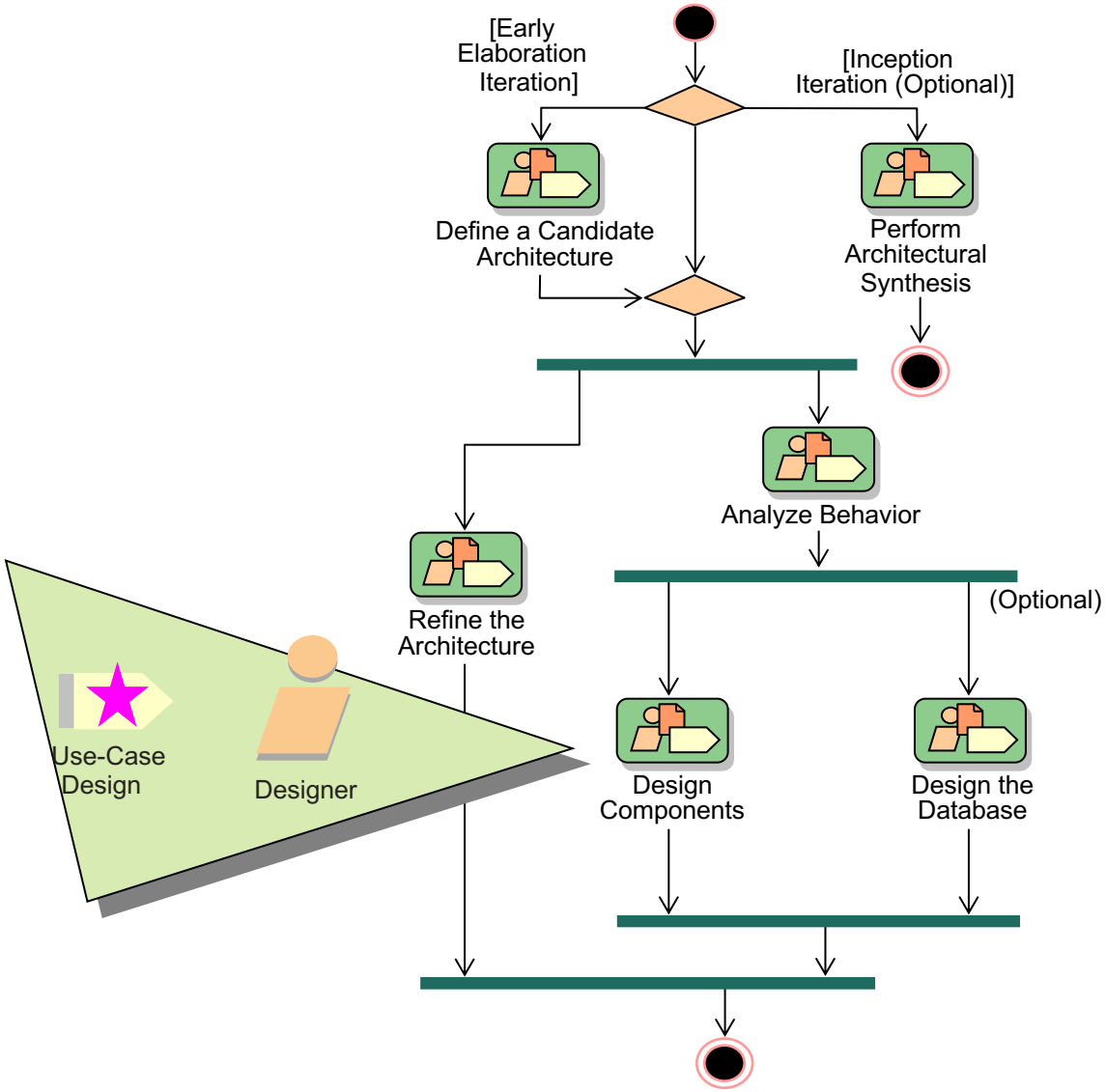
Software analysis and design

Module 15: Use-Case Design

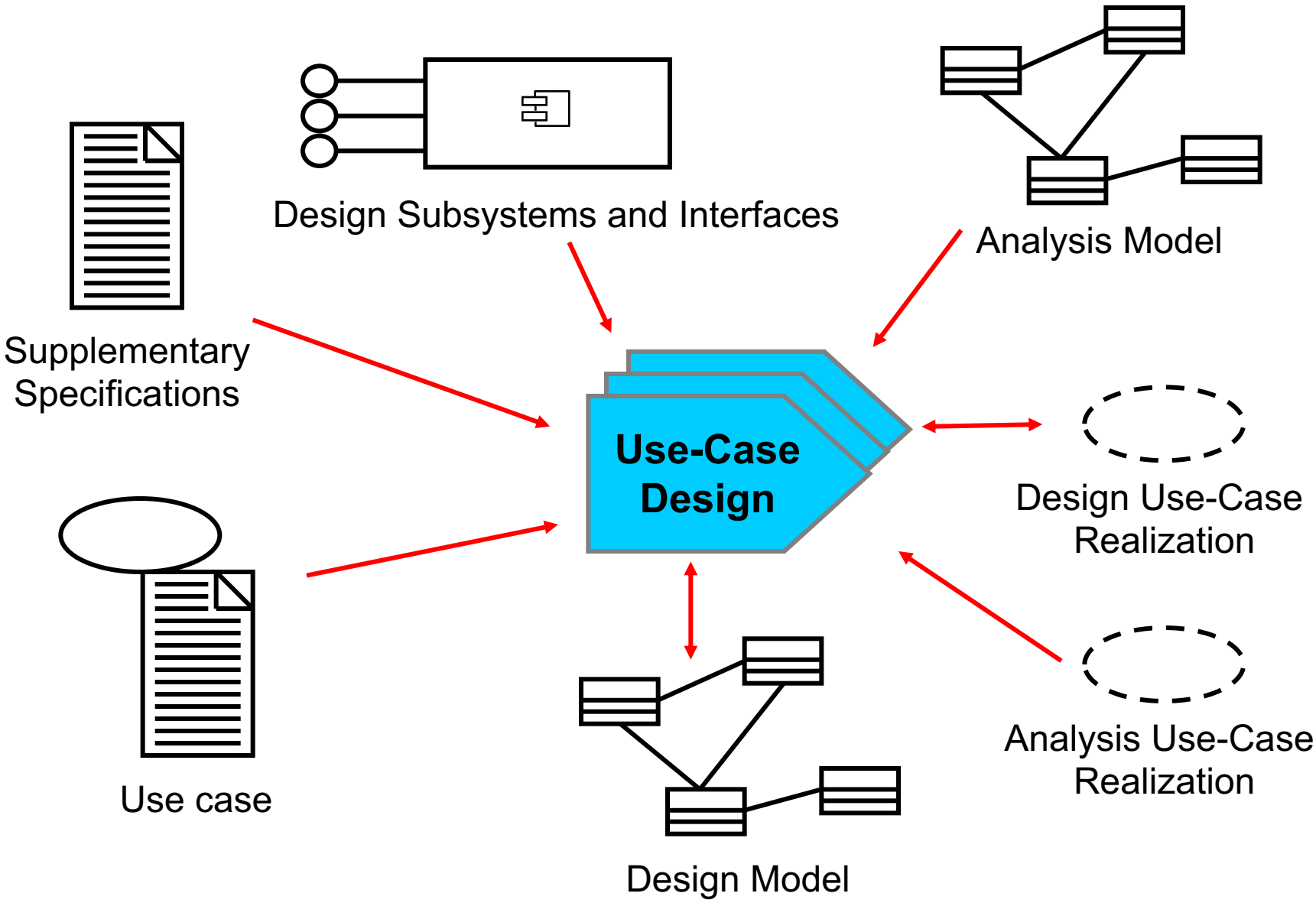
Objectives: Use-Case Design

- Define the purpose of Use-Case Design and when in the lifecycle it is performed
- Verify that there is consistency in the use-case implementation
- Refine the use-case realizations from Use-Case Analysis using defined Design Model elements

Use-Case Design in Context



Use-Case Design Overview



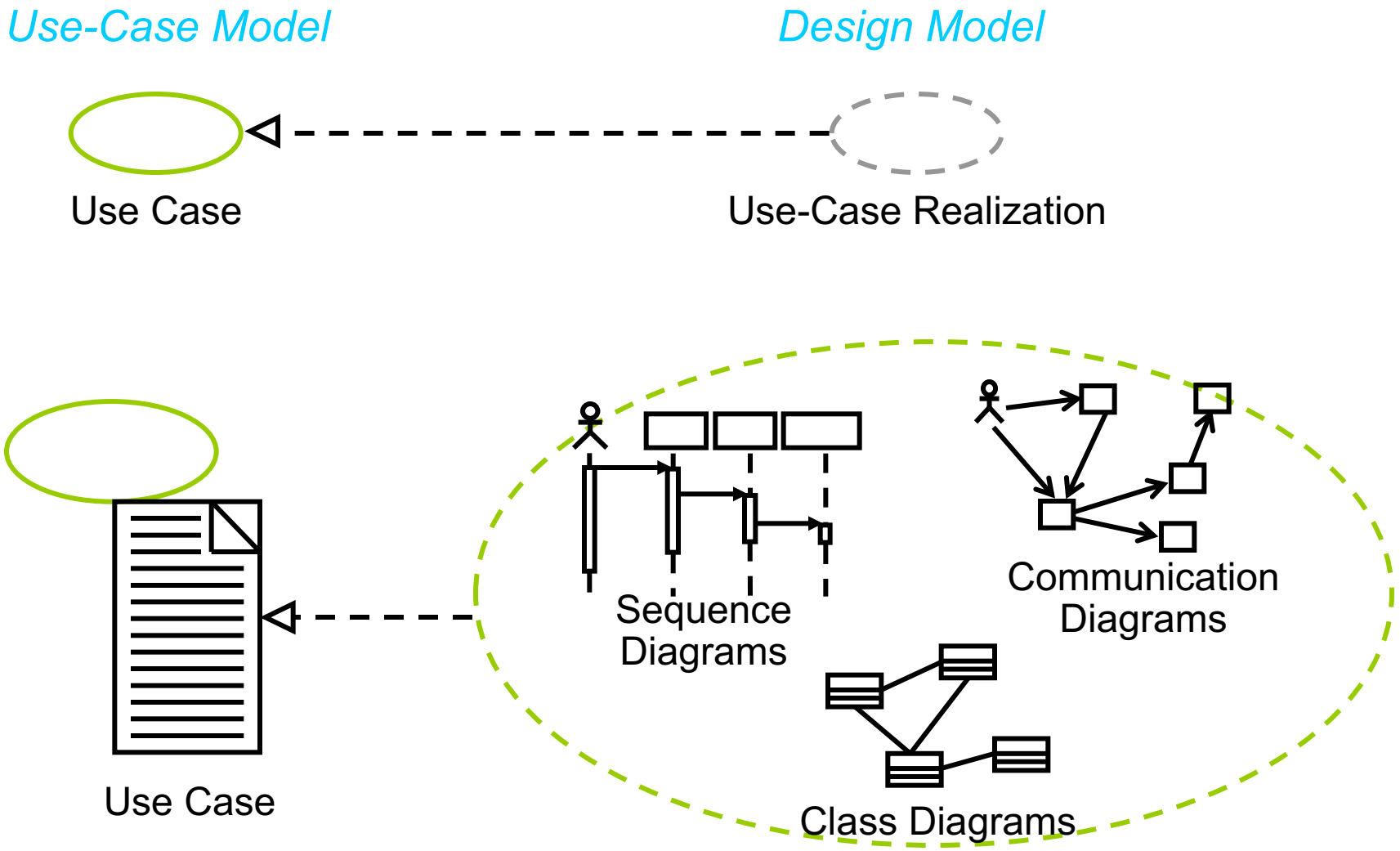
Use-Case Design Steps

- Describe interaction among design objects
- Simplify sequence diagrams using subsystems
- Describe persistence-related behavior
- Refine the flow of events description
- Unify classes and subsystems

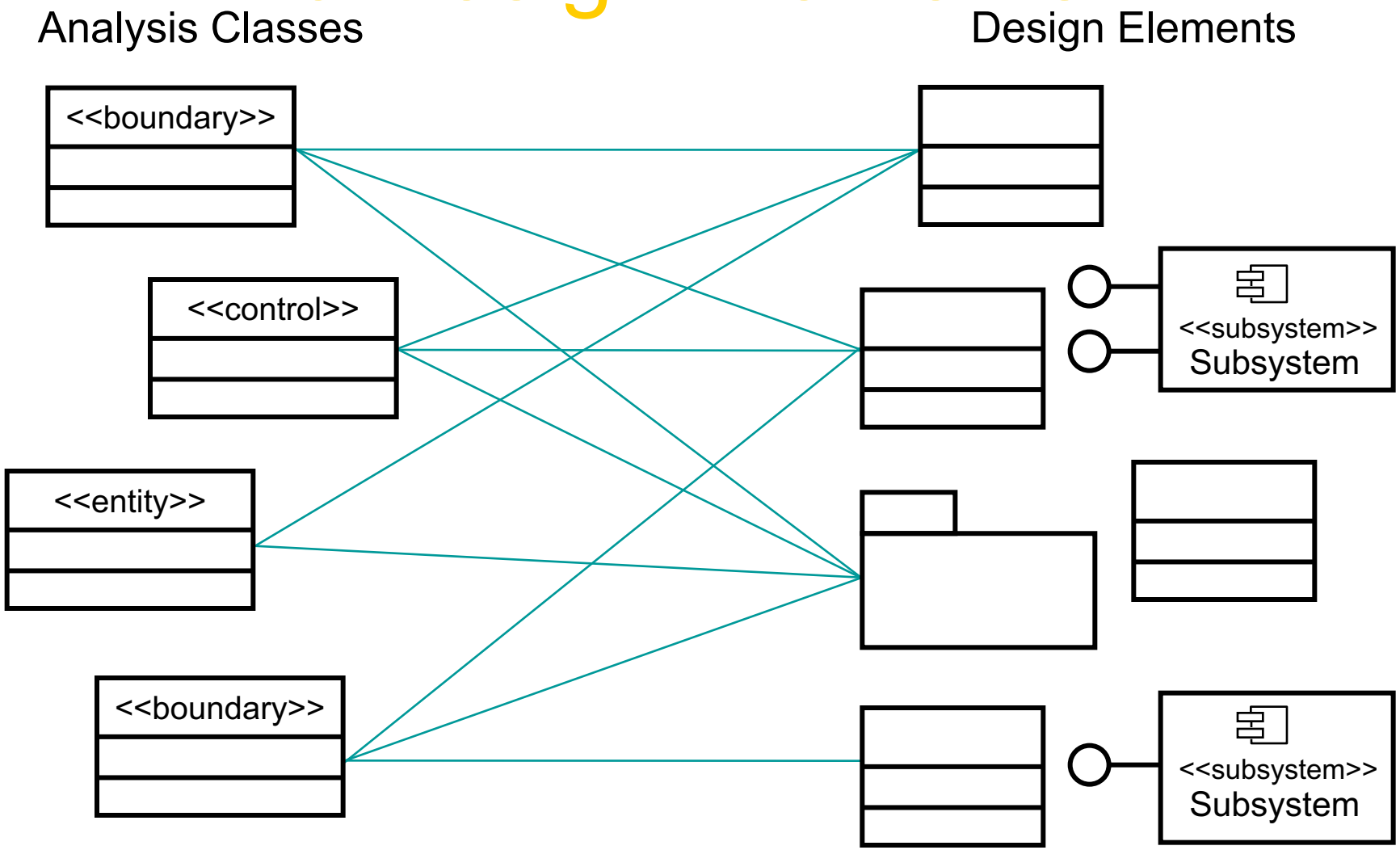
Use-Case Design Steps

- Describe interaction among design objects
- Simplify sequence diagrams using subsystems
- Describe persistence-related behavior
- Refine the flow of events description
- Unify classes and subsystems

Review: Use-Case Realization



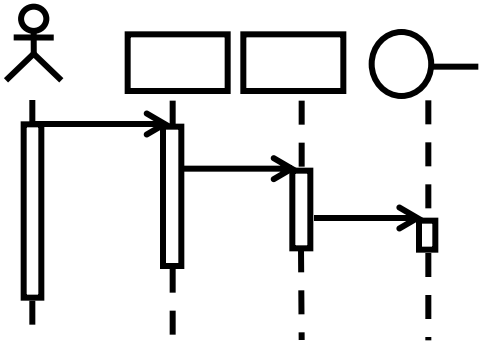
Review: From Analysis Classes to Design Elements



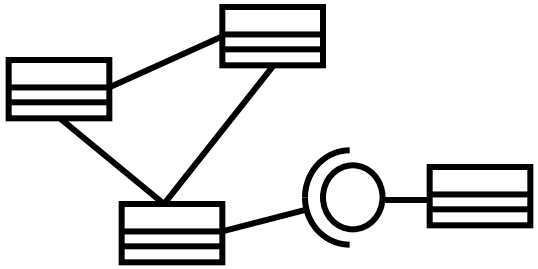
Many-to-Many Mapping

Use-Case Realization Refinement

- Identify participating objects
- Allocate responsibilities among objects
- Model messages between objects
- Describe processing resulting from messages
- Model associated class relationships



Sequence Diagrams



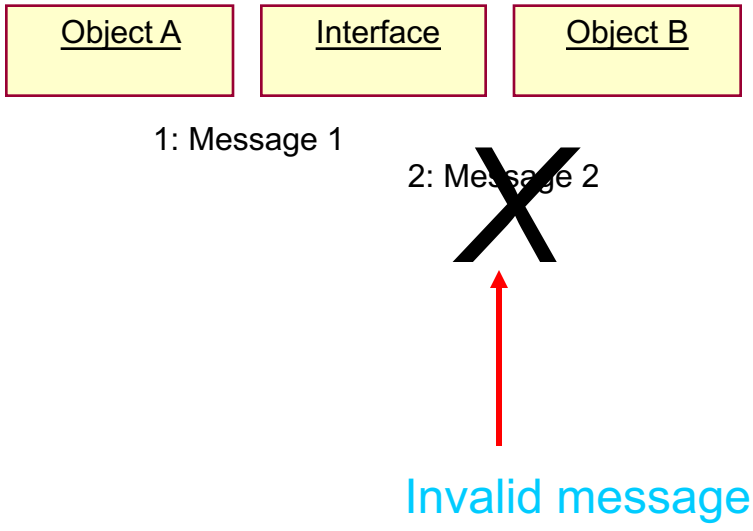
Class Diagrams

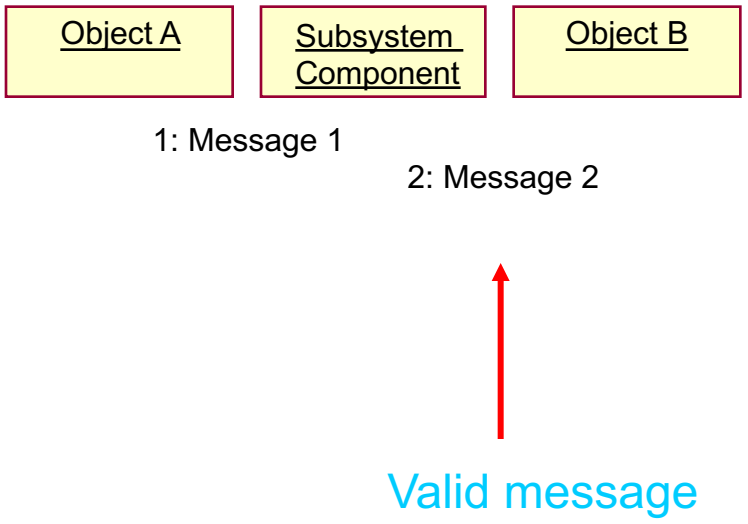
Use-Case Realization Refinement Steps

1. Identify each object that participates in the flow of the use case
2. Represent each participating object in a sequence diagram
3. Incrementally incorporate applicable architectural mechanisms

Representing Subsystems on a Sequence Diagram

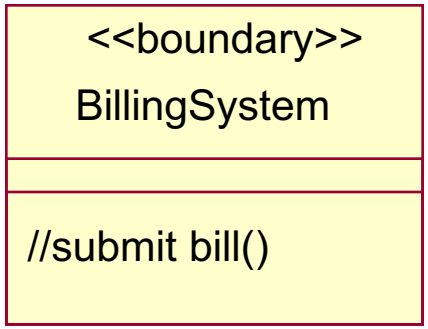
- Interfaces
  - Represent any model element that realizes the interface
  - No message should be drawn from the interface
- Subsystem Component
  - Represents a specific subsystem
  - Messages can be drawn from the subsystem

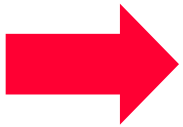




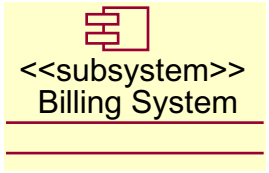
Example: Incorporating Subsystem Interfaces

Analysis Classes



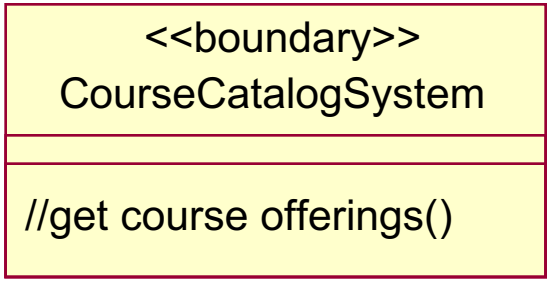


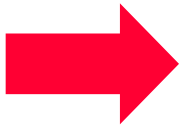
Design Elements



IBillingSystem

submitBill(forTuition : Double, forStudent : Student)





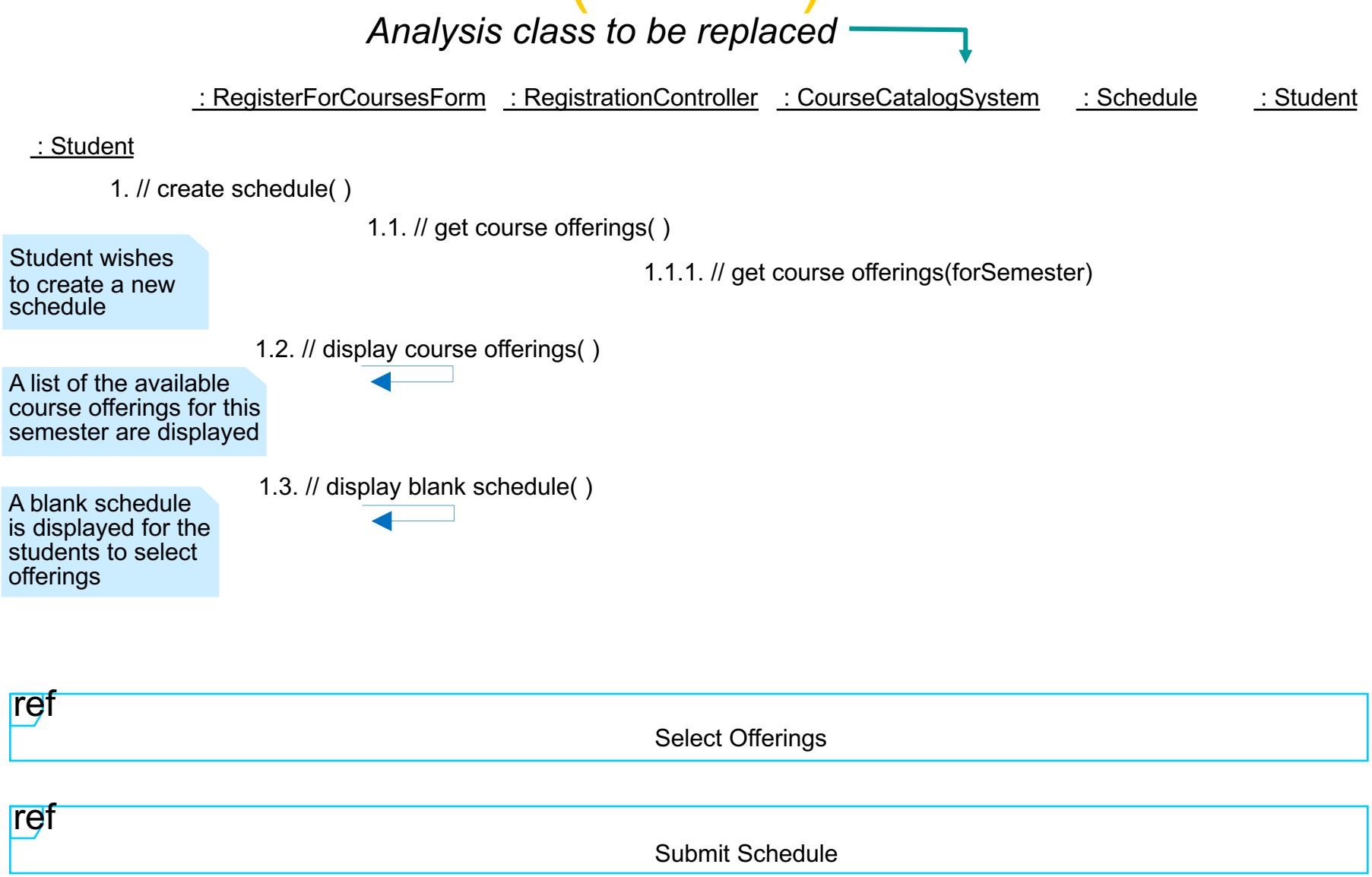
ICourseCatalogSystem

getCourseOfferings(forSemester : Semester, forStudent : Student) : CourseOfferingList
initialize()

All other analysis classes are mapped directly to design classes.

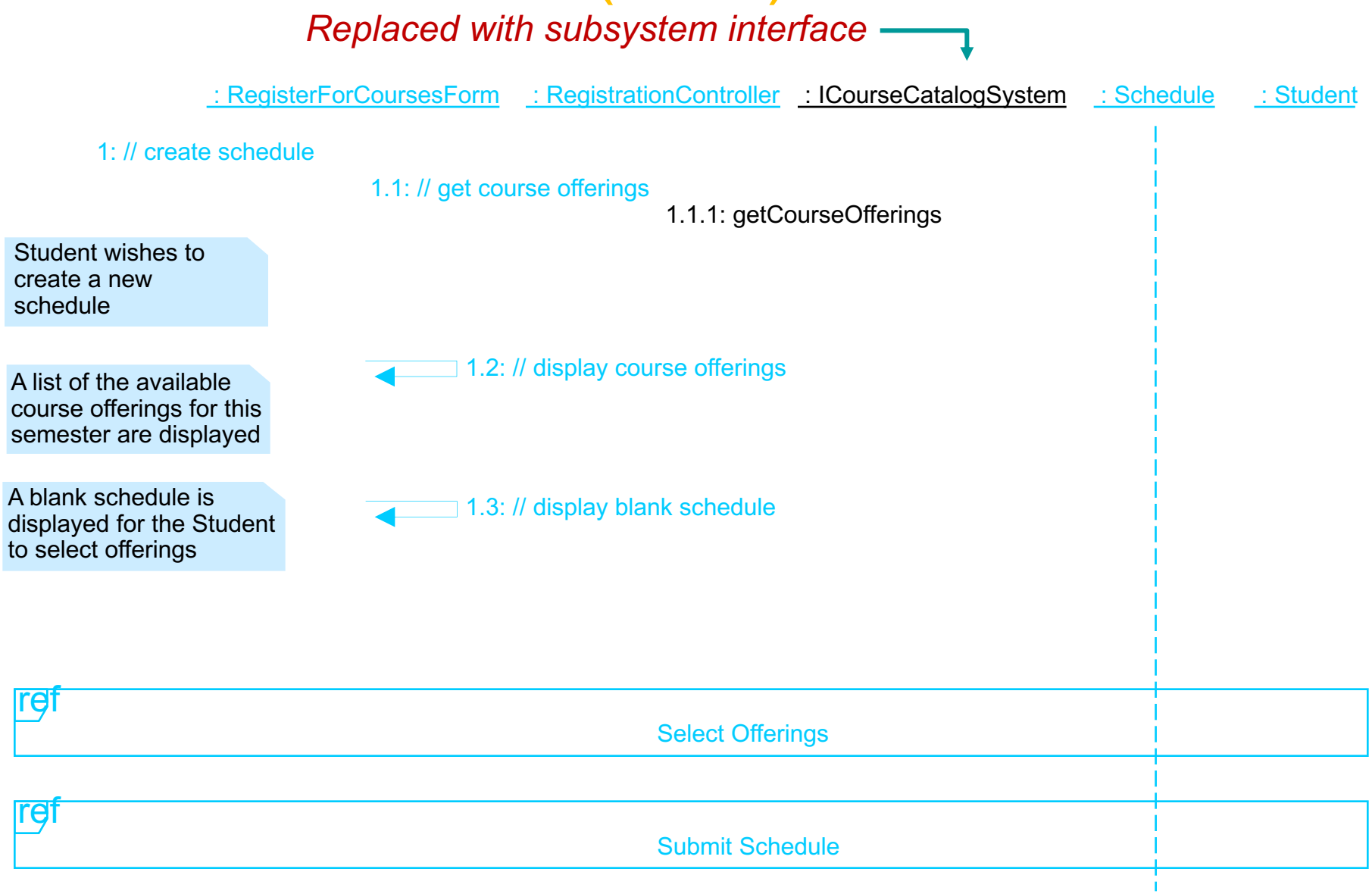
Example: Incorporating Subsystems (Before)

Analysis class to be replaced

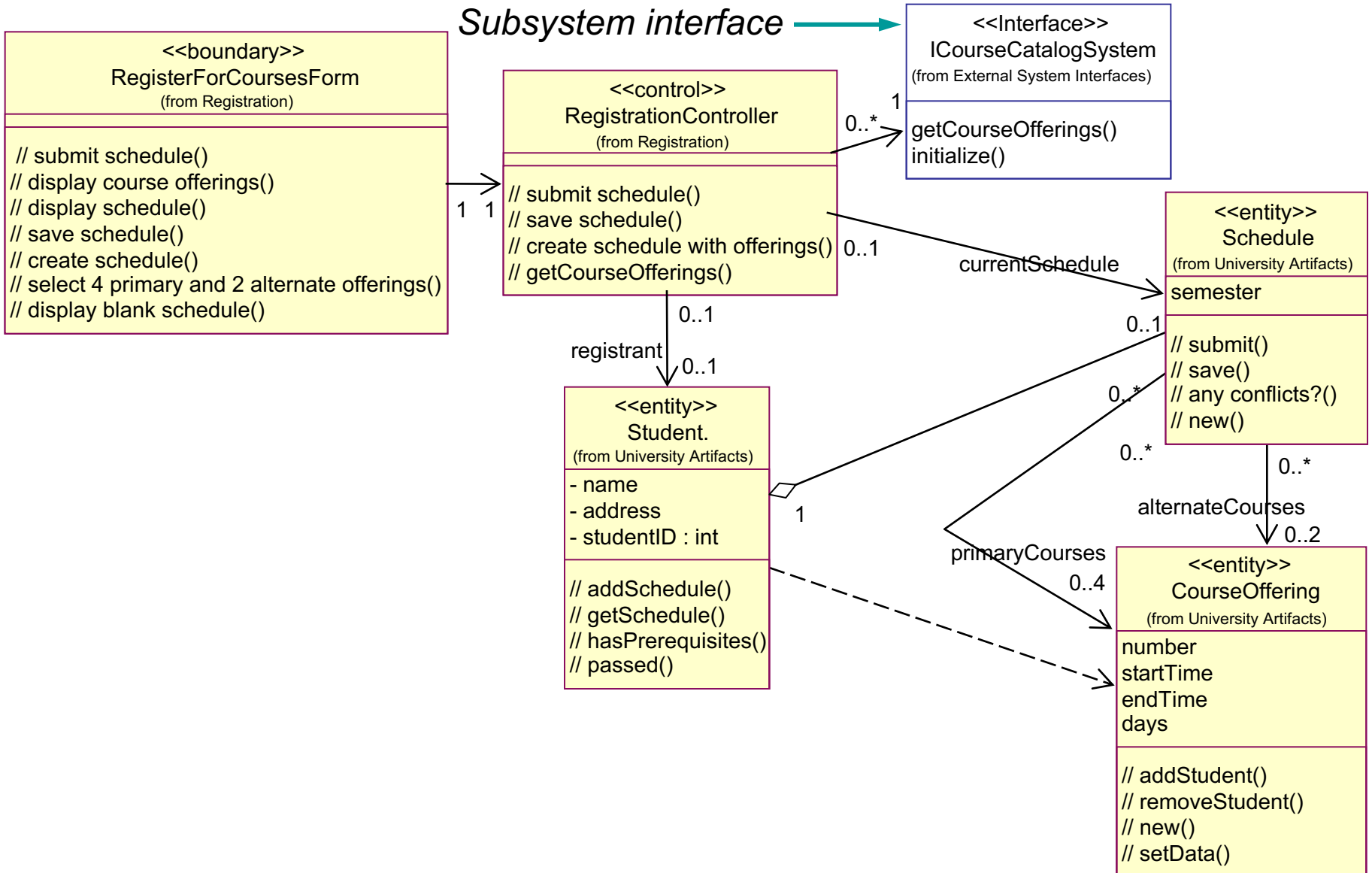


Example: Incorporating Subsystems (After)

Replaced with subsystem interface



Example: Incorporating Subsystem Interfaces (VOPC)



Incorporating Architectural Mechanisms: Security

- Analysis Class to Architectural-Mechanism Map from Use-Case Analysis

Analysis Class	Analysis Mechanism(s)
Student	Persistency, *Security*
Schedule	Persistency, *Security*
CourseOffering	Persistency, Legacy Interface
Course	Persistency, Legacy Interface
RegistrationController	Distribution

Incorporating Architectural Mechanisms: Distribution

- Analysis Class to Architectural-Mechanism Map from Use-Case Analysis

Analysis Class	Analysis Mechanism(s)
Student	Persistency, Security
Schedule	Persistency, Security
CourseOffering	Persistency, Legacy Interface
Course	Persistency, Legacy Interface
RegistrationController	*Distribution*

Review: Incorporating RMI: Steps

1. Provide access to RMI support classes (e.g., √ Remote and Serializable interfaces, Naming Service)
   - √ – *Use java.rmi and java.io package in Middleware layer*
   - √ • For each class to be distributed:
     - *Controllers to be distributed are in Application layer*
     - *Dependency from Application layer to Middleware layer is needed to access java packages*
     - Define interface for class that realizes Remote
     - Have class inherit from UnicastRemoteObject

√ = *Done*

Review: Incorporating RMI: Steps (continued)

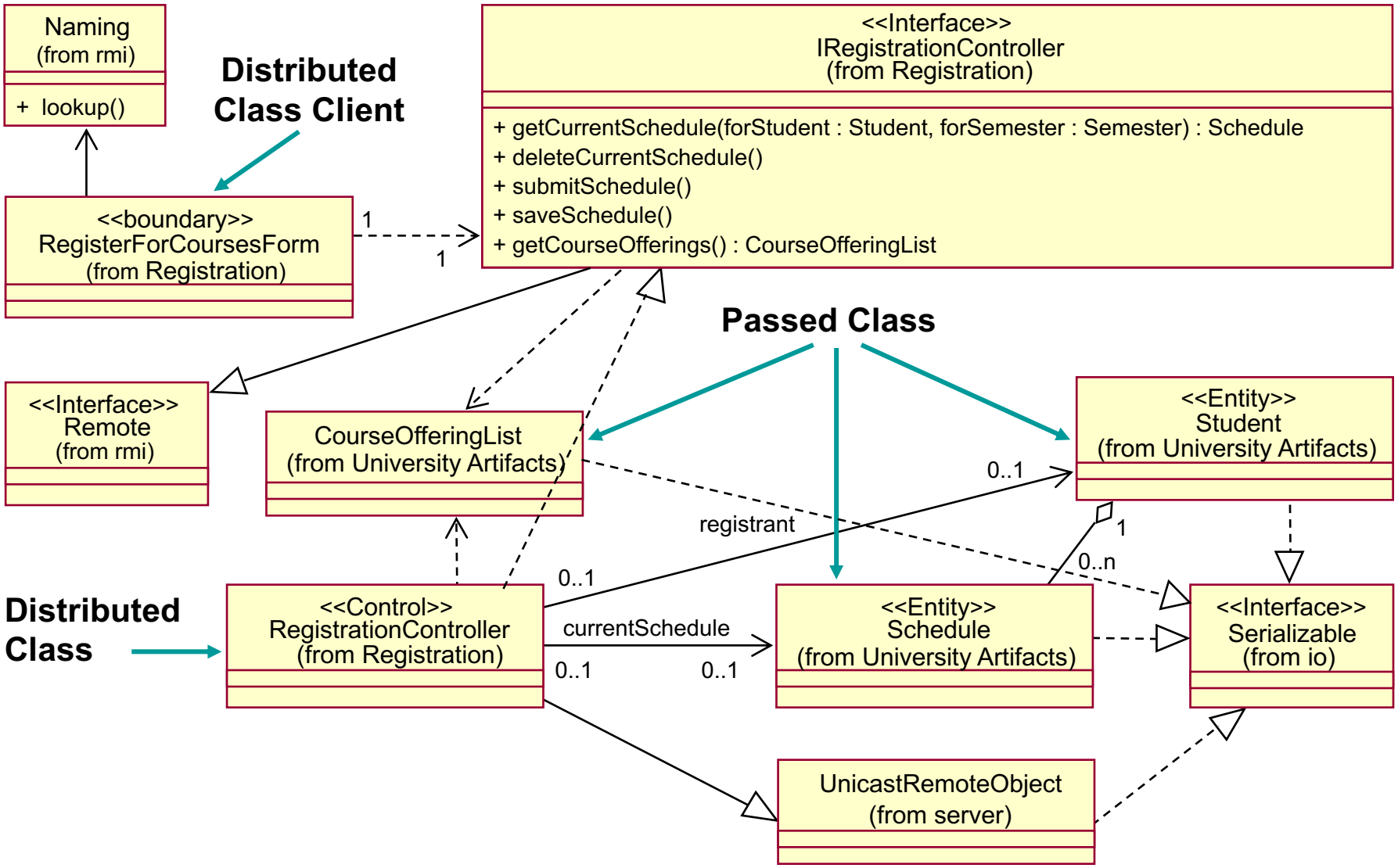
2. Have classes for data passed to
√ distributed objects realize the Serializable interface
√ – *Core data types are in Business Services layer*
– *Dependency from Business Services layer to Middleware layer is needed to get access to java.rmi*
– Add the realization relationships
3. Run pre-processor – out of scope

√ = *Done*

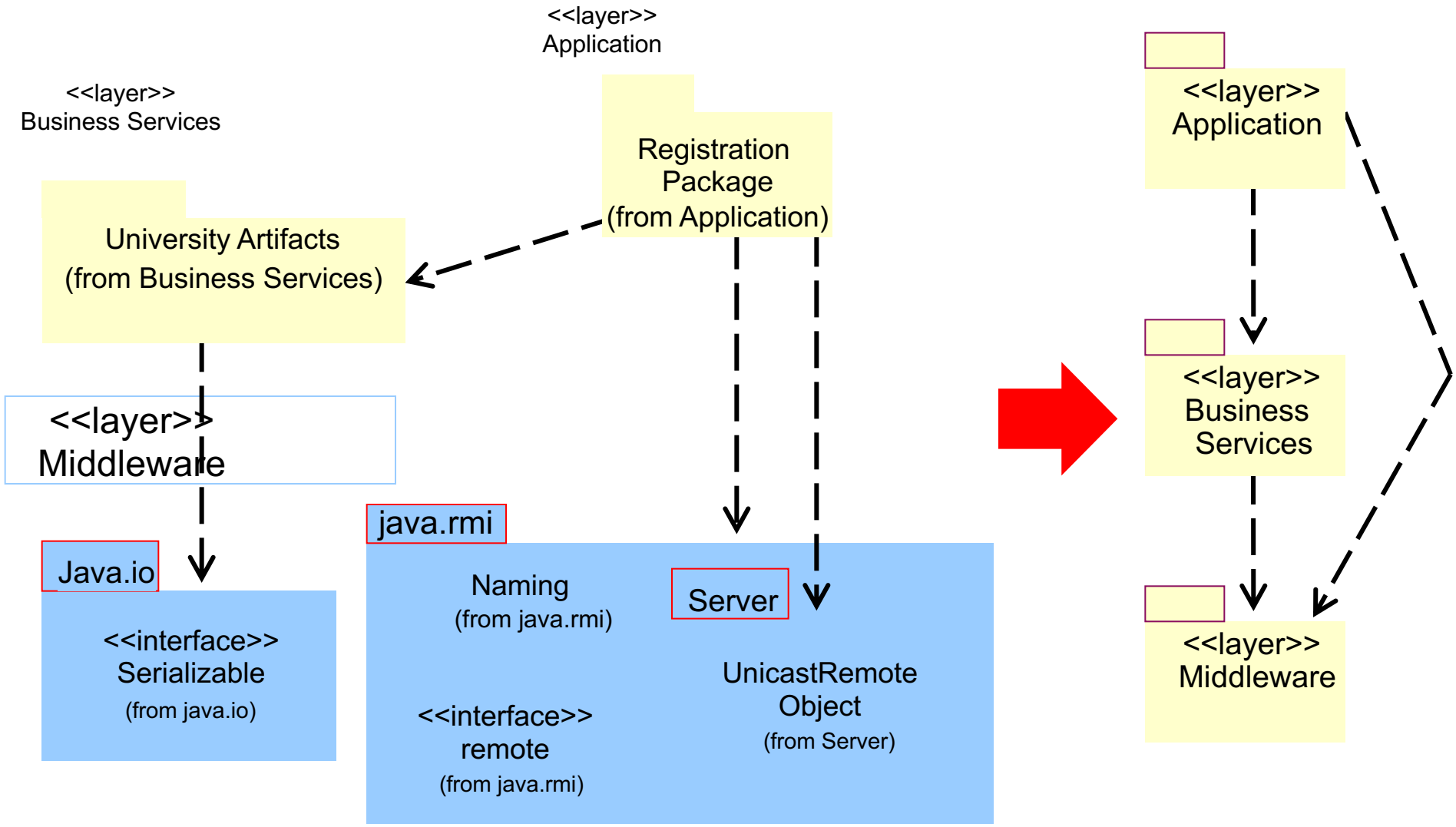
Review: Incorporating RMI: Steps (continued)

4. Have distributed class clients look up the remote objects using the Naming service
   - √ – *Most Distributed Class Clients are forms*
   - √ – *Forms are in Application layer*
   - – *Dependency from Application layer to Middleware layer is needed to get access to java.rmi*
   - – Add relationship from Distributed Class Clients to Naming Service
5. Create/update interaction diagrams with distribution processing (optional)

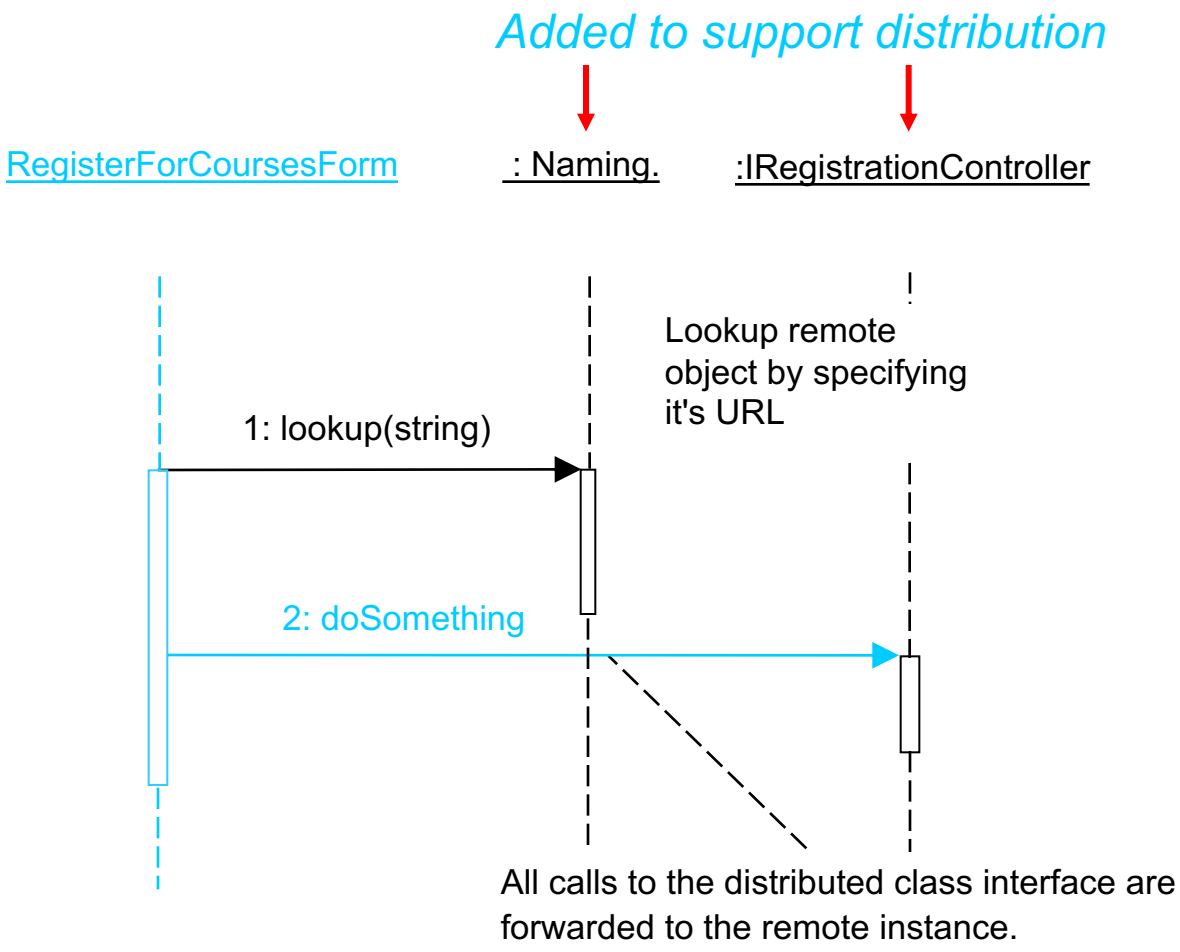
Example: Incorporating RMI



Example: Incorporating RMI (continued)



Example: Incorporating RMI (continued)

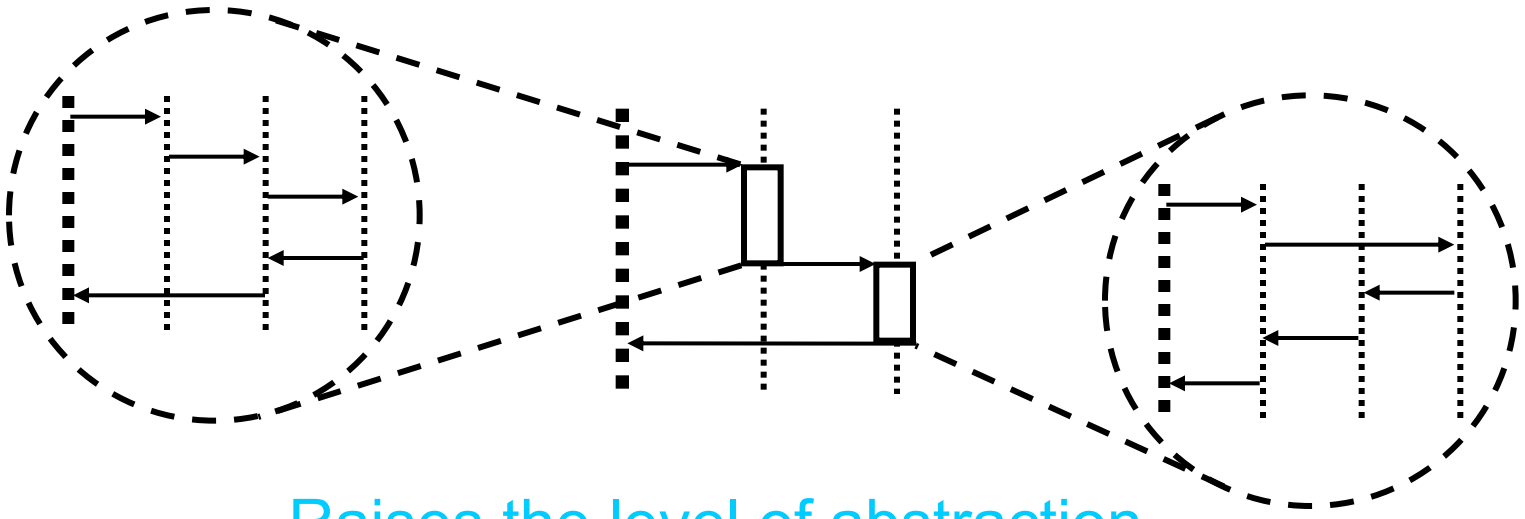


Use-Case Design Steps

- Describe interaction among design objects
- Simplify sequence diagrams using subsystems
- Describe persistence-related behavior
- Refine the flow of events description
- Unify classes and subsystems

Encapsulating Subsystem Interactions

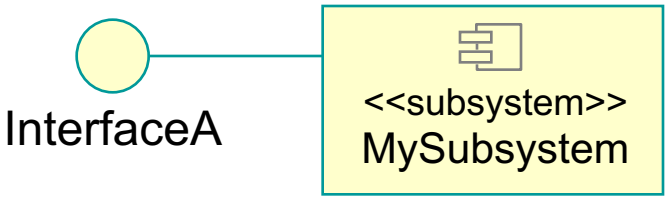
- Interactions can be described at several levels
- Subsystem interactions can be described in their own interaction diagrams

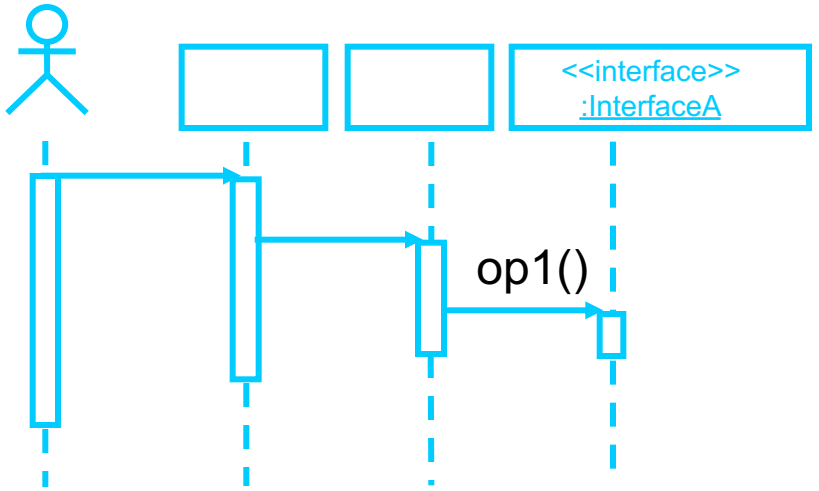


Raises the level of abstraction.

Guidelines: Encapsulating Subsystem Interactions

- Subsystems should be represented by their interfaces on interaction diagrams
- Messages to subsystems are modeled as messages to the subsystem interface
- Messages to subsystems correspond to operations of the subsystem interface
- Interactions within subsystems are modeled in Subsystem Design





Advantages of Encapsulating Subsystem Interactions

Use-case realizations:

- Are less cluttered
- Can be created before the internal designs of subsystems are created (parallel development)
- Are more generic and easier to change (Subsystems can be substituted.)

Parallel Subsystem Development

- Concentrate on requirements that affect subsystem interfaces
- Outline required interfaces
- Model messages that cross subsystem boundaries
- Draw interaction diagrams in terms of subsystem interfaces for each use case
- Refine the interfaces needed to provide messages
- Develop each subsystem in parallel

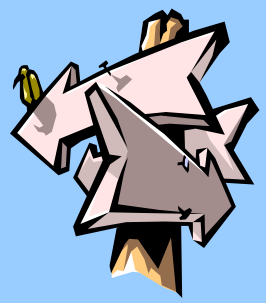
Use-Case Design Steps

- Describe interaction among design objects
- Simplify sequence diagrams using subsystems
- Describe persistence-related behavior
- Refine the flow of events description
- Unify classes and subsystems

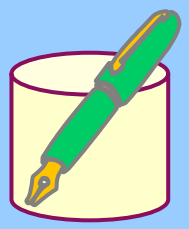
Use-Case Design Steps: Describe Persistence-Related Behavior

- Describe Persistence-Related Behavior

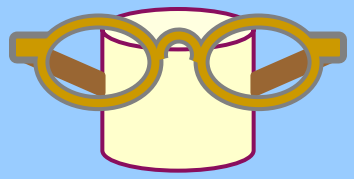
Modeling Transactions



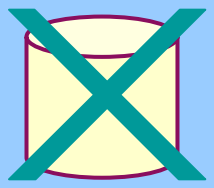
Writing Persistent Objects



Reading Persistent Objects



Deleting Persistent Objects



Incorporating the Architectural Mechanisms: Persistency

- Analysis-Class-to-Architectural-Mechanism Map from Use-Case Analysis

Analysis Class	Analysis Mechanism(s)
Student	*Persistency*, Security
Schedule	*Persistency*, Security
CourseOffering	Persistency, Legacy Interface
Course	Persistency, Legacy Interface
RegistrationController	Distribution

OODBMS Persistency

RDBMS Persistency

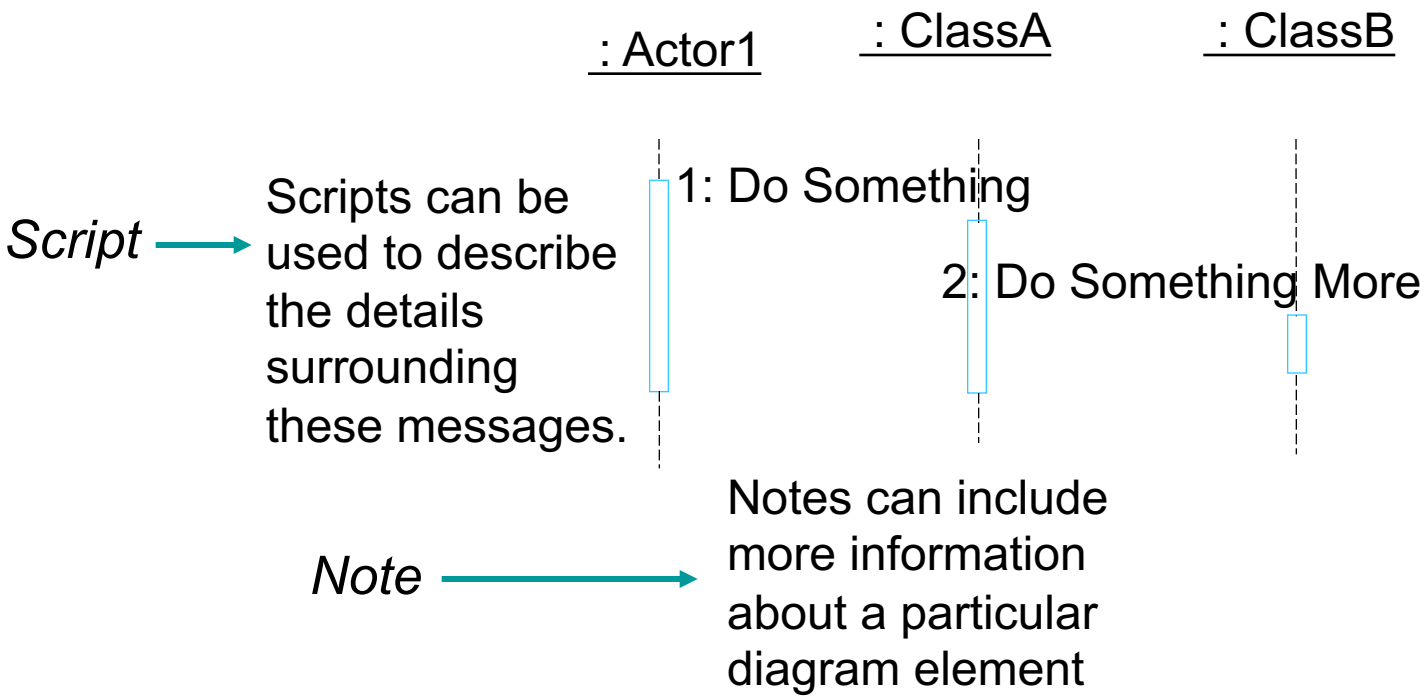
Legacy persistency (RDBMS) is deferred to Subsystem Design.

Use-Case Design Steps

- Describe interaction among design objects
- Simplify sequence diagrams using subsystems
- Describe persistence-related behavior
- Refine the flow of events description
- Unify classes and subsystems

Detailed Flow of Events Description Options

- Annotate the interaction diagrams

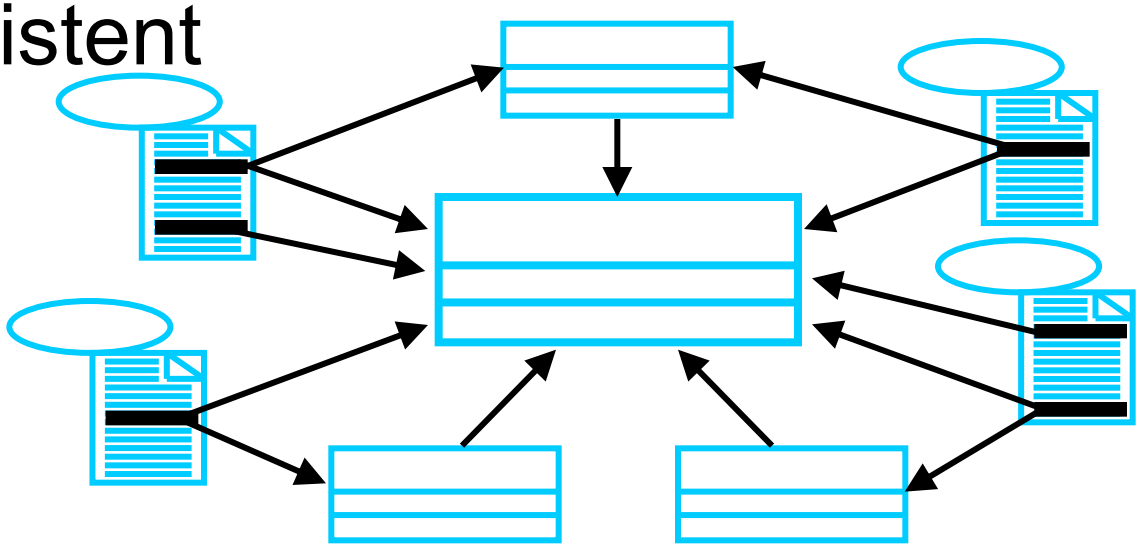


Use-Case Design Steps

- Describe interaction among design objects
- Simplify sequence diagrams using subsystems
- Describe persistence-related behavior
- Refine the flow of events description
- Unify classes and subsystems

Design Model Unification Considerations

- Model element names should describe their function
- Merge similar model elements
- Use inheritance to abstract model elements
- Keep model elements and flows of events consistent



Checkpoints: Use-Case Design

- Is package/subsystem partitioning logical and consistent?
- Are the names of the packages/subsystems descriptive?
- Do the public package classes and subsystem interfaces provide a single, logically consistent set of services?
- Do the package/subsystem dependencies correspond to the relationships between the contained classes?
- Do the classes contained in a package belong there according to the criteria for the package division?
- Are there classes or collaborations of classes that can be separated into an independent package/subsystem?

Checkpoints: Use-Case Design (continued)

- Have all the main and/or subflow for this iteration been handled?
- Has all behavior been distributed among the participating design elements?
- Has behavior been distributed to the right design elements?
- If there are several interaction diagrams for the use-case realization, is it easy to understand which Communication diagrams relate to which flow of events?

Review: Use-Case Design

- What is the purpose of Use-Case Design?
- What is meant by encapsulating subsystem interactions? Why is it a good thing to do?