


ESW聯盟「嵌入式系統與軟體工程」

Requirements Analysis to Embedded Software

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



Outline

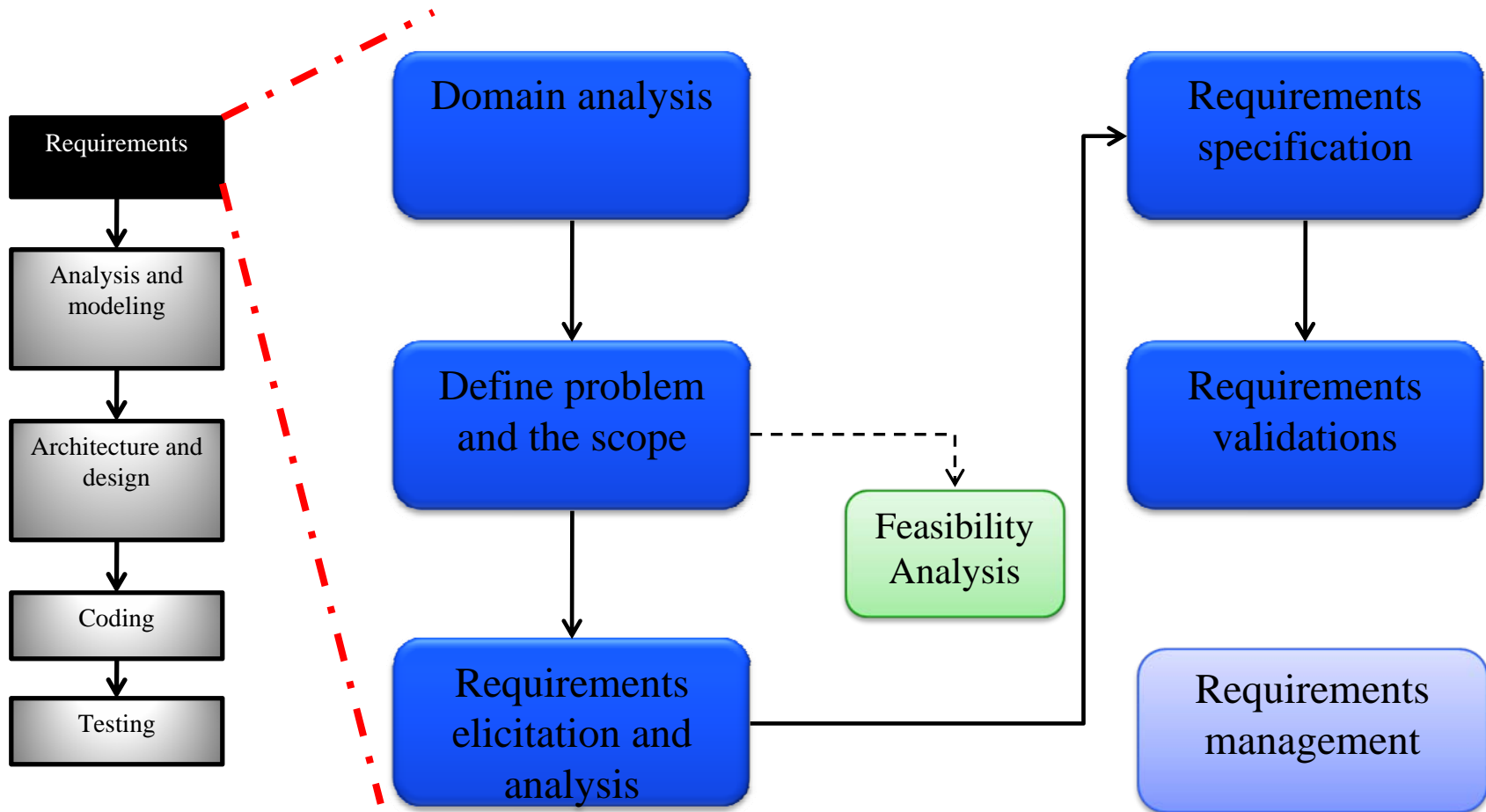
- Requirements Analysis of Embedded Systems
- Object Domain Analysis
- Defining Object Behavior

Requirements Analysis of Embedded Systems

Outline

- Identify types of requirements
 - Group requirements together
 - Identify use cases
 - Capture and represent quality of service requirements
 - Identify operational scenarios
 - Use state machines for representing requirements
 - Capture complex requirements
 - Capture algorithm requirements with activity diagrams
 - Validate requirements models
-

Process for Developing Requirements



Requirements Analysis

- In UML, use case model captures the interactions between real-time system and their external environment.
- A use case is a system capability that is detailed with accompanying text, examples (scenarios) or state models.
- The use case model decomposes the primary functionality of the system and protocols necessary to meet these functional requirements.

What is a Requirement ?

It is a statement describing either

- 1) an aspect of what the proposed system must do, or
 - 2) a constraint (quality) on the system's development.
-
- In either case it must contribute in some way towards adequately solving the customer's problem;
 - the set of requirements as a whole represents a negotiated agreement among the stakeholders.

A collection of requirements is a *requirements document*.

Requirements

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so **may be part of a contract** between client and contractor.

Software specification

- A detailed software description which can **serve as a basis for a design or implementation**. Written for developers
-

Types of Requirements

Functional requirements

- Describe *what* the system should do

Quality requirements

- *Constraints on the design* to meet specified levels of quality

Platform requirements

- *Constraints on the environment and technology* of the system

Process requirements

- *Constraints on the project plan and development methods*

Domain requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain

Non-functional requirements

NON-FUNCTIONAL REQUIREMENTS (QUALITY REQUIREMENTS)

All must be **verifiable** (testable)

Three main types

1. Categories reflecting: **usability, efficiency, reliability, maintainability** and **reusability**
 - Response time
 - Throughput
 - Resource usage
 - Reliability
 - Availability
 - Recovery from failure
 - Allowances for maintainability and enhancement
 - Allowances for reusability
-

Some Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Non-functional requirements

2. Categories **constraining the *environment and technology*** of the system.
 - Platform
 - Technology to be used (C, Java, or database etc.)

3. Categories **constraining the *project plan and development methods***
 - Development process (methodology) to be used
 - Cost and delivery date
 - Often put in **contract** or **project plan (PEP)** instead

EXERCISES

- Classify the following requirements statements into **F** for functional, **Q** for quality, **PL** for platform, **PR** for process, and **X** for “should not be a requirement”. Justify your answer.
- The system must use 128-bit encryption for all transactions.

PL - technology

- If the alarm system is ringing, then the elevators (lifts) will proceed to the ground floor, open their doors and suspend further operation. **F**
- The student information system will provide output from all commands within one second.

Q – response time

EXERCISES

PL – computing platform

- The system will be able to print to an LC-9 plotter
- The system will use an array to hold the invoices **X**
- The system can read images of the following formats: JPEG, GIF, BMP. **F**

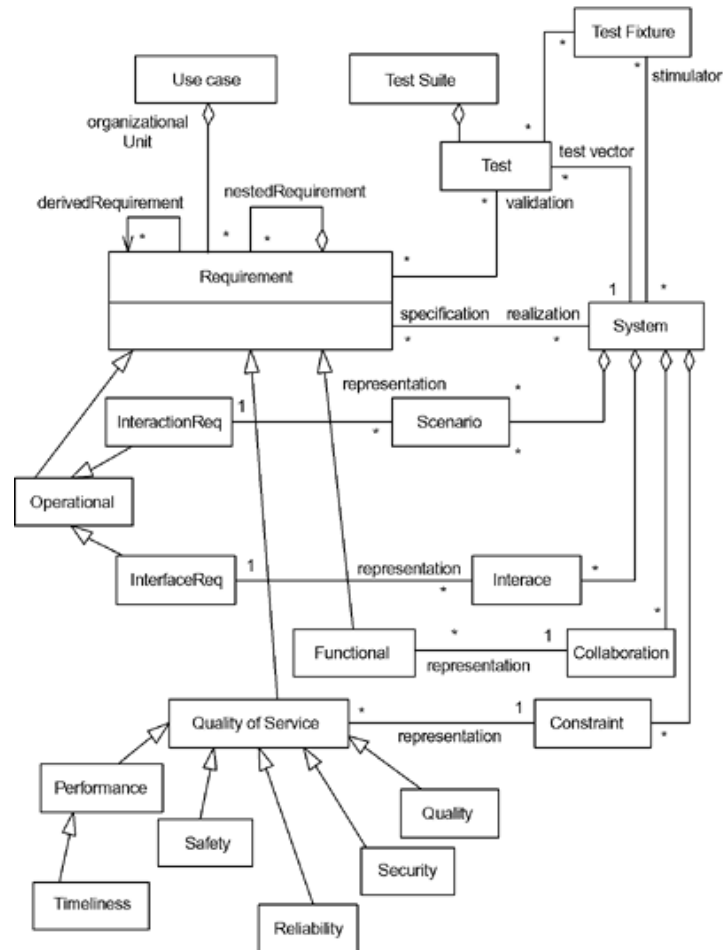
PL – computing platform

- The system must no more than 32MB of RAM
- The java.util.Date class should be used to handle dates. **X**
- The Junit framework should be used to test the system.

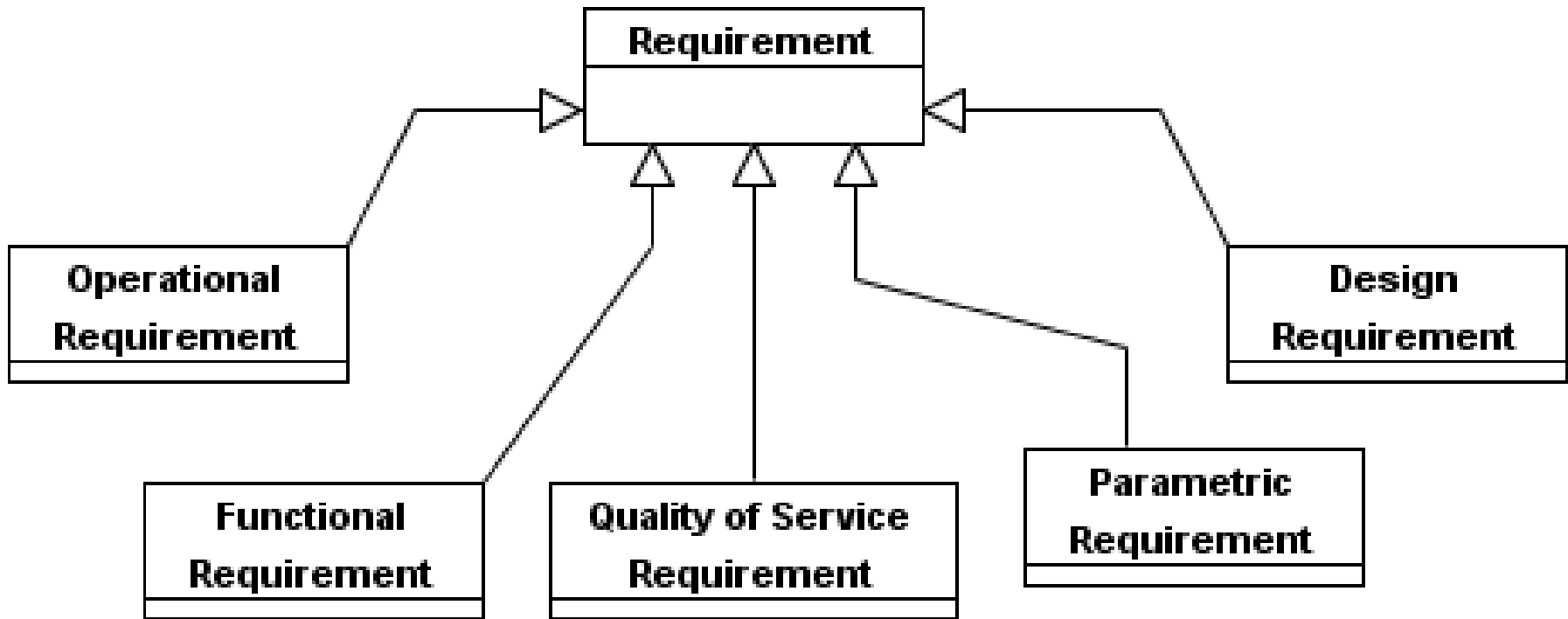
PR – development process

- The system must run under both Linux and Windows operating systems. **PL – computing platform**
-

Requirements Taxonomy



Requirements Types



The Roadrunner Intersection Controller System

- The Roadrunner Intersection Controller (RIC) is an automobile intersection controller that controls individual traffic lights, subsurface and above-surface vehicle detectors and pedestrian button signal sources for a single intersection.
- The RIC can be programmed from a front panel display (FPD) or a remote traffic manager via a wired Ethernet interface.

Intersection

Intersection Modes

- Mode 0: Safe mode
 - All vehicle lights outputs are set to FLASHING RED and pedestrian lights are disabled.
 - Mode 1: Evening Low Volume Mode
 - The designed primary road is set to FLASHING YELLOW, secondary road is set to FLASHING RED, and pedestrian lights are set to off.
 - Mode 2: Fixed Cycle Time
 - The lanes cycle GREEN-YELLOW-RED in opposite sequences with fixed intervals.
-

Intersection Modes

- Mode 3: Responsive Cycle Mode
 - Provides a fixed cycle time when secondary road is triggered by either pedestrian or vehicle.

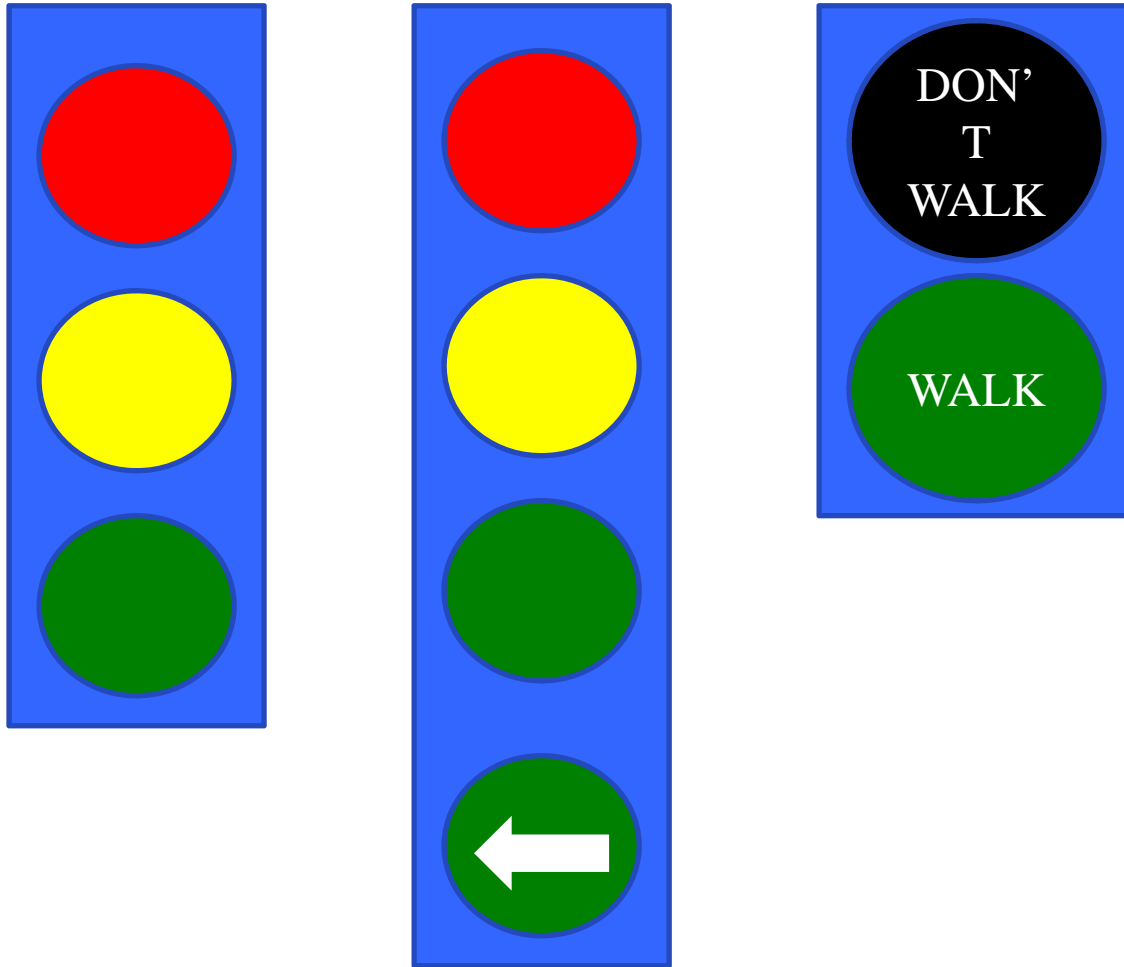
- Mode 4: Adaptive Mode
 - This mode is for intersections with higher density traffic.
 - In this mode, the intersection adapts to the local history of traffic by adjusting the cycle times depending on traffic density.

The Vehicle Detector

- Three types of vehicular detectors
 - Subsurface passive loop inductors (SPLIs)
 - Above-surface infrared sensors (ASIs)
 - Above-surface radars (ASRs)

Infrared and Radar Vehicle Detector

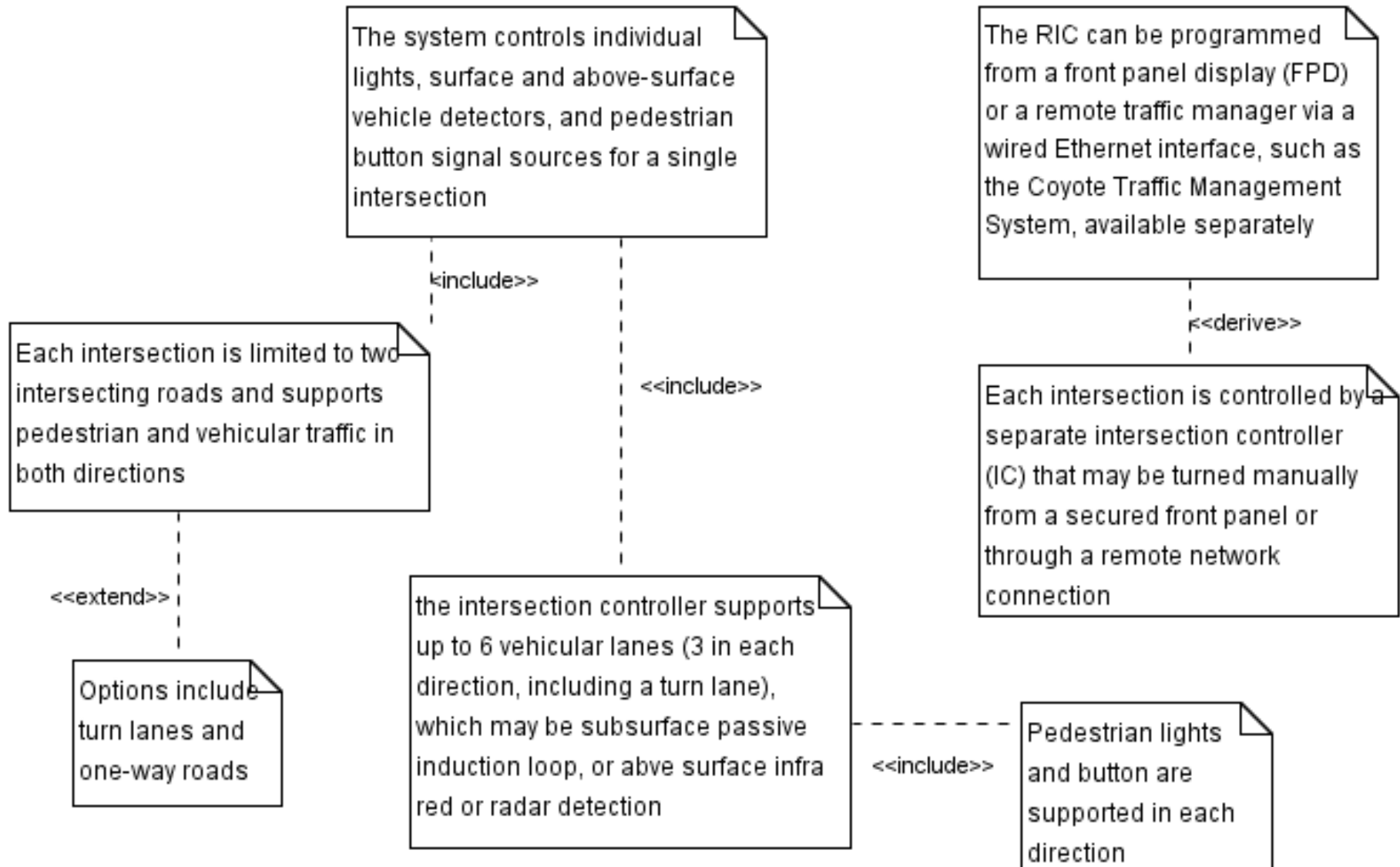
Pedestrian Light and Sensor



Case Study: Specifying Requirements

- Identify types of requirements
- Group requirements together
- Identify use cases
- Capture and represent quality of service requirements
- Use state machines for representing requirements

Identifying Kinds of Requirements

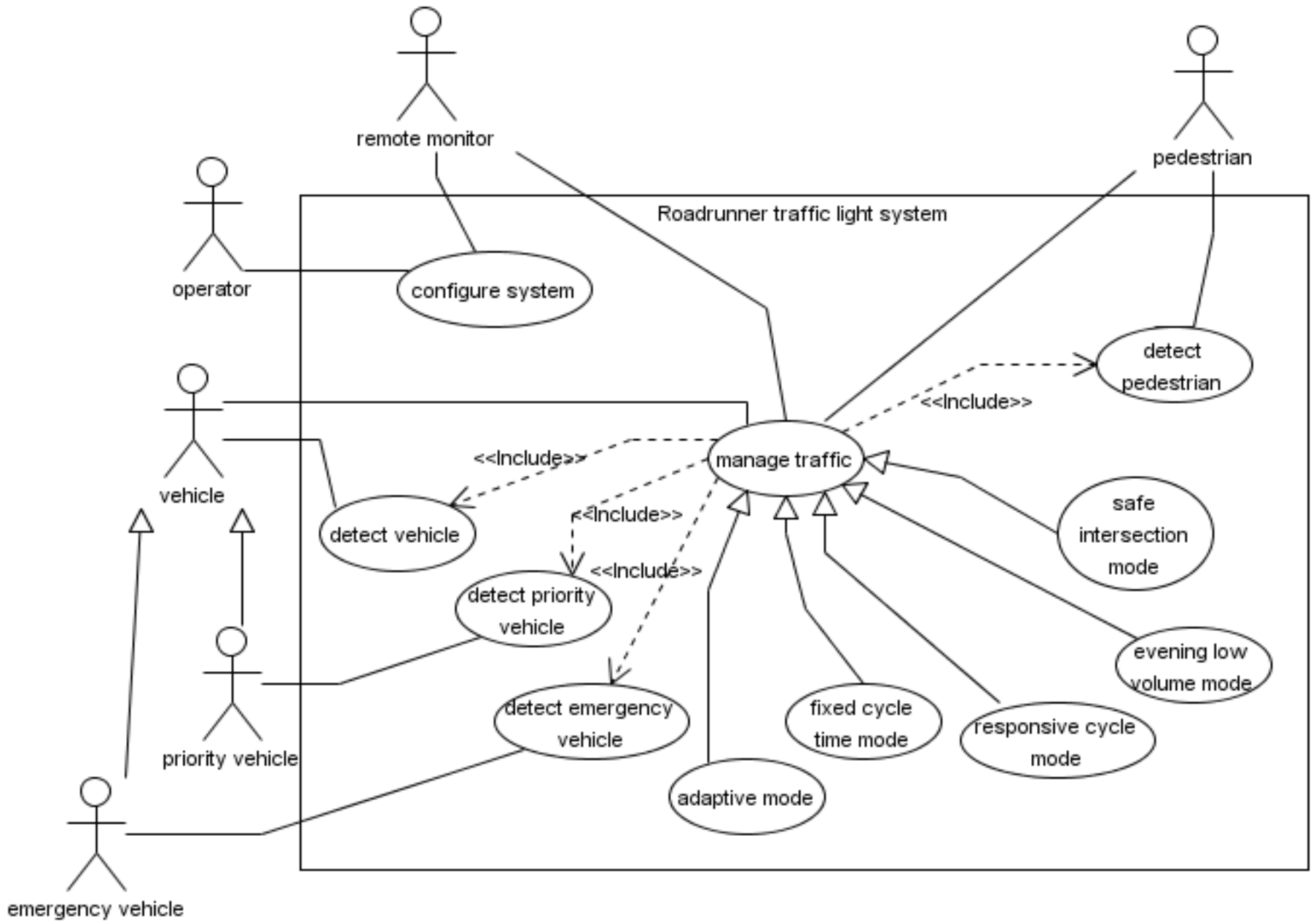


Case Study: Specifying Requirements

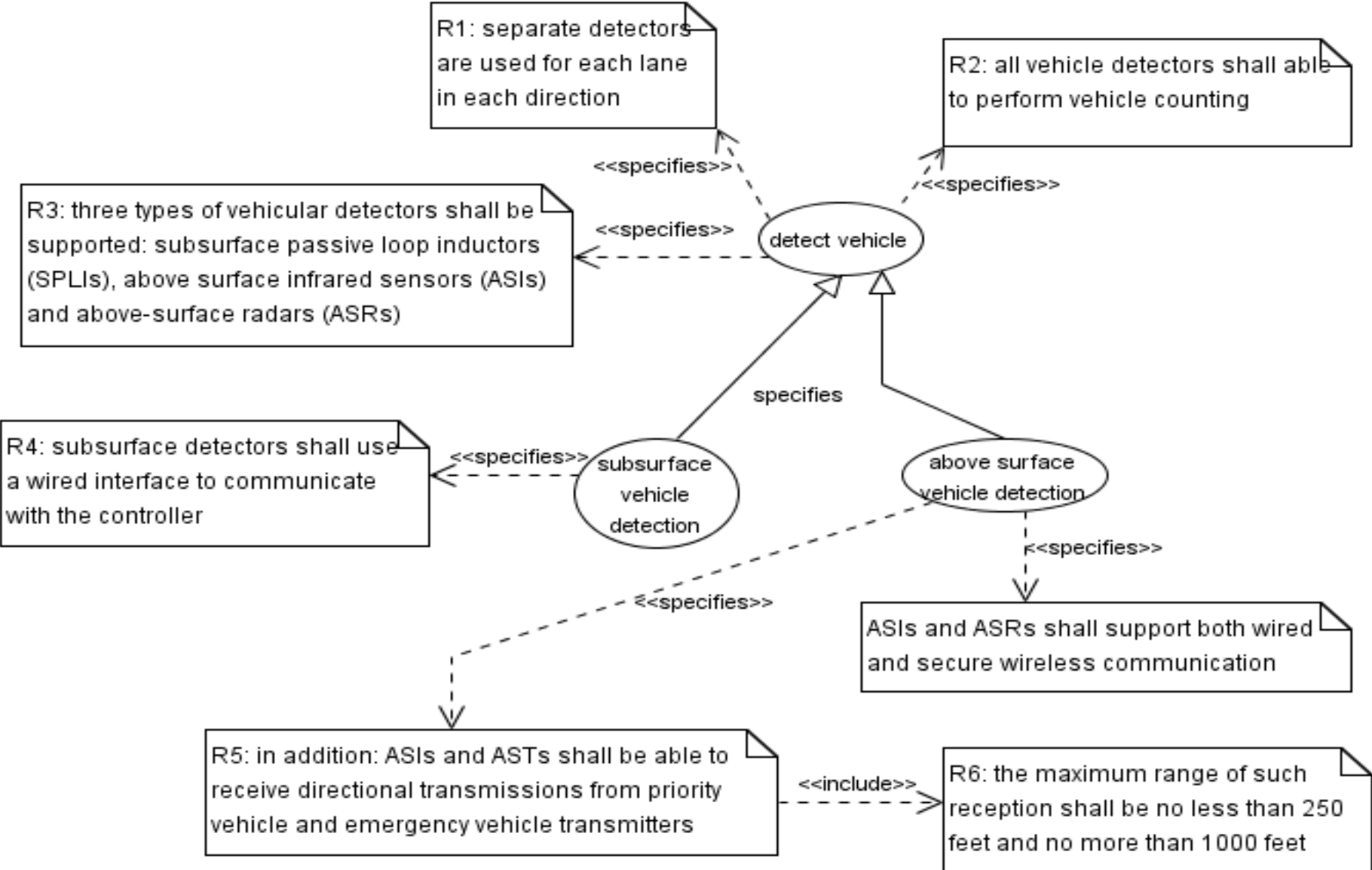
- Identify types of requirements
- Group requirements together
- **Identify use cases**
- Capture and represent quality of service requirements
- Use state machines for representing requirements

Identify Use Cases

- It returns a value to at least one actor
- It contains at least three scenarios, each scenario consisting of multiple actor-system messages
- It logically contains many operational, function, or quality of service requirements
- It does not reveal or imply anything about internal structure of the system
- It is independent of other use cases and may (or may not) be concurrent with them
- There should be no fewer than three and no more than three dozen use cases at the “high level”
 - <<include>> and <<extended>>
 - generalization

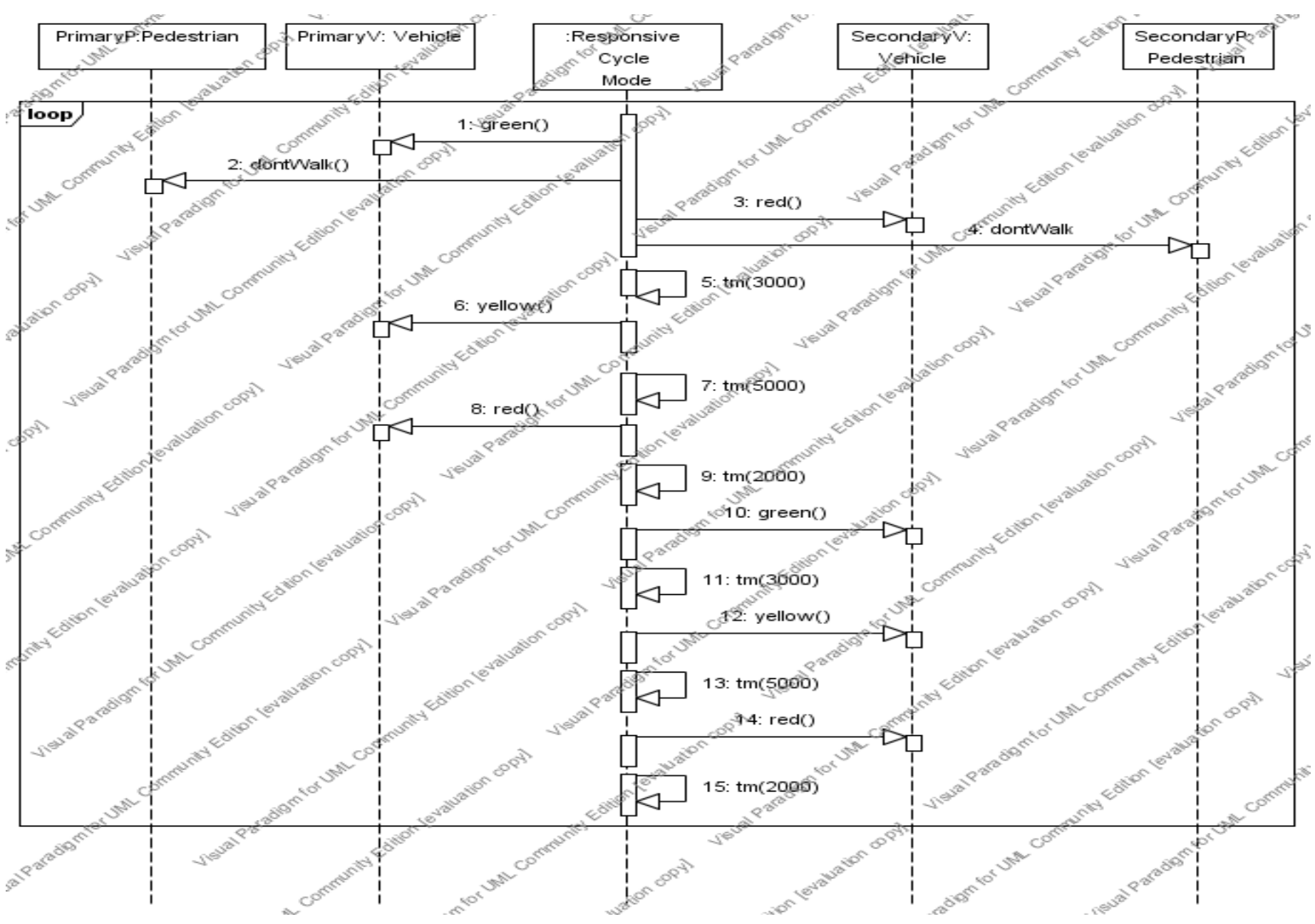


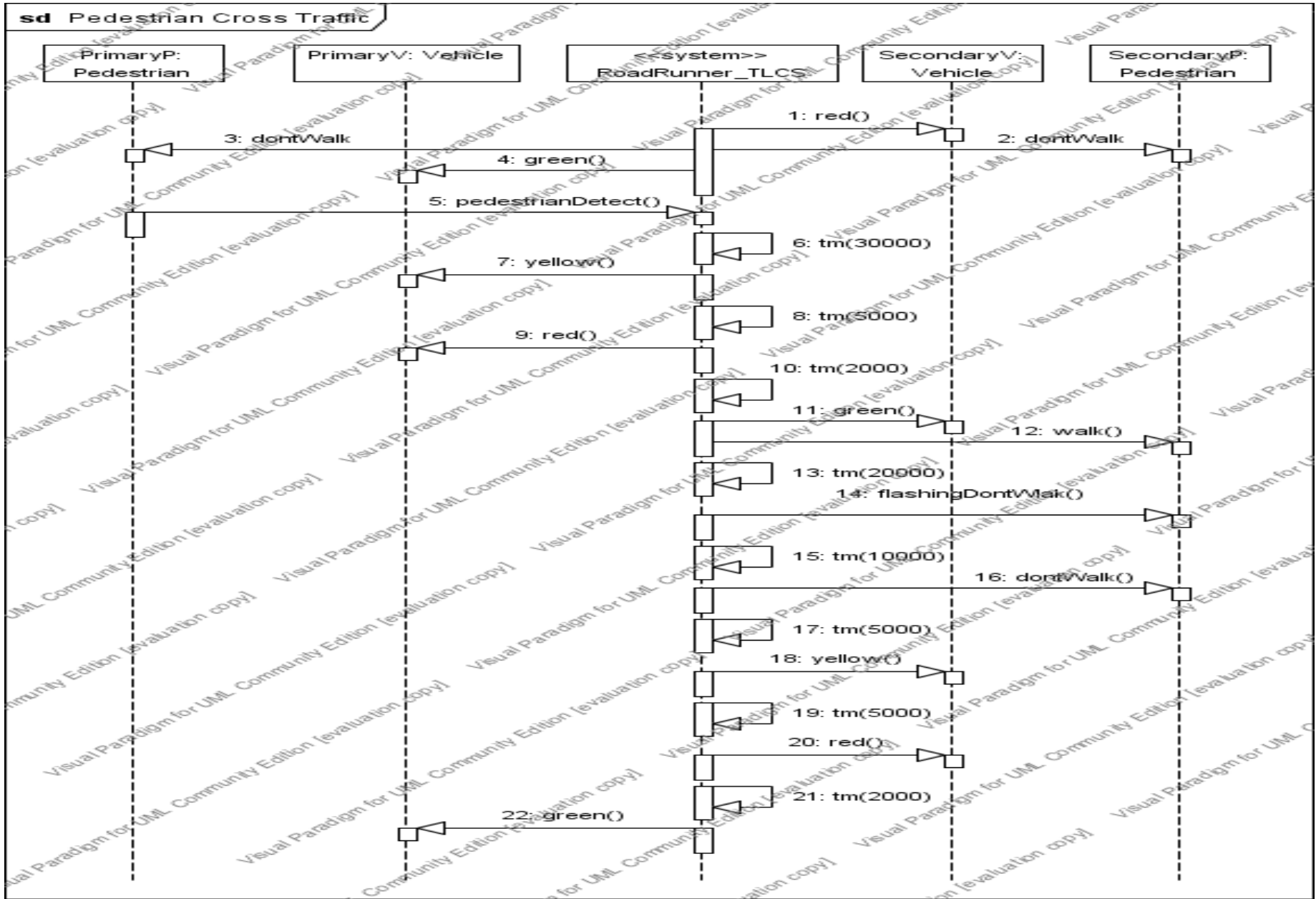
Mapping Requirements to Use Cases



Operational View

- A **scenario** can be thought of as a specific sequence of inputs and outputs that represents a single path through a use case.
- A use case normally has many different scenarios
- Scenarios are useful because they relate the system capabilities with the expectations of how the system will interact with other elements in its environment.
- Scenarios are valuable at every level of abstraction
 - *System level*: single lifeline for the system others are actors
 - *Subsystem level*: system lifeline is “opened up” to subsystem
 - *Collaboration level*: object roles collaboration





Use Case Description

- Name
- Owner
- Purpose
- Requirements (optional)
- Data (optional)
- Preconditions
- Postconditions
- Reference documents

Use Case Descriptions

The screenshot shows a software interface for editing use case details. The title bar reads 'detect vehicle Details'. Below the title bar, there is a 'Name:' field containing 'detect vehicle'. A tabbed interface has 'Info', 'Description', and 'Diagrams' tabs, with 'Info' currently selected. A rich text editor toolbar is visible, including icons for undo, redo, bold, italic, underline, and bulleted list, along with a font color dropdown set to 'Agency FB' and a font size dropdown set to '8'. On the left, a tree view shows 'Detect Vehicle' selected. The main area displays a table with the following data:

Super Use Case		
Author	user	
Date	2008/11/13 下午 03:28:22	
Brief Description	this use case specify how the system detects vehicle and the information it gathers when is does so. It is used in managing traffic in responsive modes of operation (not in fixed cycle or safe modes). Vehicle counting is used in adaptive modes to adjust the length of cycle times and also to provide demographic statistics to the system operators	
Preconditions	the system must be configured to use whichever vehicle detection method (below-surface passive induction, above-surface infrared, or above-surface radar	
Post-conditions	approaching vehicles are detected and counted	
Flow of Events		
	Actor Input	System Response
1		

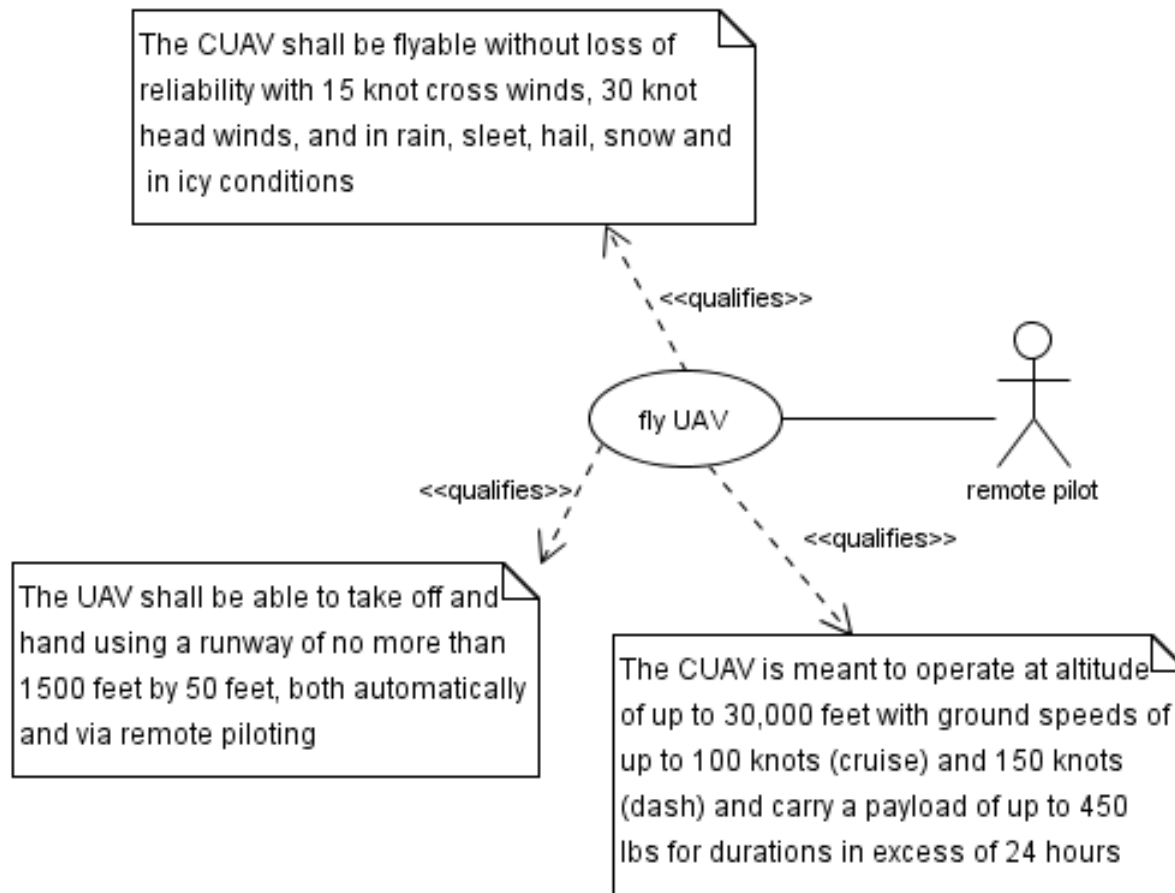
Case Study: Specifying Requirements

- Identify types of requirements
- Group requirements together
- Identify use cases
- **Capture and represent quality of service requirements**
- Use state machines for representing requirements

CAPTURING QoS REQUIREMENTS IN USE CASES

- QoS specifies “how much”, “how fast”, and “to what degree”.
 - Nonfunctional requirements (or QoS requirements)

QoS REQUIREMENTS



Case Study: Specifying Requirements

- Identify types of requirements
- Group requirements together
- Identify use cases
- Capture and represent quality of service requirements
- Use state machines for representing requirements

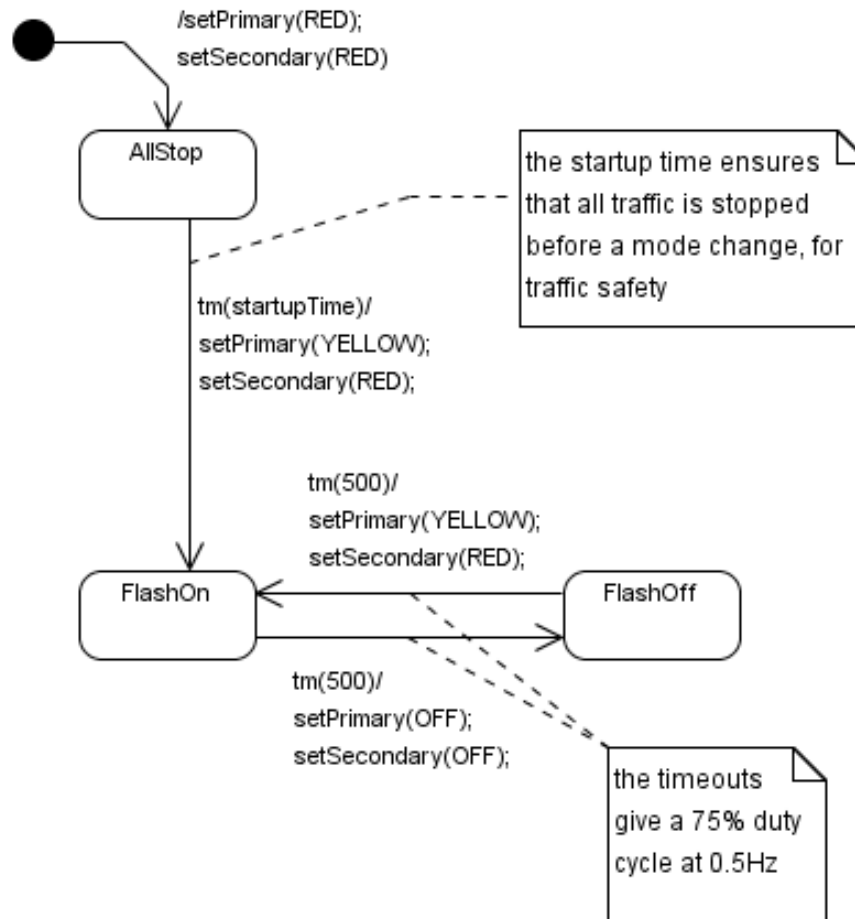
SPECIFICATION VIEW: CAPTURING COMPLEX REQUIREMENTS

- Using state machine diagram to specify the requirements of complex “*reactive behavior*”
- The advantages of state machine over the use of text for requirements specifications
 - State machine is precise and unambiguous
 - State machine is executable (in determining the consistency, correctness, and accuracy of complex sets of requirements)
 - State machine is testable

Guidelines for Use-Case State Machine

- Incoming message or commands become events (possibly with data attached as parameters) in the state machine
 - Outgoing message or commands become actions in the state machine
 - Underlying technology and design should not be expressed in the state machine
 - Scenarios for the use cases are nothing more than sets of transition “paths” through the state machine
 - Every scenario should represent transition path
 - For minimal coverage, every transition should be represented in at least one scenario
-

Evening Low Volume Mode



Basic Fixed Cycle Mode

