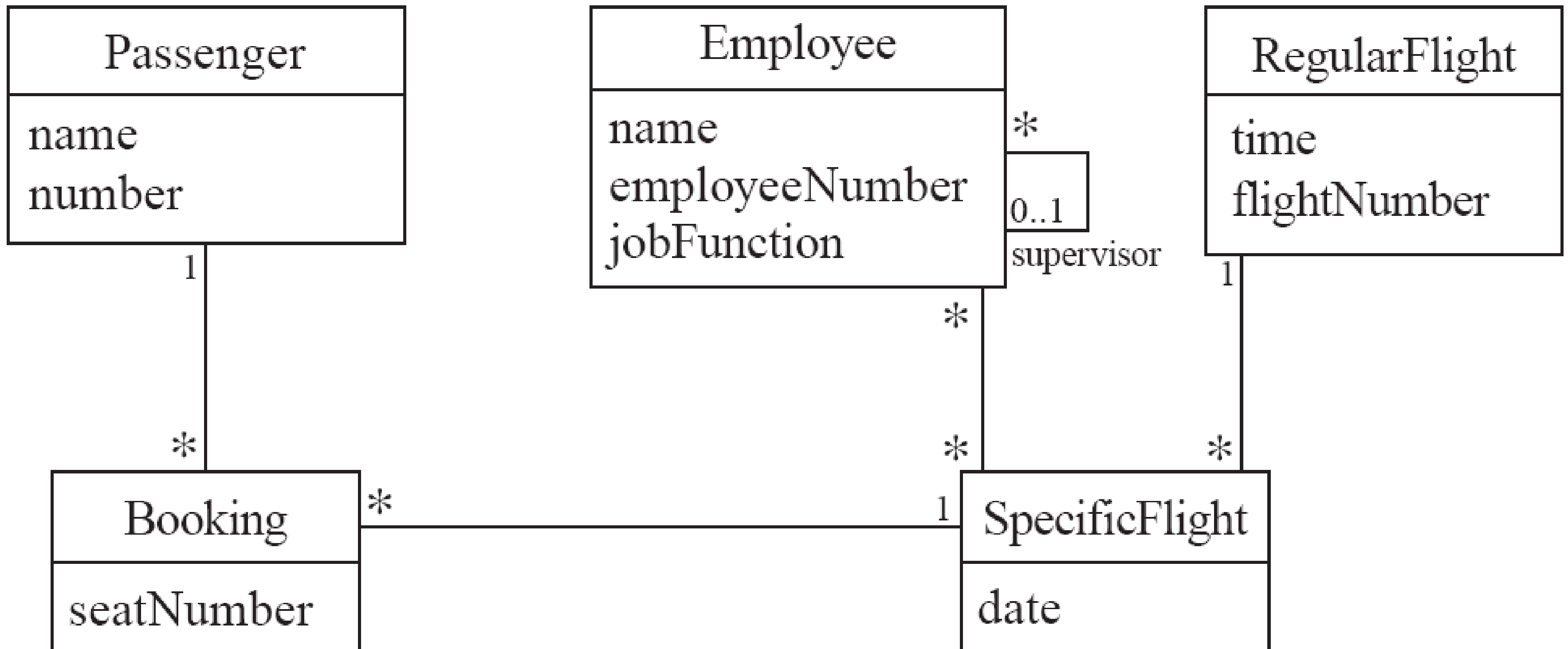
Bad, due to a plural attribute

Bad, due to too many attributes, and the inability to add more addresses



Good solution. The type indicates whether it is a home address, business address etc.

An example
(attributes and associations)

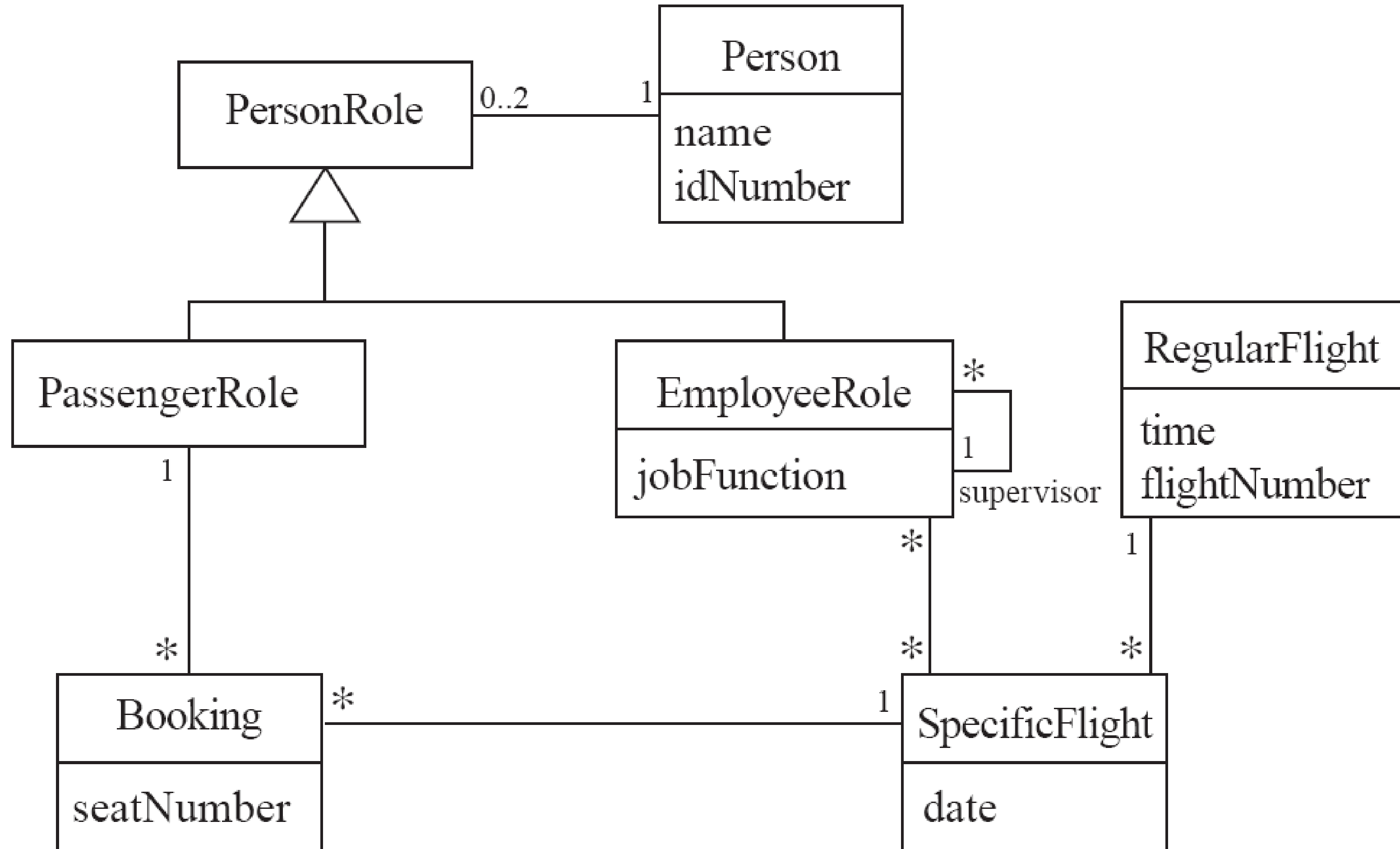


Identifying generalizations and interfaces

- There are two ways to identify generalizations:
  - bottom-up
    - Group together similar classes creating a new superclass
  - top-down
    - Look for more general classes first, specialize them if needed

- Create an ***interface***, instead of a superclass if
  - The classes are very dissimilar except for having a few operations in common
  - One or more of the classes already have their own superclasses
  - Different implementations of the same class might be available

An example (generalization)



Allocating responsibilities to classes

A *responsibility* is something that the system is required to do.

- Each functional requirement must be attributed to one of the classes
  - All the responsibilities of a given class should be *clearly related*.
  - If a class has too many responsibilities, consider *splitting* it into distinct classes
  - **If a class has no responsibilities attached to it, then it is probably *useless***
  - When a responsibility cannot be attributed to any of the existing classes, then a *new class* should be created

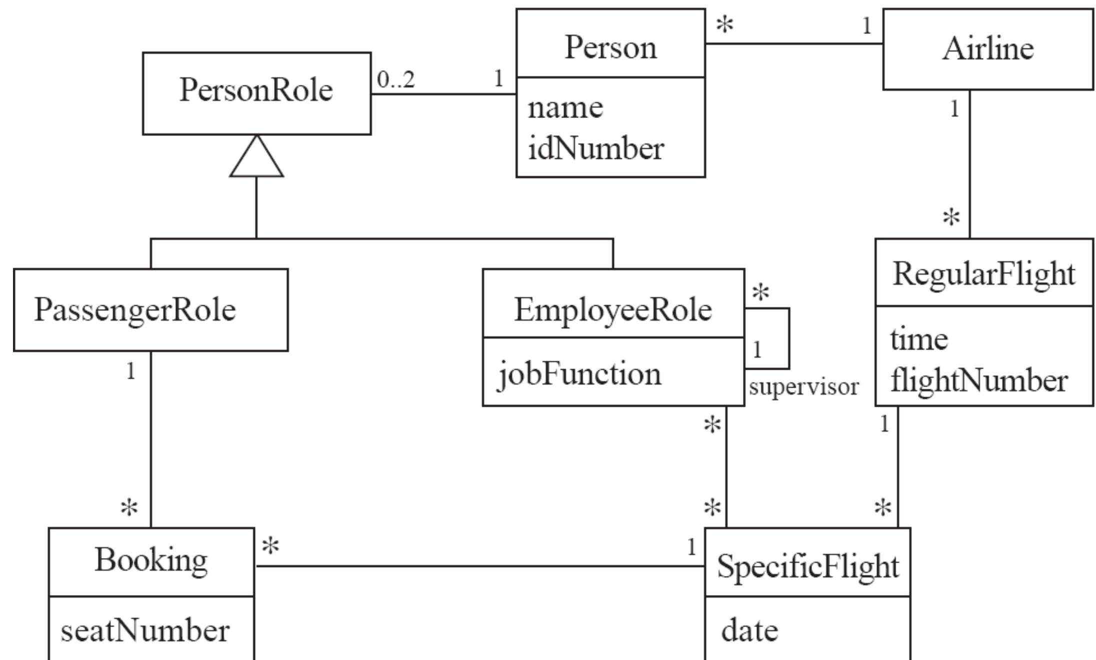
- To determine responsibilities
  - Perform use case analysis
  - Look for verbs and nouns describing *actions* in the system description

Categories of responsibilities

- Setting and getting the values of attributes
- Creating and initializing new instances
- Loading to and saving from persistent storage
- Destroying instances
- Adding and deleting links of associations
- Copying, converting, transforming, transmitting or outputting
- Computing numerical results
- Navigating and searching
- Other specialized work

An example (responsibilities)

- Creating a new regular flight
- Searching for a flight
- Modifying attributes of a flight
- Creating a specific flight
- Booking a passenger
- Canceling a booking



Prototyping a class diagram on paper

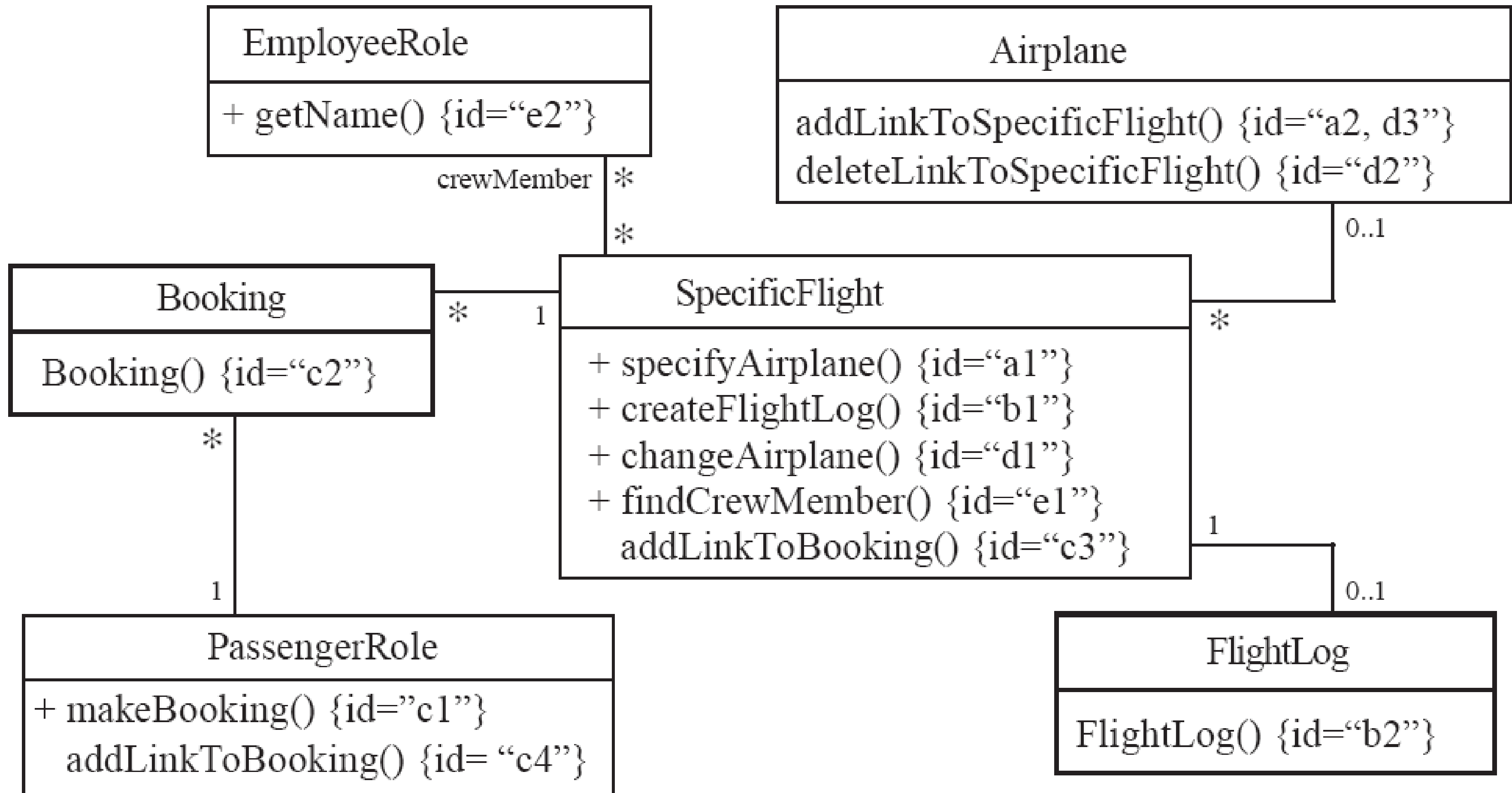
- As you identify classes, you write their names on small cards
- As you identify attributes and responsibilities, you list them on the cards
  - If you cannot fit all the responsibilities on one card:
    - this suggests you should split the class into two related classes.
- Move the cards around on a whiteboard to arrange them into a class diagram.
- Draw lines among the cards to represent associations and generalizations.

Identifying operations

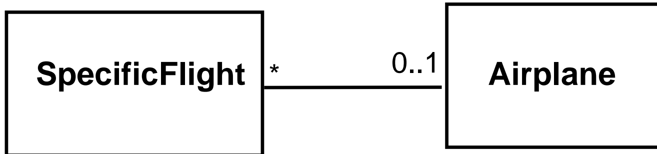
Operations are needed to realize the responsibilities of each class

- There may be several operations per responsibility
- The main operations that implement a responsibility are normally declared **public**
- Other methods that collaborate to perform the responsibility must be as private as possible

AN EXAMPLE (CLASS COLLABORATION)



Class collaboration 'a'

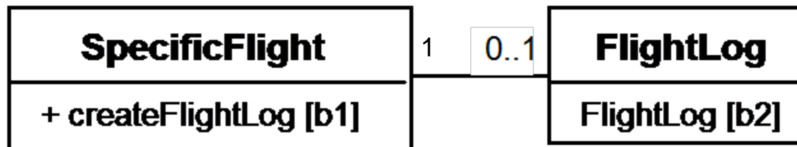


Making a *bi-directional link* *between two existing objects*;

e.g. adding a link between an instance of SpecificFlight and an instance of Airplane.

1. (public) The instance of SpecificFlight
   - **makes a one-directional link to the instance of** Airplane
   - **then calls operation 2.**
2. (non-public) The instance of Airplane
   - **makes a one-directional link back to the instance of** SpecificFlight

Class collaboration ‘b’

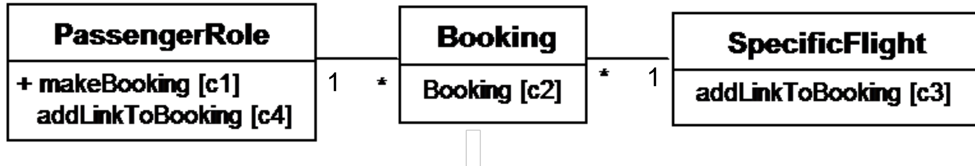


Creating an object and linking it to an existing object

e.g. creating a FlightLog, and linking it to a SpecificFlight.

1. (public) The instance of SpecificFlight
   - **calls the constructor of** FlightLog **(operation 2)**
   - **then makes a one-directional link to the new instance of** FlightLog**.**
2. (non-public) Class FlightLog’s constructor
   - **makes a one-directional link back to the instance of** SpecificFlight**.**

Class collaboration ‘c’

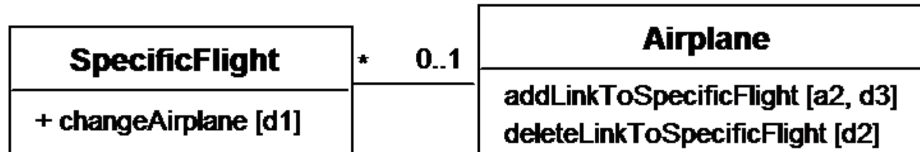


Creating an association class, given two existing objects

e.g. creating an instance of Booking, which will link a SpecificFlight to a PassengerRole.

1. (public) The instance of PassengerRole
   - calls the constructor of **Booking** (operation 2).
2. (non-public) Class Booking’s constructor, among its other actions
   - makes a one-directional link back to the instance of **PassengerRole**
   - makes a one-directional link to the instance of **SpecificFlight**
   - calls operations 3 and 4.
3. (non-public) The instance of SpecificFlight
   - makes a one-directional link to the instance of **Booking**.
4. (non-public) The instance of PassengerRole
   - makes a one-directional link to the instance of `Booking`.

Class collaboration ‘d’



Changing the destination of a link

e.g. changing the Airplane of to a SpecificFlight, from airplane1 to airplane2

1. (public) The instance of SpecificFlight
   - deletes the link to **airplane1**
   - makes a one-directional link to **airplane2**
   - calls operation 2
   - then calls operation 3.
2. (non-public) airplane1
   - deletes its one-directional link to the instance of **SpecificFlight**.
3. (non-public) airplane2
   - makes a one-directional link to the instance of **SpecificFlight**.

Class collaboration 'e'

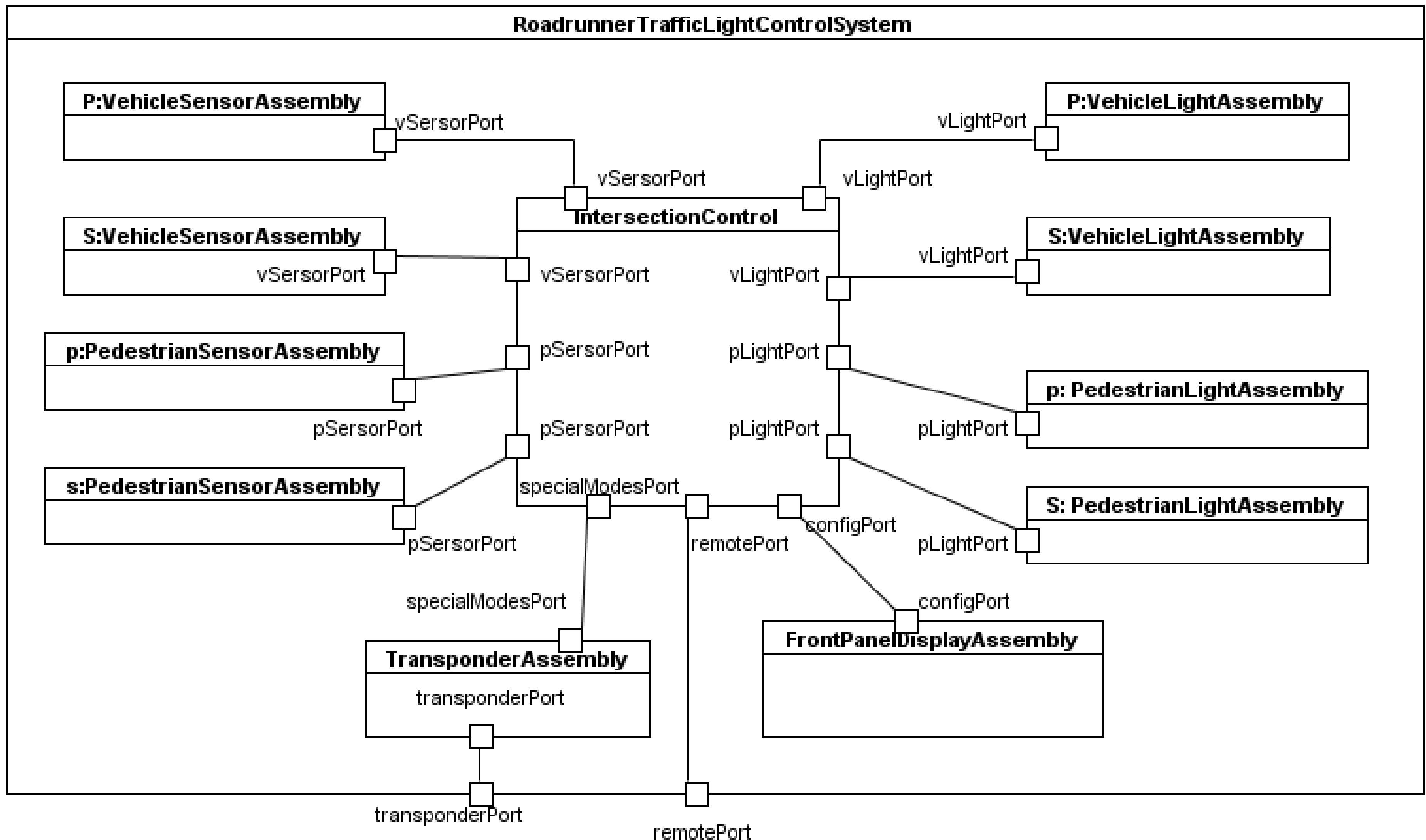
Searching for an associated instance

e.g. searching for a crew member associated with a SpecificFlight that has a certain name.

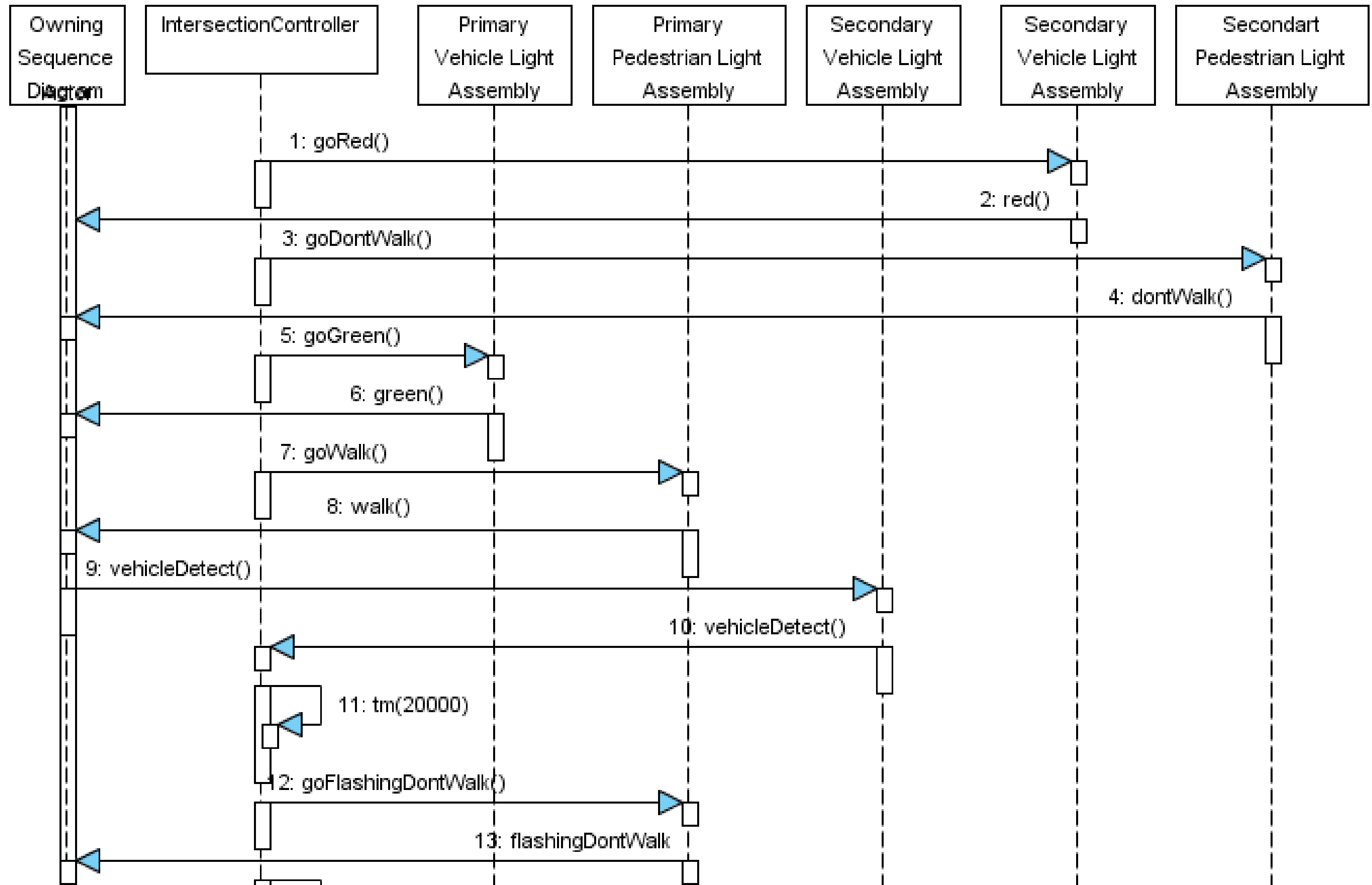
1. (public) The instance of SpecificFlight
   - creates an Iterator over all the crewMember links of the SpecificFlight
   - for each of them call operation 2, until it finds a match.
2. (may be public) The instance of EmployeeRole returns its name.

Example

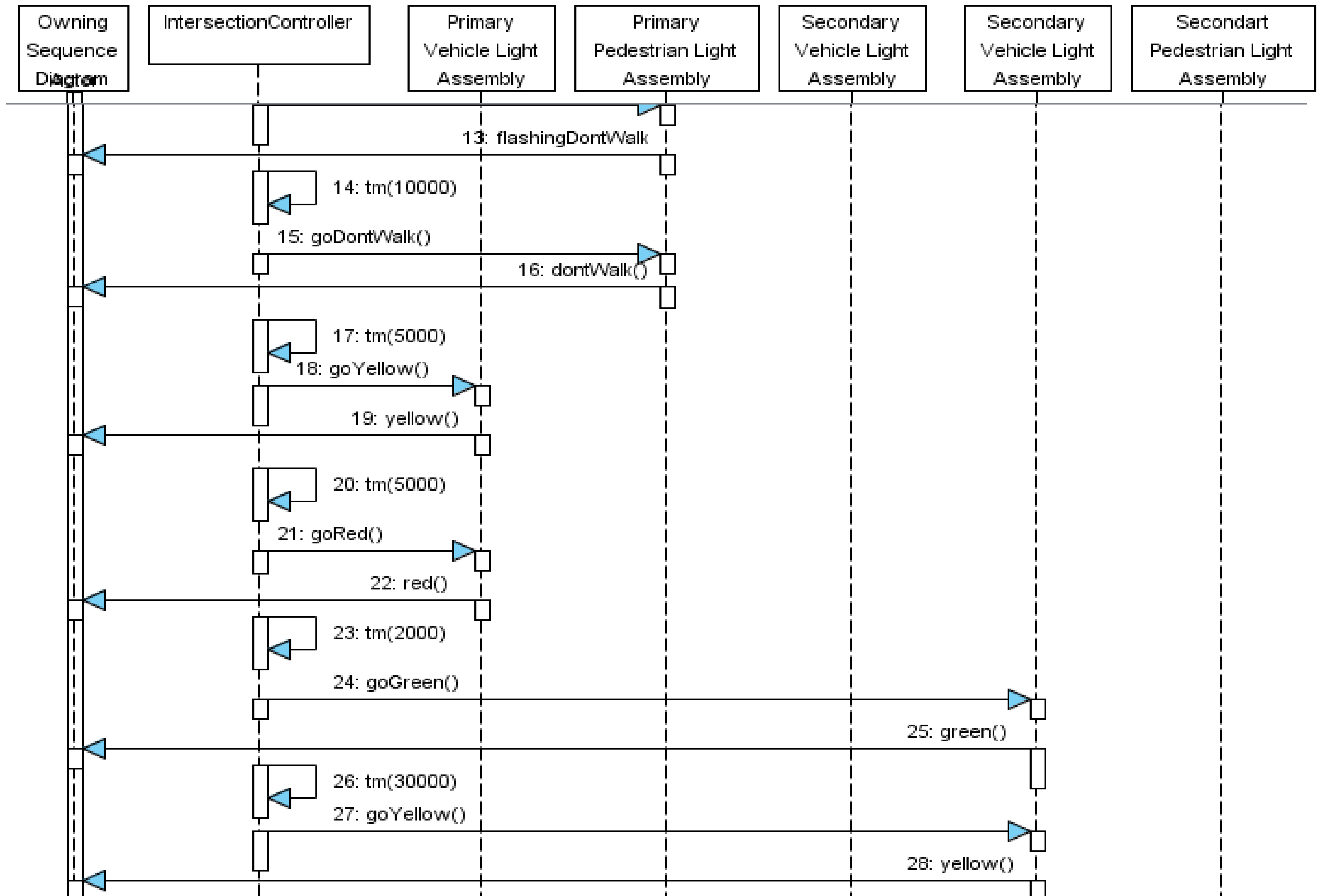
Roadrunner System Context Diagram



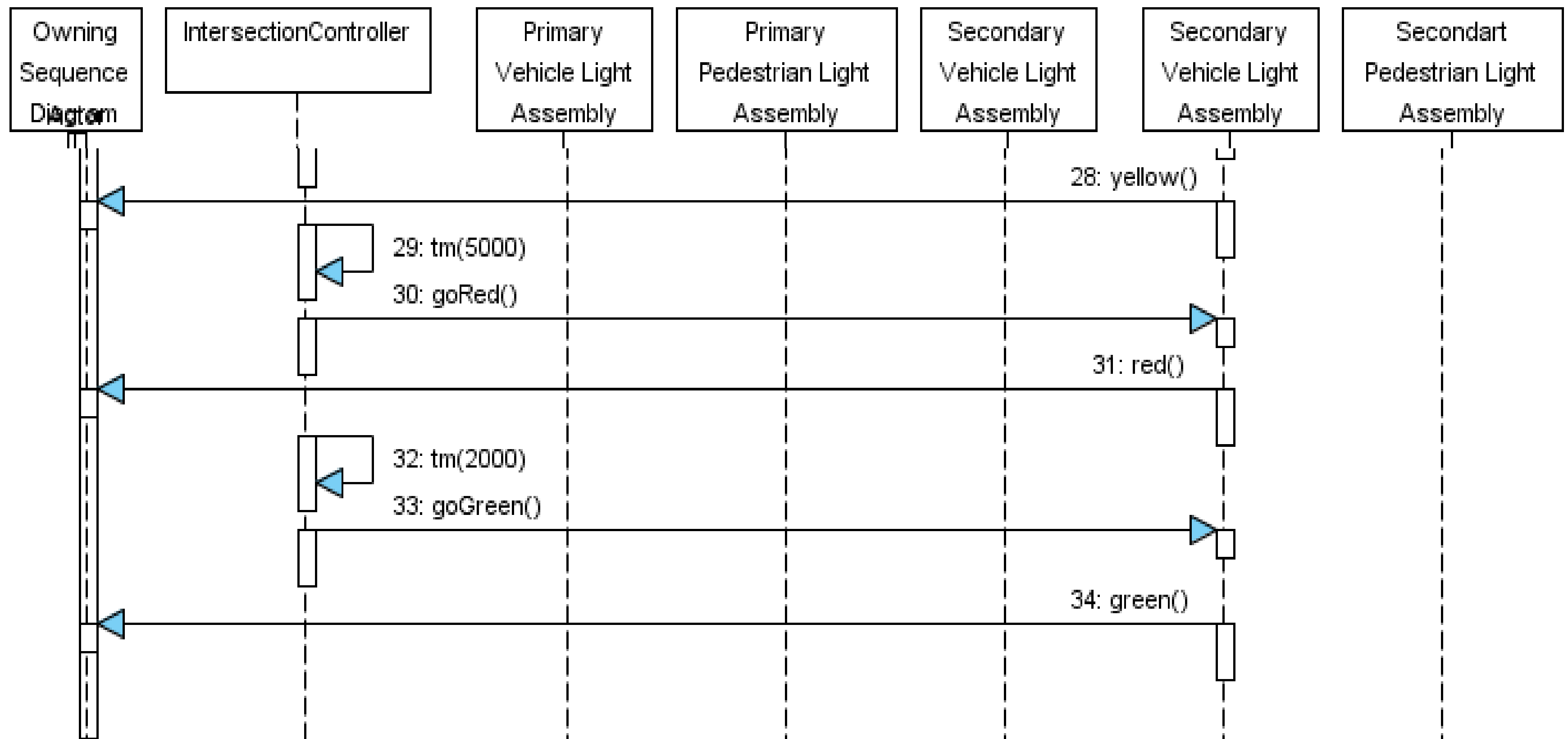
Sequence Diagram



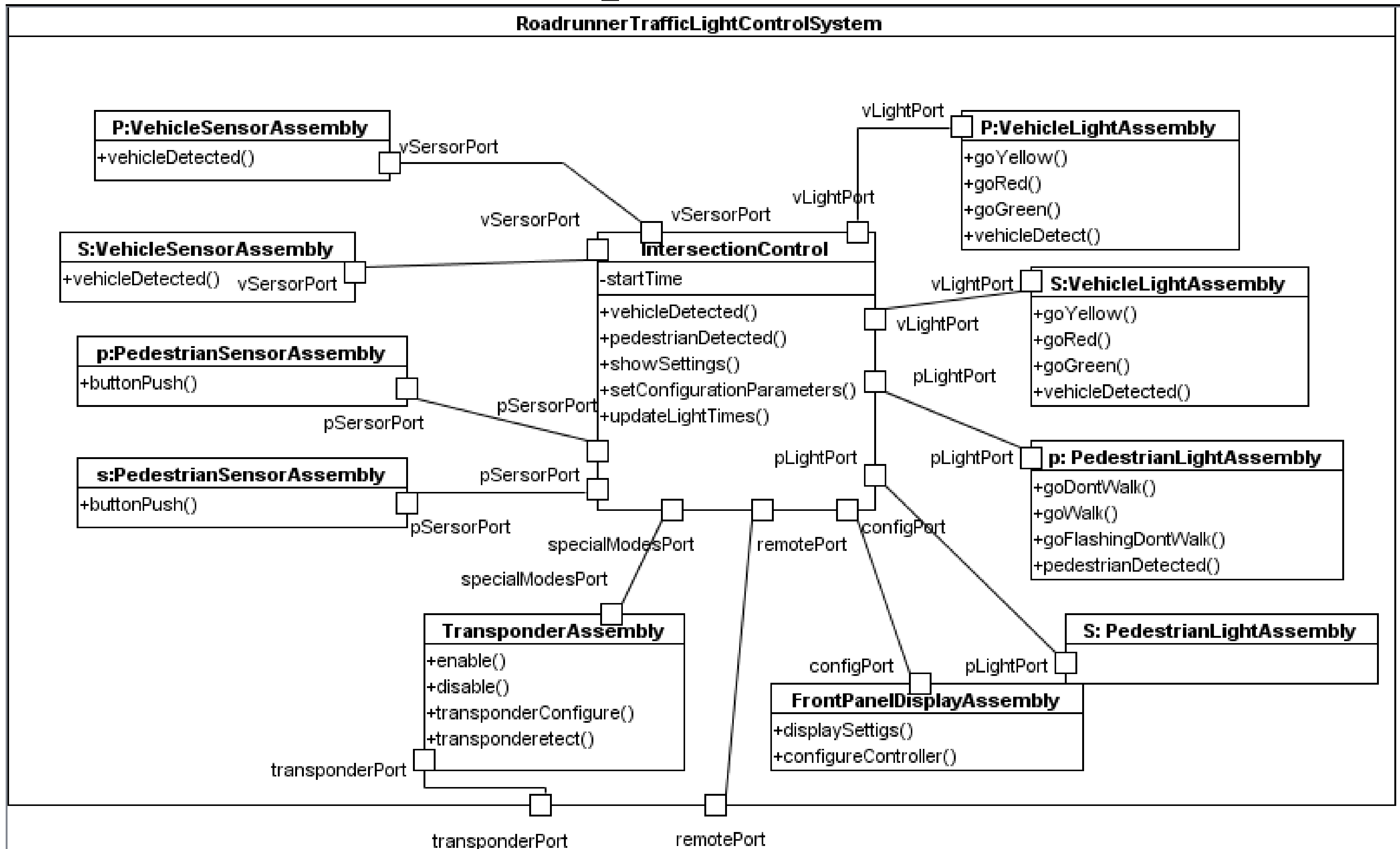
Sequence Diagram



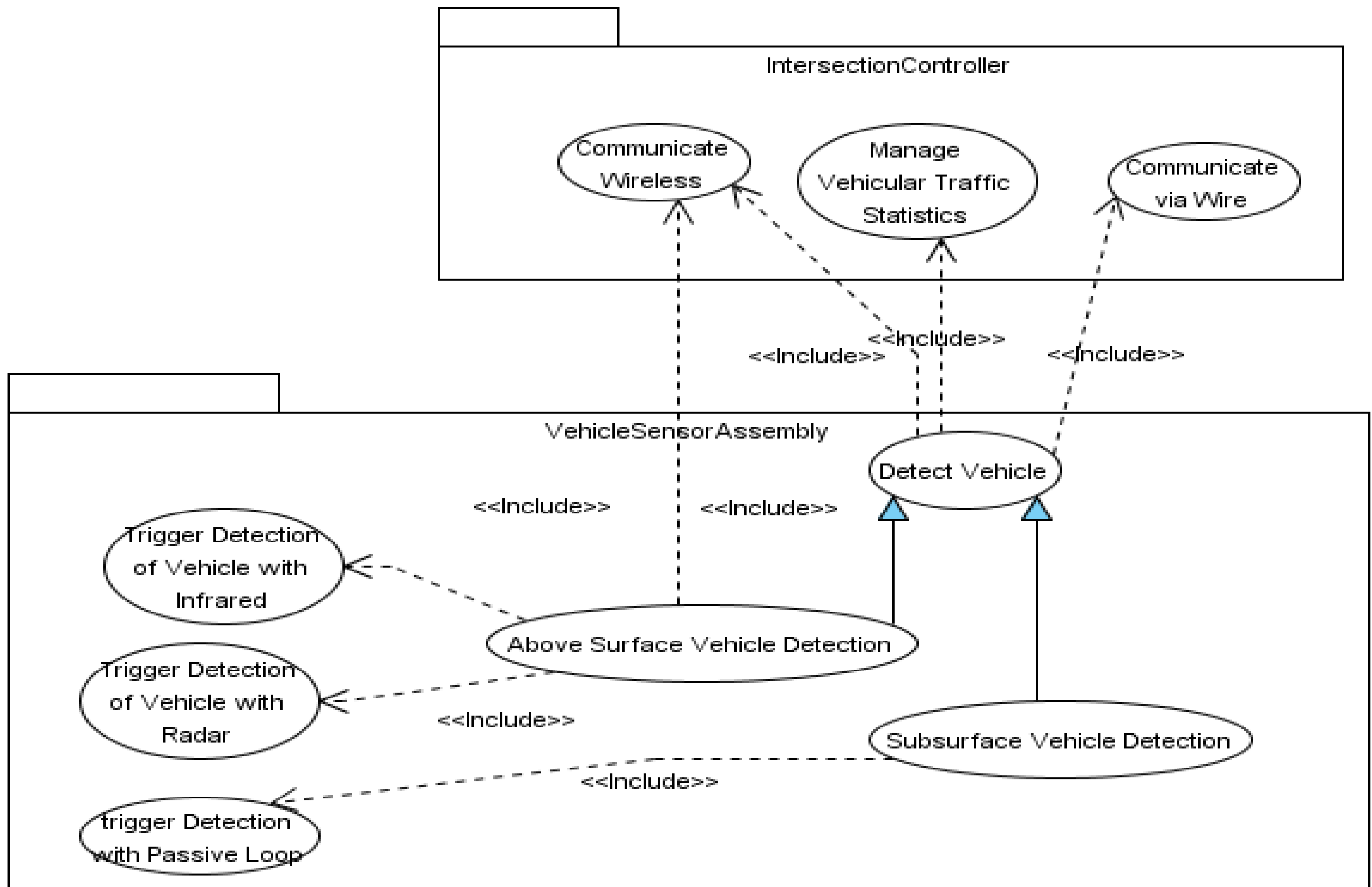
Sequence Diagram



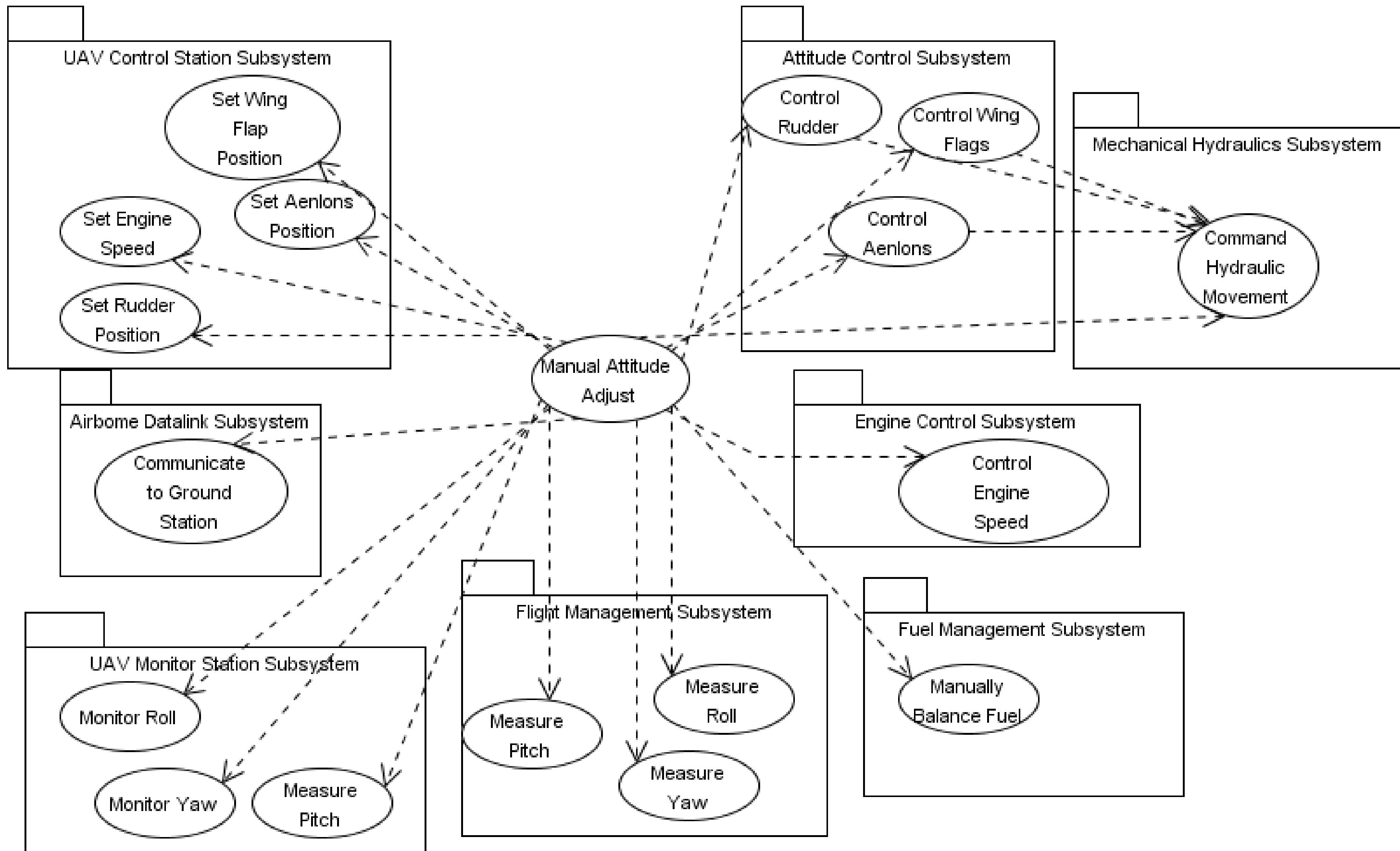
Roadrunner Context Diagram with Operations



Use Cases maps to Subsystems



CUAV Manual Adjust Use Cases mapped to subsystem



Object Analysis

The Vehicle Detector

- Three **types of Vehicular Detectors** shall be supported: **subsurface passive loop inductors (SPLIS)**, **above-surface infrared sensors(ASIs)** and **above-surface radars(ASRs)**.
- **Subsurface detectors** shall use a **wired interface** to communicate with the **controller**, while **ASIs** and **ASRs** shall support both **wired and secure wireless communication**. All **vehicle detectors** shall be able to perform **vehicle counting**.
- In addition, ASIs and ASRs shall be able to receive **directional transmissions** from **priority vehicle and emergency vehicle transmitters**. The **maximum range** of such **reception** shall be no less than **250 feet** and no more that **1000 feet**.
- **Figure 10.1** shows the **relevant measures** for both **ASI and ASR detectors**. When a **vehicle** enters the **detection area** (shown as the **shaded area** in the figure), the **detectors** shall report the **presence of a vehicle**. **Separated detectors** are used for **each lane** in **each direction**.

Infrared and Radar Vehicle Detector

Roadrunner Detect Vehicle Nouns

Noun Phrase	Element Type	Element Name
Type of vehicle detectors	class	VehicleDector
Subsurface passive loop inductors	class	PassiveLoopInductor
SPLI	class	PassiveLoopInductor
above-surface infrared sensors	class	InfraredVehicleDetector
ASIs	class	InfraredVehicleDetector
above-surface radars	class	RadarDetector
ASRs	class	RadarDetector
Subsurface detectors	Class	PassiveLoopInductor
wired interface	Class	WiredNetworkInterface
wired and secure wireless communication	Unknown	Unknown
vehicle counting	Attribute	VehicleCount.count
priority vehicle and emergency vehicle transmitters	actor	

Class Diagram