Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 19:

Use Case Realization – Design

Department of Computer Engineering

Design Workflow: *Design a Use Case*

- Place in the *Design Workflow:*
  - Architectural Design
  - **Design a Use Case**
  - Design a Class
  - Design a Subsystem
- The activity *Design a Use Case* is concerned with finding the design classes, interfaces, and components that interact to provide the behavior specified by a use case.

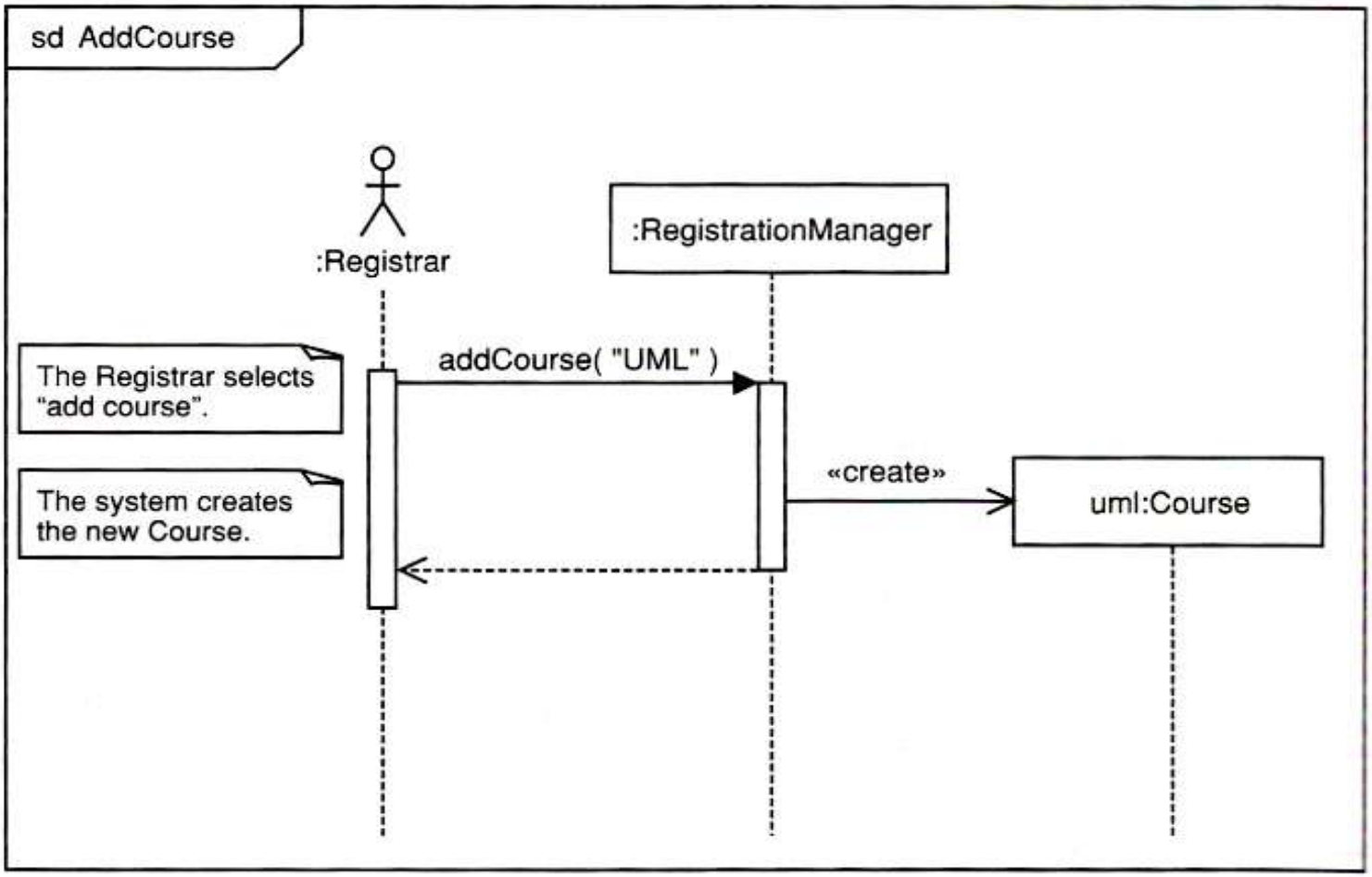
Use Case Realizations - Design

- Use case realization in design is really just an extension of use case realization in analysis.
- Use case realizations-design are collaborations of design objects and classes that realize a use case; models consist of:
  - design interaction diagrams - these are refinements of analysis interaction diagrams;
  - design class diagrams - these are refinements of analysis class diagrams.
- Interaction diagrams can be used in design to model central mechanisms such as object persistence; these mechanisms may cut across many use cases.

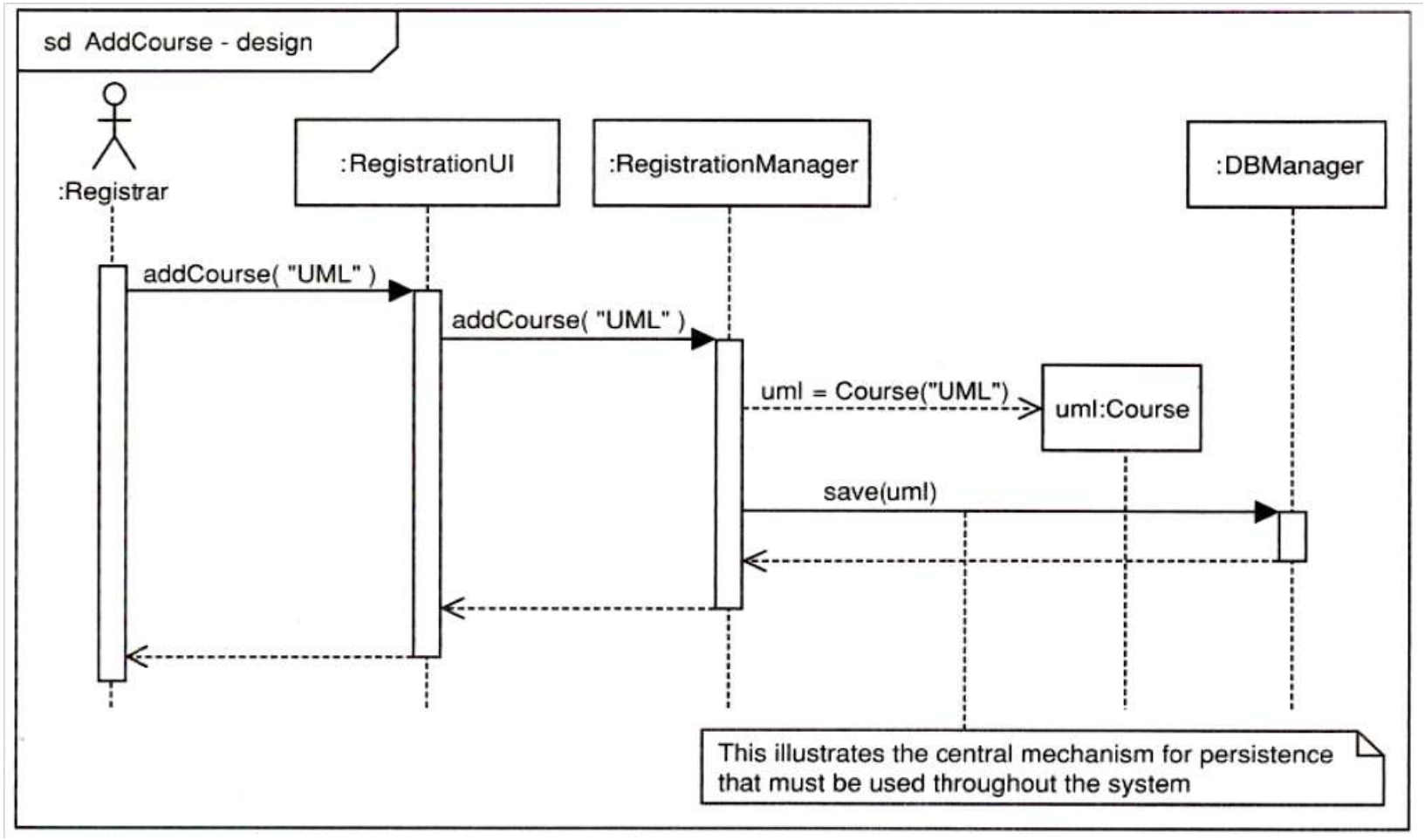
Interaction Diagrams: Example – Use Case

Use case: AddCourse
ID: 8
Brief description: Add details of a new course to the system.
Primary actors: Registrar
Secondary actors: None.
Preconditions: 1. The Registrar has logged on to the system.
Main flow: 1. The Registrar selects "add course". 2. The Registrar enters the name of the new course. 3. The system creates the new course.
Postconditions: 1. A new course has been added to the system.
Alternative flows: CourseAlreadyExists

Interaction Diagrams: Example – Analysis



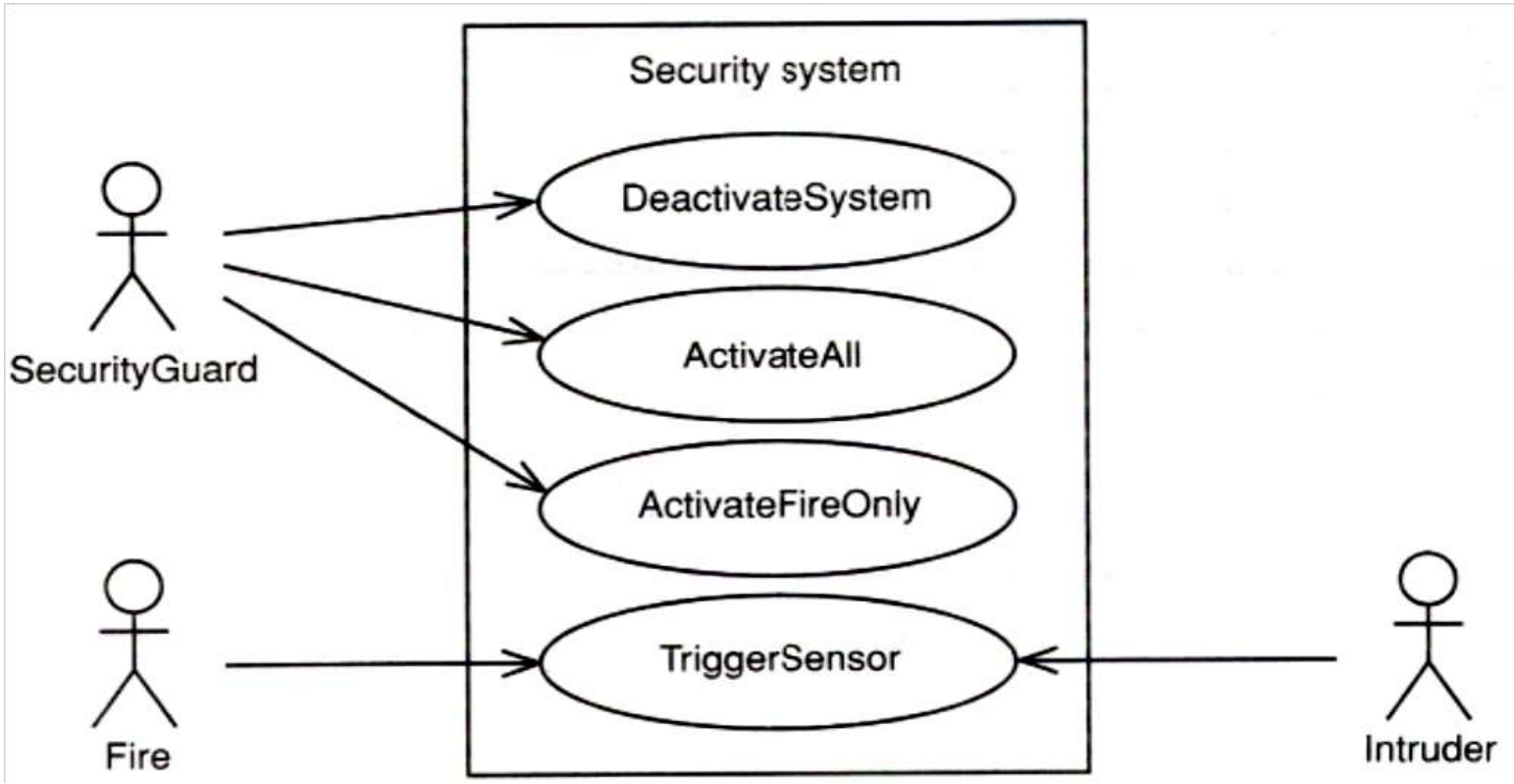
Interaction Diagrams: Example – Design



Modeling Concurrency

- Use active classes and objects.
- Sequence diagrams:
  - **par** - all operands execute in parallel;
  - **critical** - the operand executes atomically without interruption.
- Communication diagrams:
  - postfix the sequence number with a label to indicate the thread of control.
- Activity diagrams:
  - forks;
  - joins.

Concurrency: Sample System – Use Case Diagram



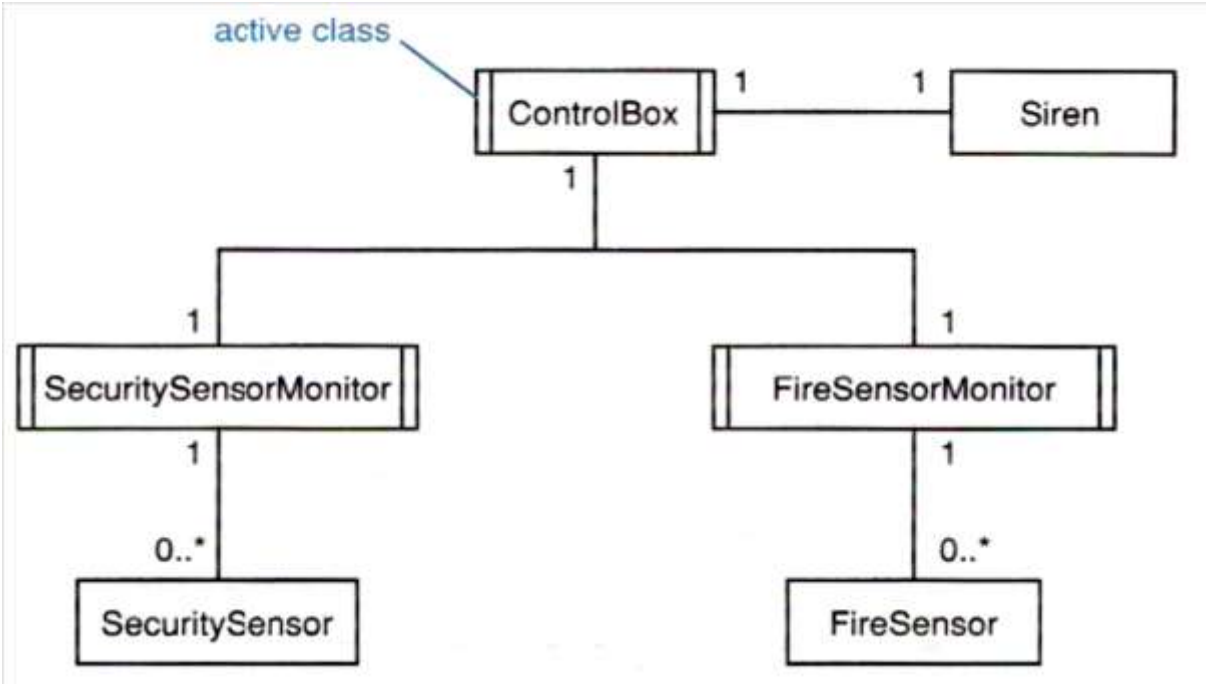
Concurrency: Sample System – Use Cases

Use case: DeactivateSystem
ID: 1
Brief description: Deactivate the system.
Primary actors: SecurityGuard
Secondary actors: None.
Preconditions: 1. The SecurityGuard has the activation key.
Main flow: 1. The SecurityGuard uses the activation key to switch the system off. 2. The system stops monitoring security sensors and fire sensors.
Postconditions: 1. The security system is deactivated. 2. The security system is not monitoring the sensors.
Alternative flows: None.

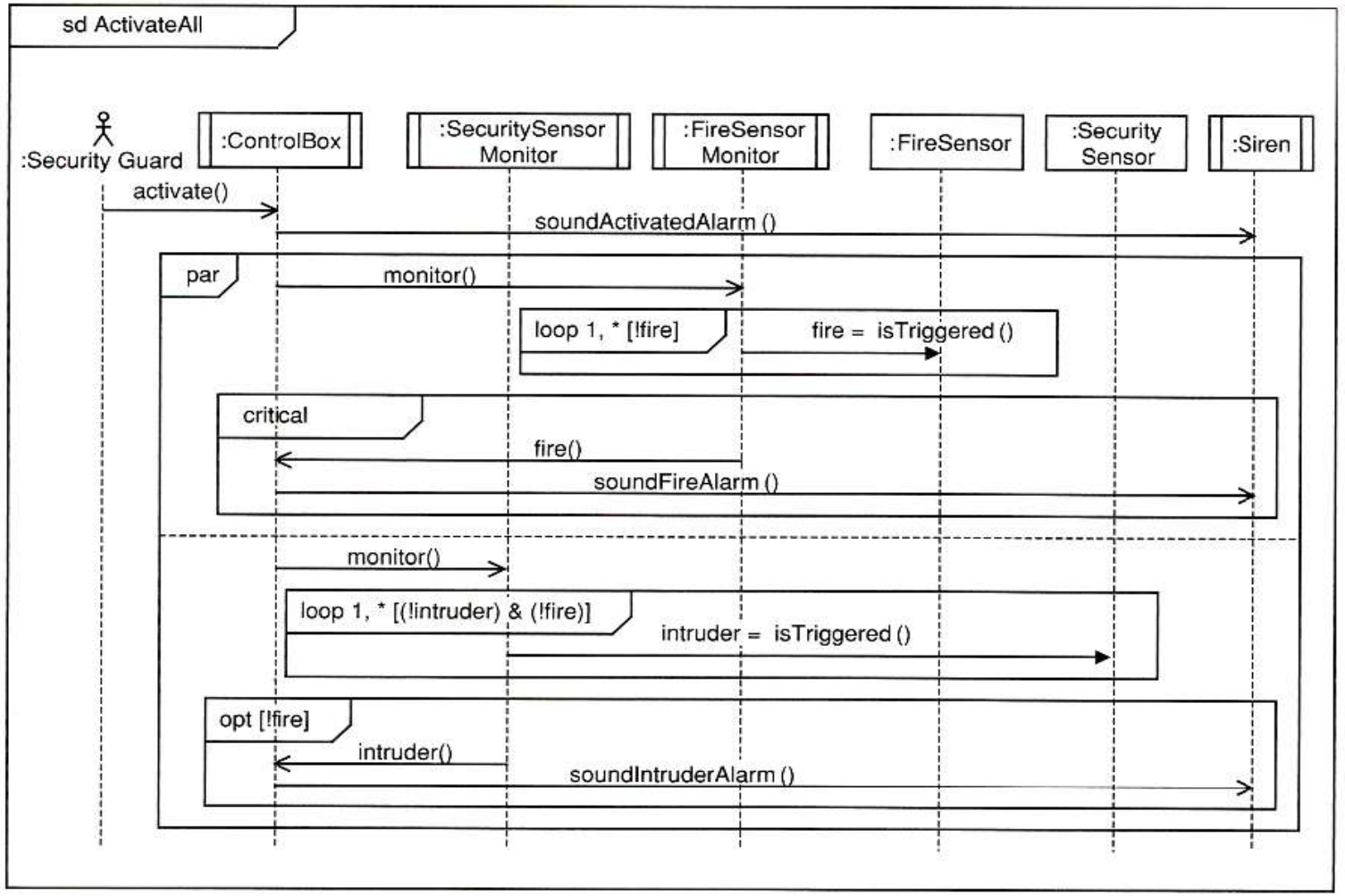
Use case: ActivateAll
ID: 2
Brief description: Activate the system.
Primary actors: SecurityGuard
Secondary actors: None.
Preconditions: 1. The SecurityGuard has the activation key.
Main flow: 1. The SecurityGuard uses the activation key to switch the system on. 2. The system starts monitoring security sensors and fire sensors. 3. The system sounds the siren to indicate that it is armed.
Postconditions: 1. The security system is activated. 2. The security system is monitoring the sensors.
Alternative flows: None.

Use case: TriggerSensor
ID: 3
Brief description: A sensor is triggered.
Primary actors: Fire Intruder
Secondary actors: None.
Preconditions: 1. The security system is activated.
Main flow: 1. If the Fire actor triggers a FireSensor 1.1 The Siren sounds a fire alarm. 2. If the Security actor triggers a SecuritySensor 2.1 The Siren sounds a security alarm.
Postconditions: 1. The Siren sounds.
Alternative flows: None.

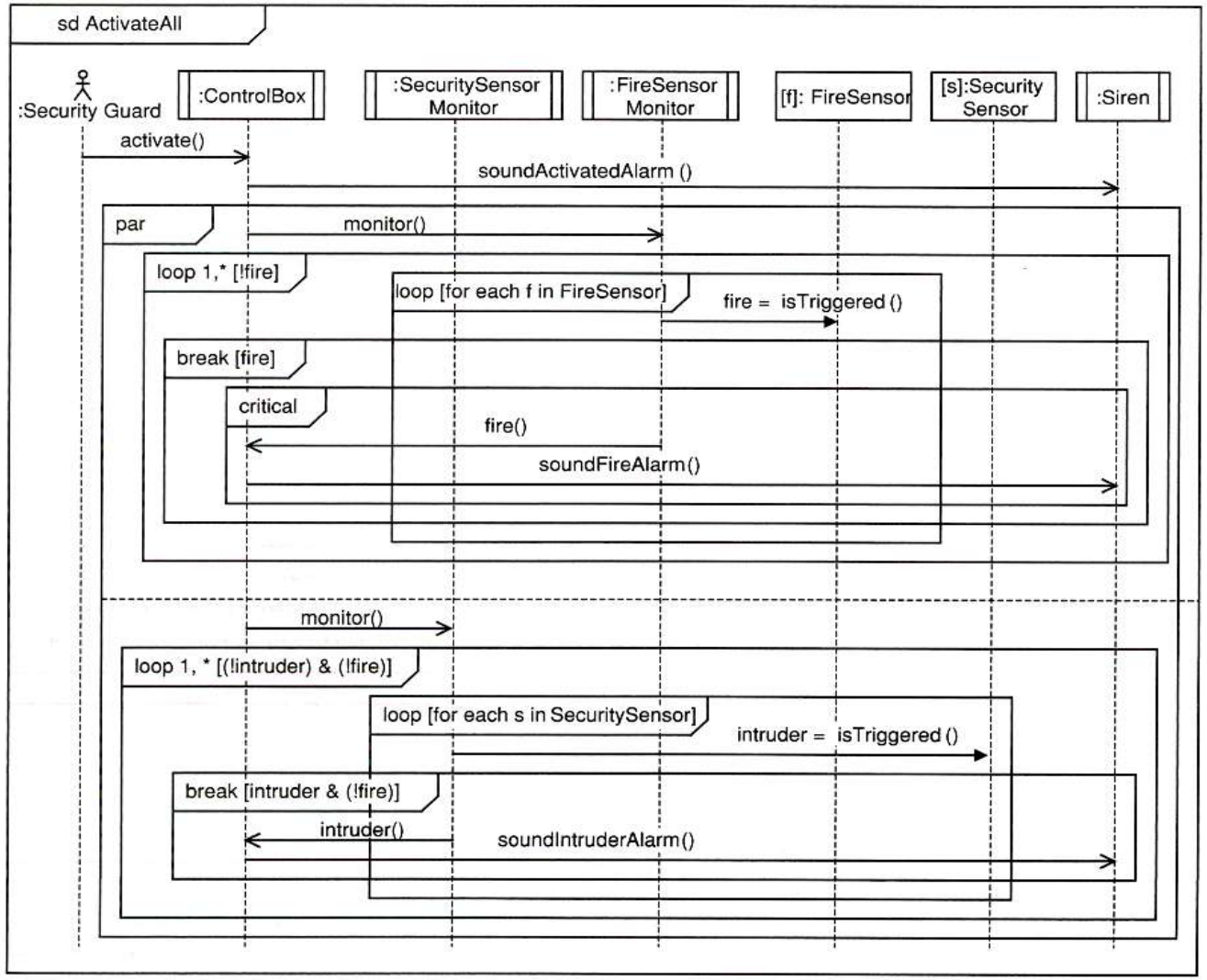
Concurrency: Active Classes



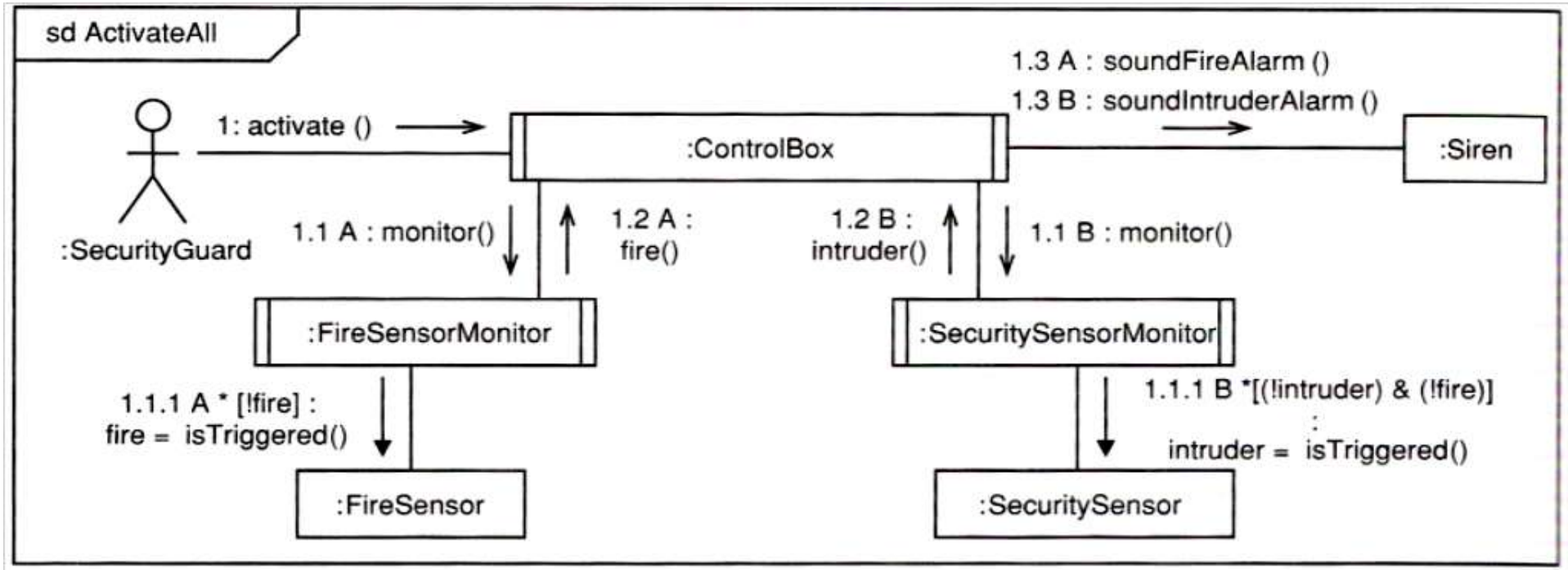
Concurrency: Interaction Diagrams (1)



Concurrency: Interaction Diagrams (2)

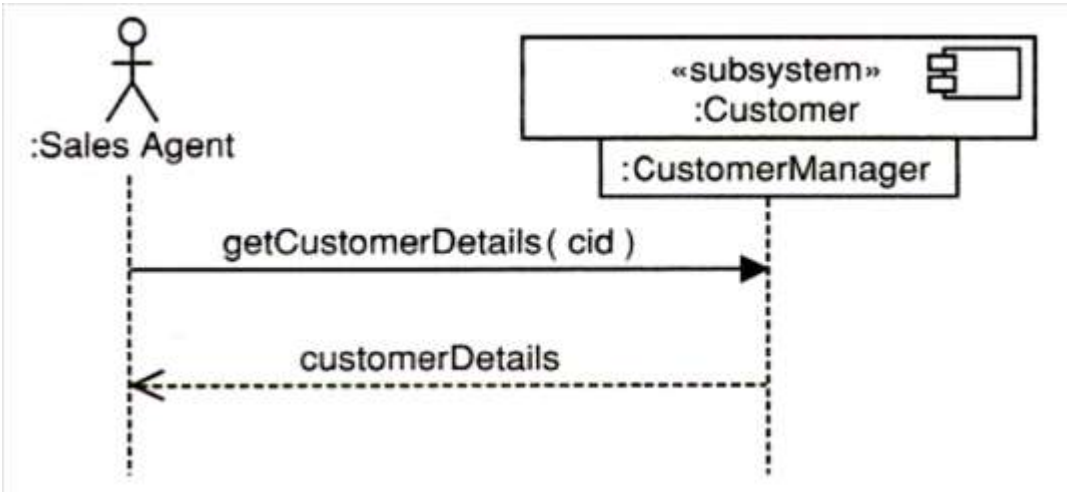


Concurrency: Communication Diagrams



Subsystem Interaction Diagrams

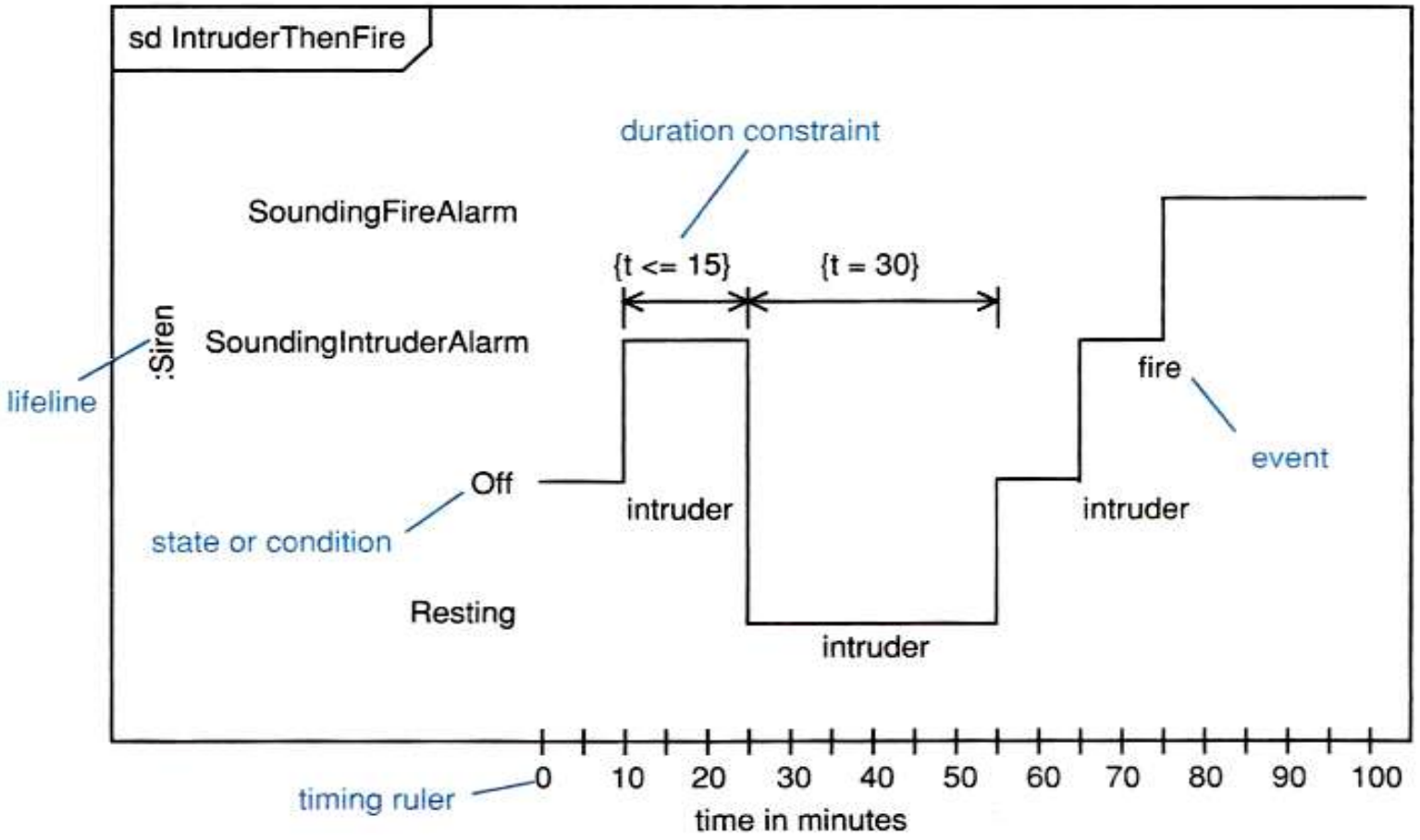
- Subsystem interaction diagrams show the interactions between the different parts of the system at a high level:
  - they may contain actors, subsystems, components, and classes;
  - you can show parts of the subsystem (e.g., provided interfaces) in boxes hanging down below the subsystem.



Timing Diagrams

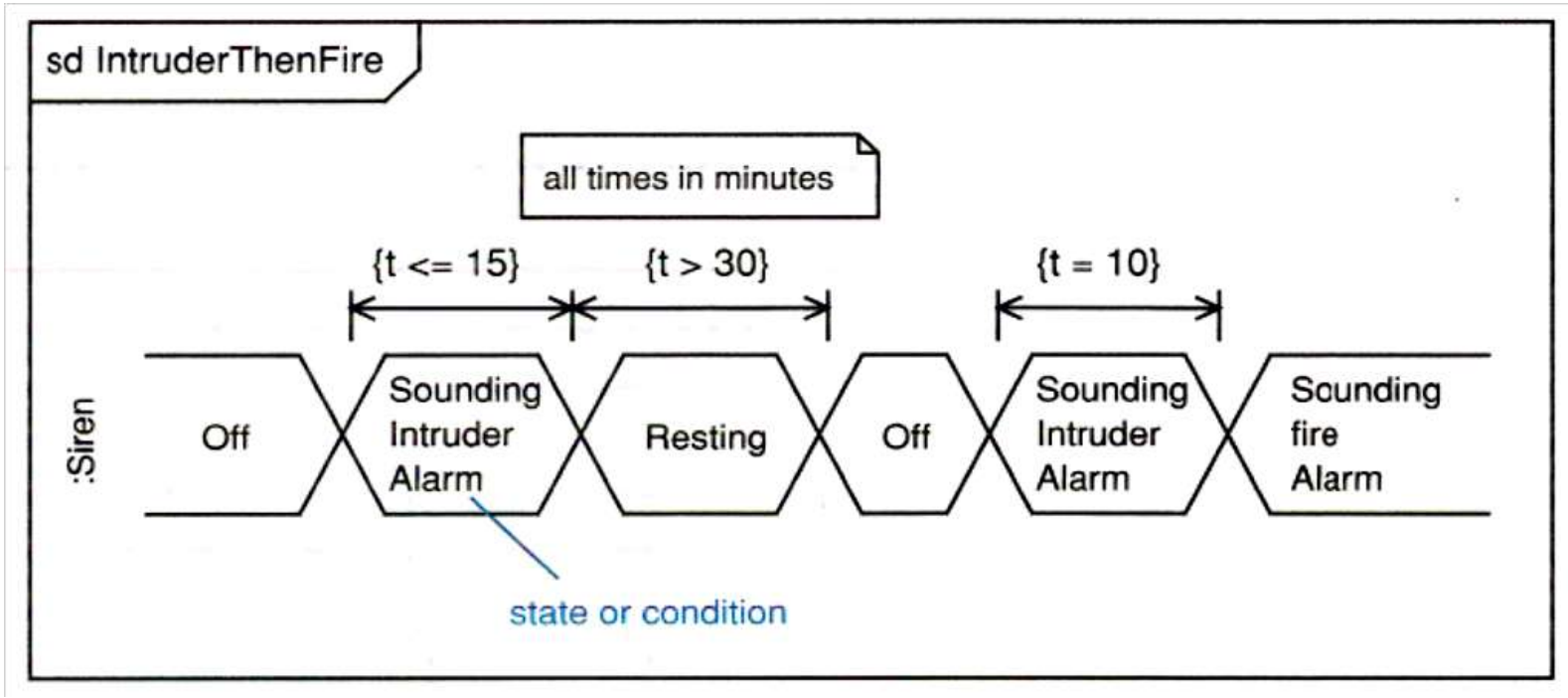
- Timing diagrams - model timing constraints:
  - very useful for modeling real-time and embedded systems;
  - time increases horizontally from left to right;
  - lifelines, states, and conditions are placed vertically;
  - transitions between states or conditions are shown as a graph;
  - timing constraints and events can be shown.

Timing Diagrams: Notation

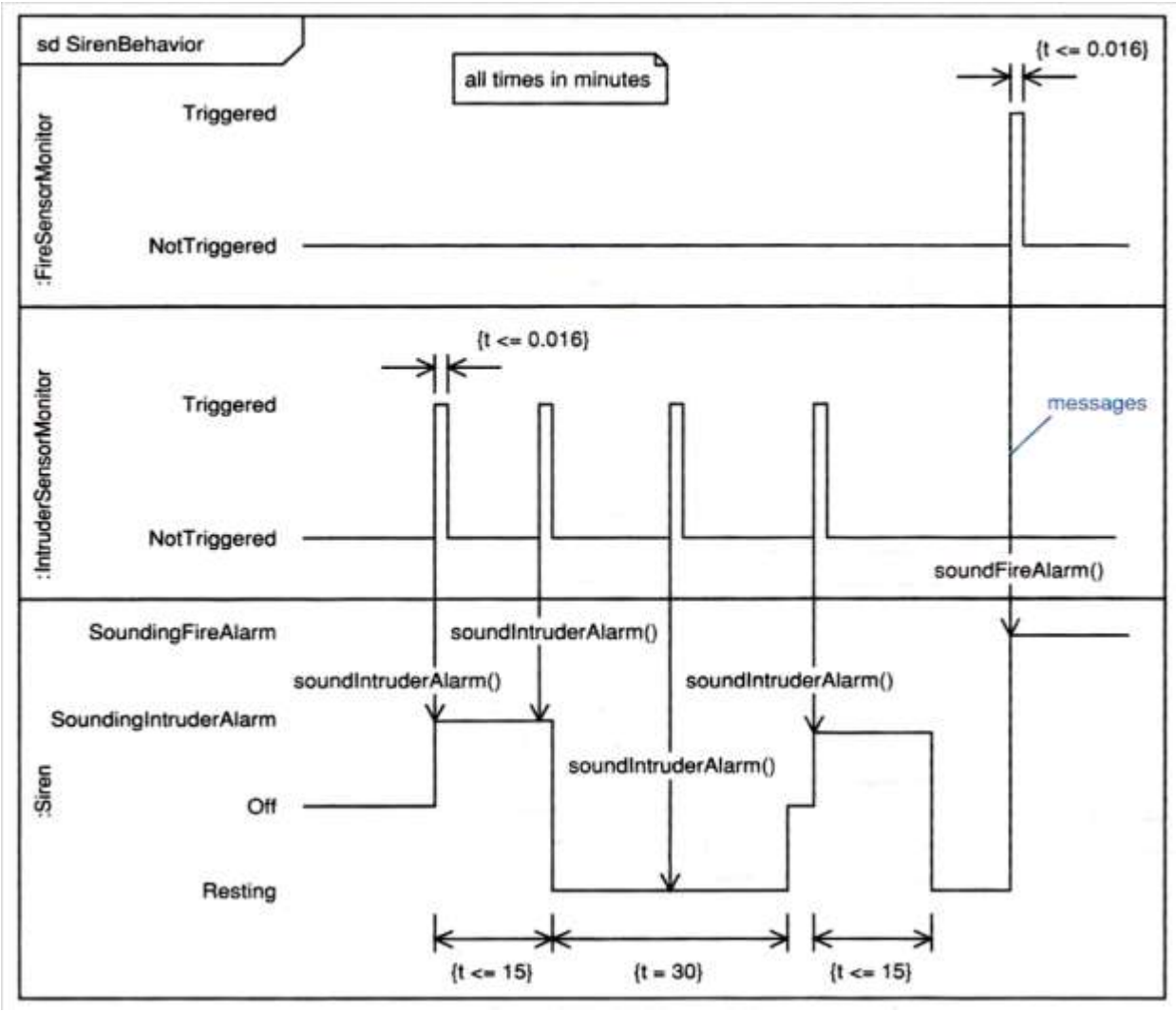


Timing Diagrams: Compact Form

- The compact form emphasizes relative time.



Timing Diagrams: Complex Timing Constraints



Reference

- Arlow, J., Neustadt, I., *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, 2nd Ed. Addison-Wesley, 2005.