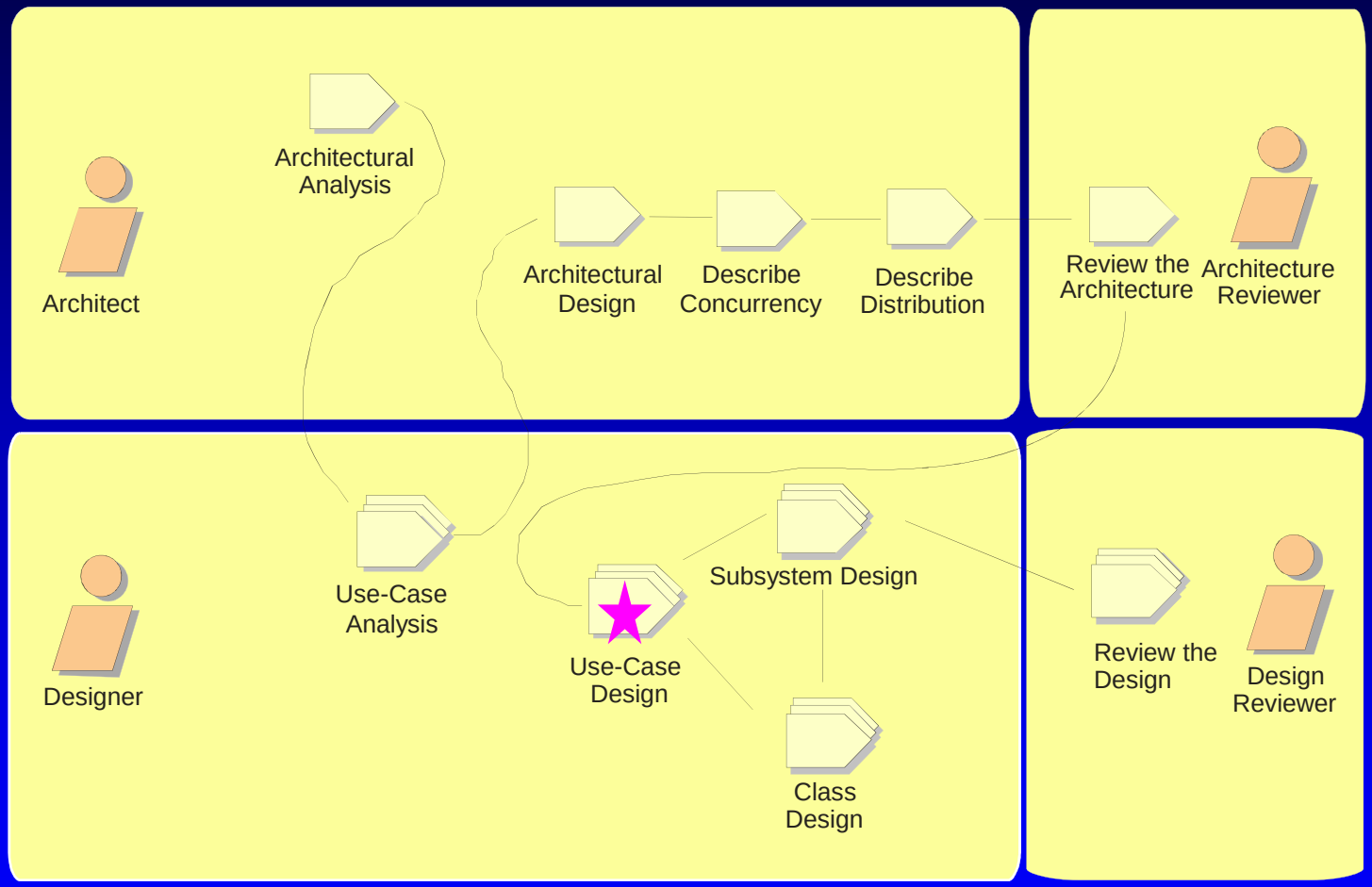
Use Case Design

FROM Dr. Giuseppe Calavaro, Ratiolal®
TO Students in the DISP, University of Roma “Tor Vergata”
2003

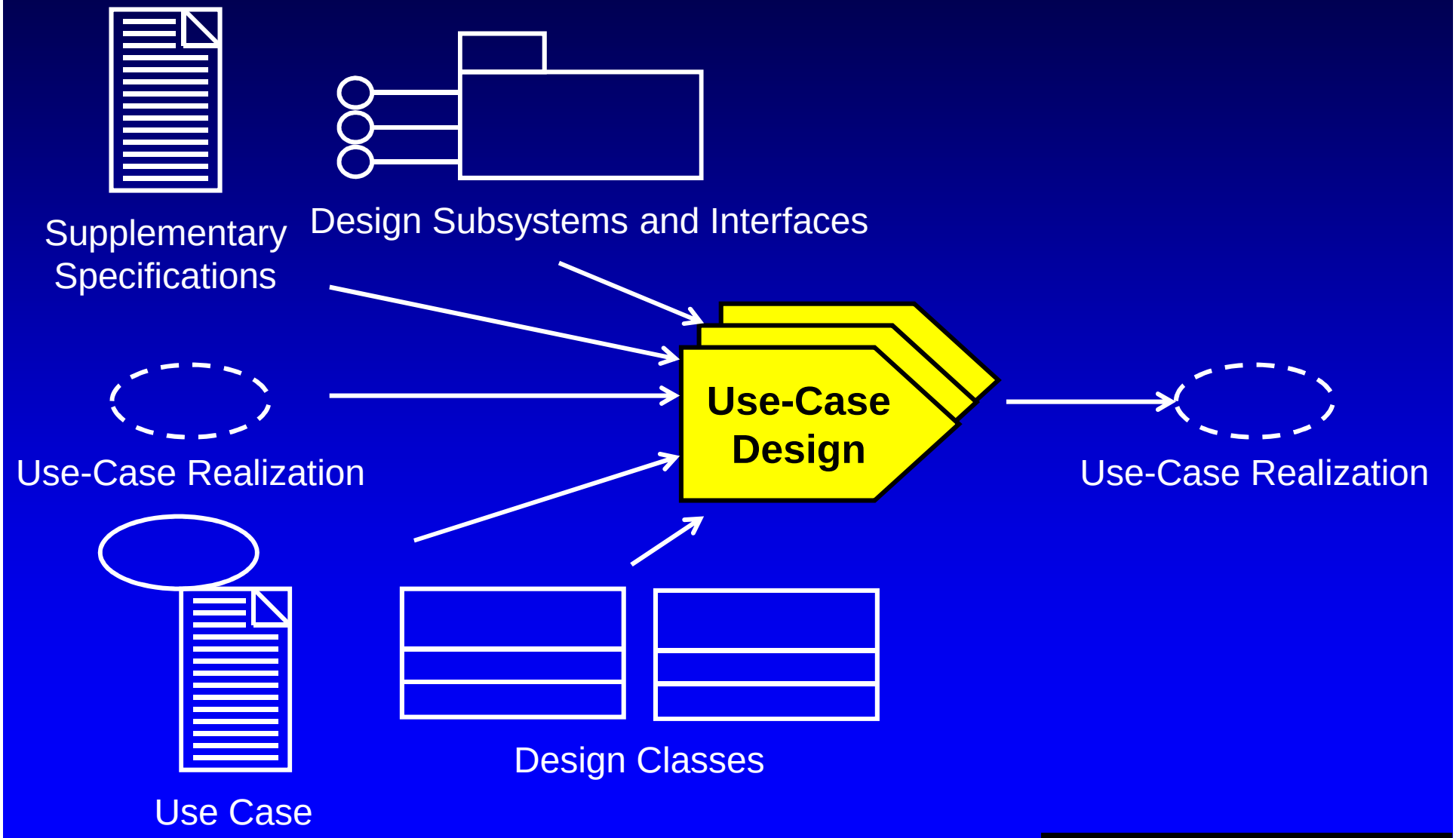
Objectives: Use-Case Design

- Understand the purpose of Use-Case Design and where 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 Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

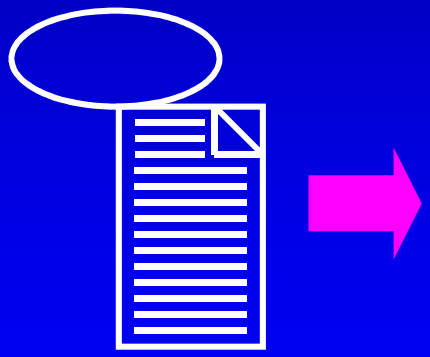
Review: Use-Case Realization

Use-Case Model

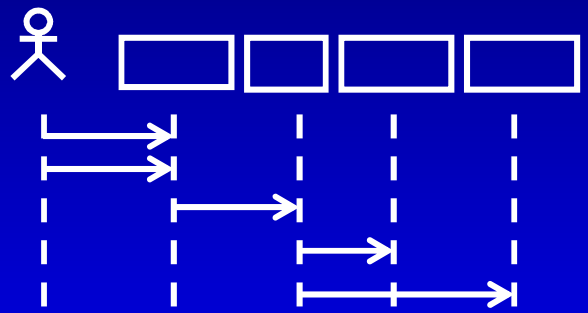
Design Model

Use Case

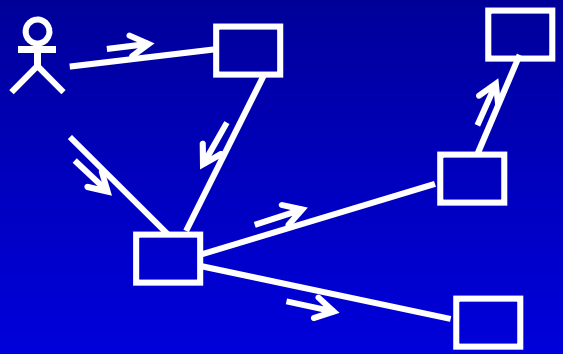
Use-Case Realization



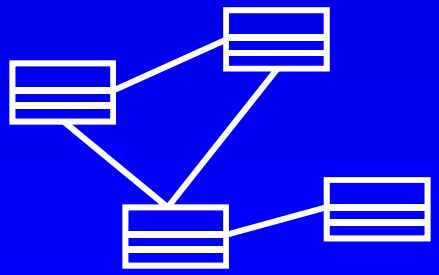
Use Case



Sequence Diagrams



Collaboration Diagrams



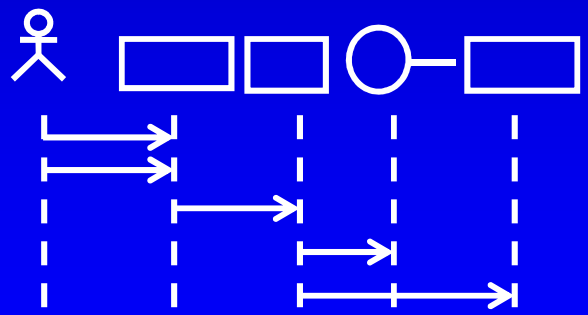
Class Diagrams

Use-Case Design Steps

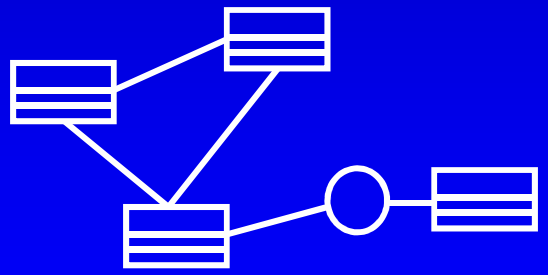
- Describe Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

Use-Case Realization Refinement

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



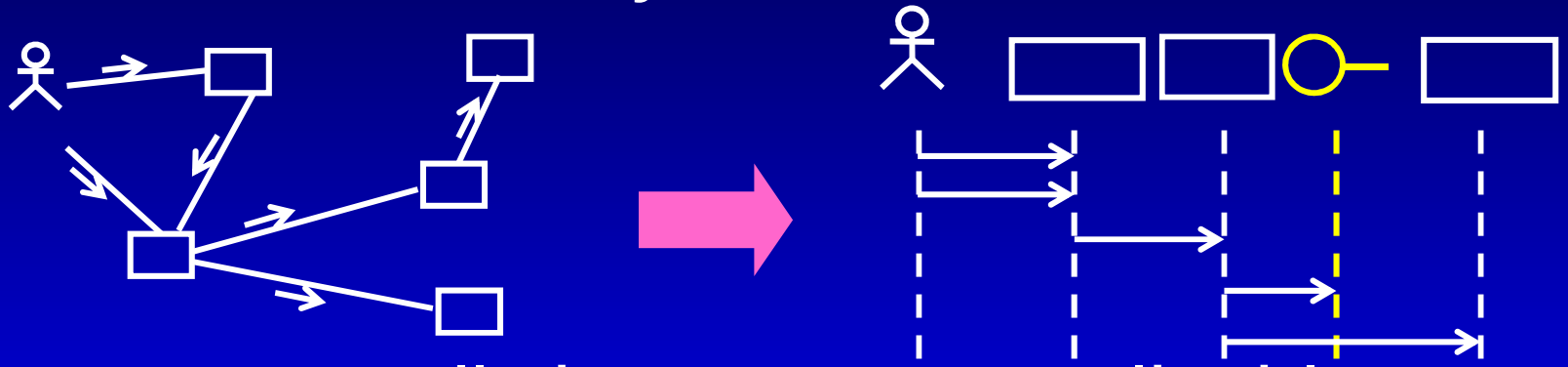
Sequence Diagrams



Class Diagrams

Use-Case Realization Refinement Steps

- Replace applicable classes with the associated subsystem interfaces



- Incrementally incorporate applicable architectural mechanisms
- Update use-case realization
  - Interaction diagrams
  - View of participating classes (VOPC) class diagram(s)

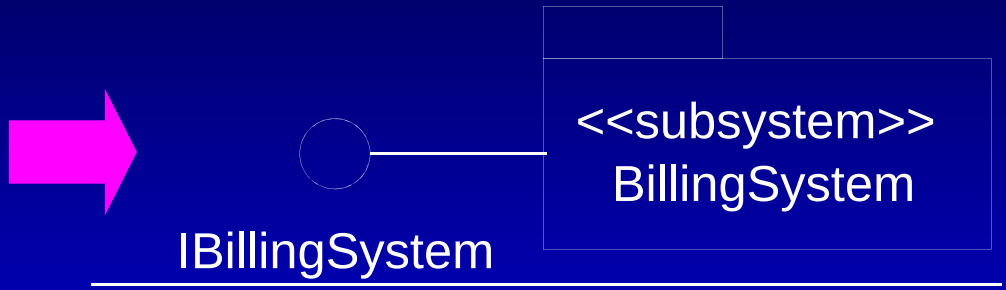
Example: Incorporating Subsystem Interfaces

Analysis Classes

<<boundary>>
BillingSystem

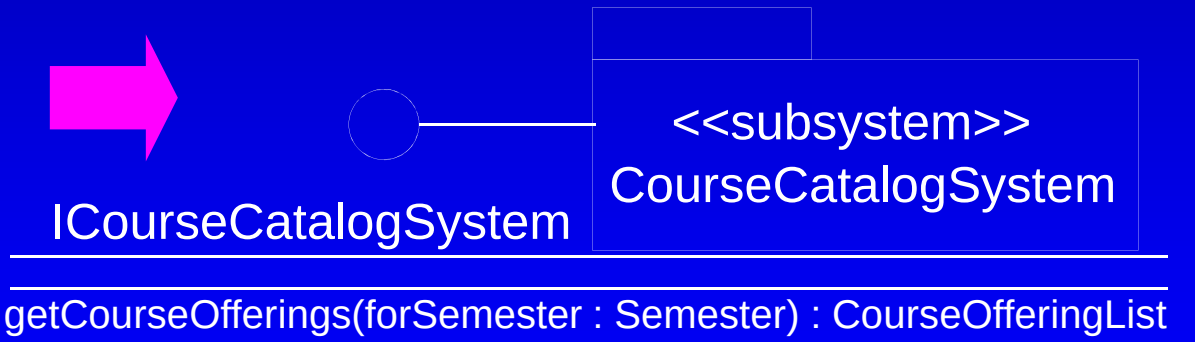
// submit bill()

Design Elements



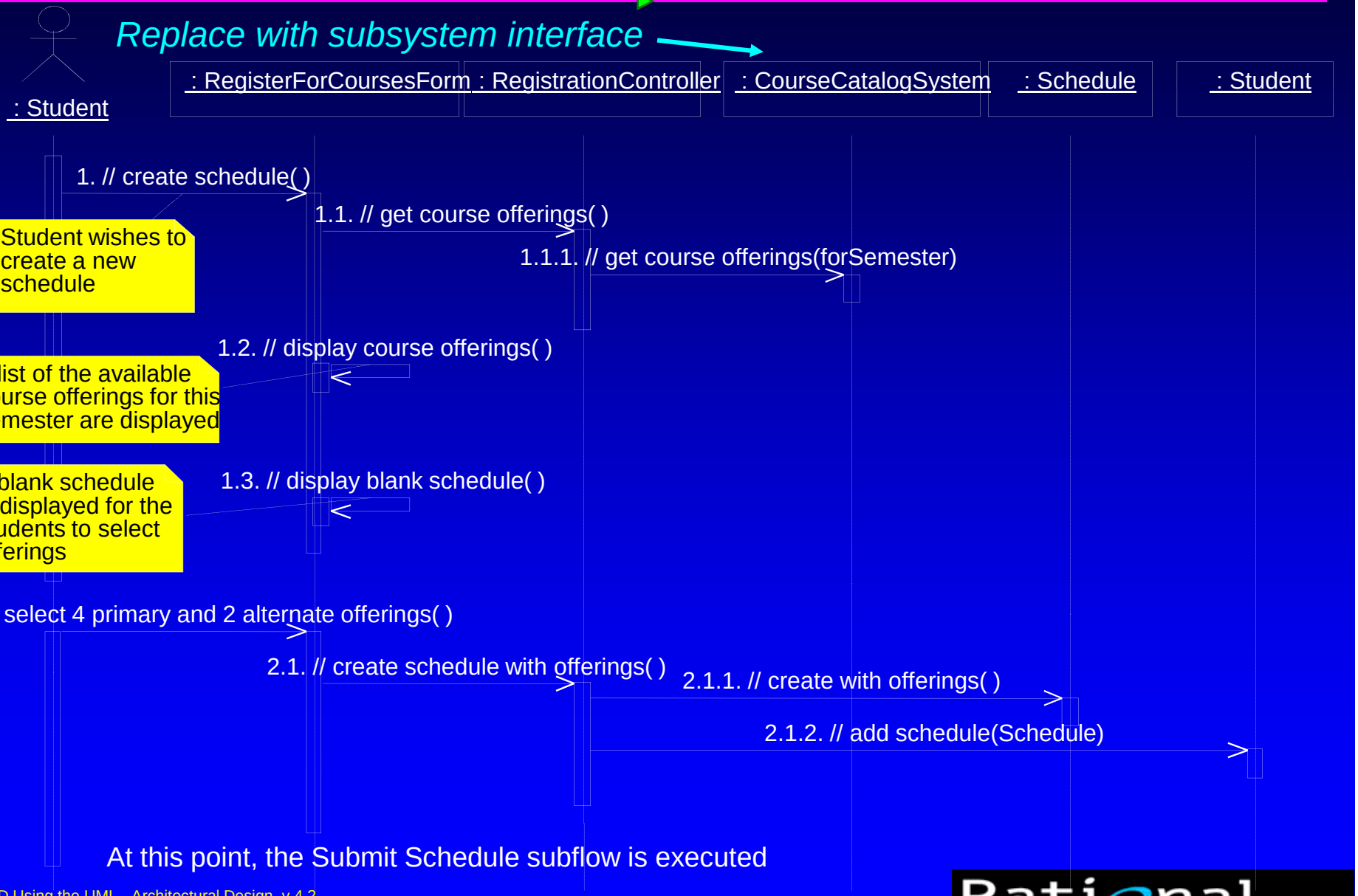
<<boundary>>
CourseCatalogSystem

// get course offerings()



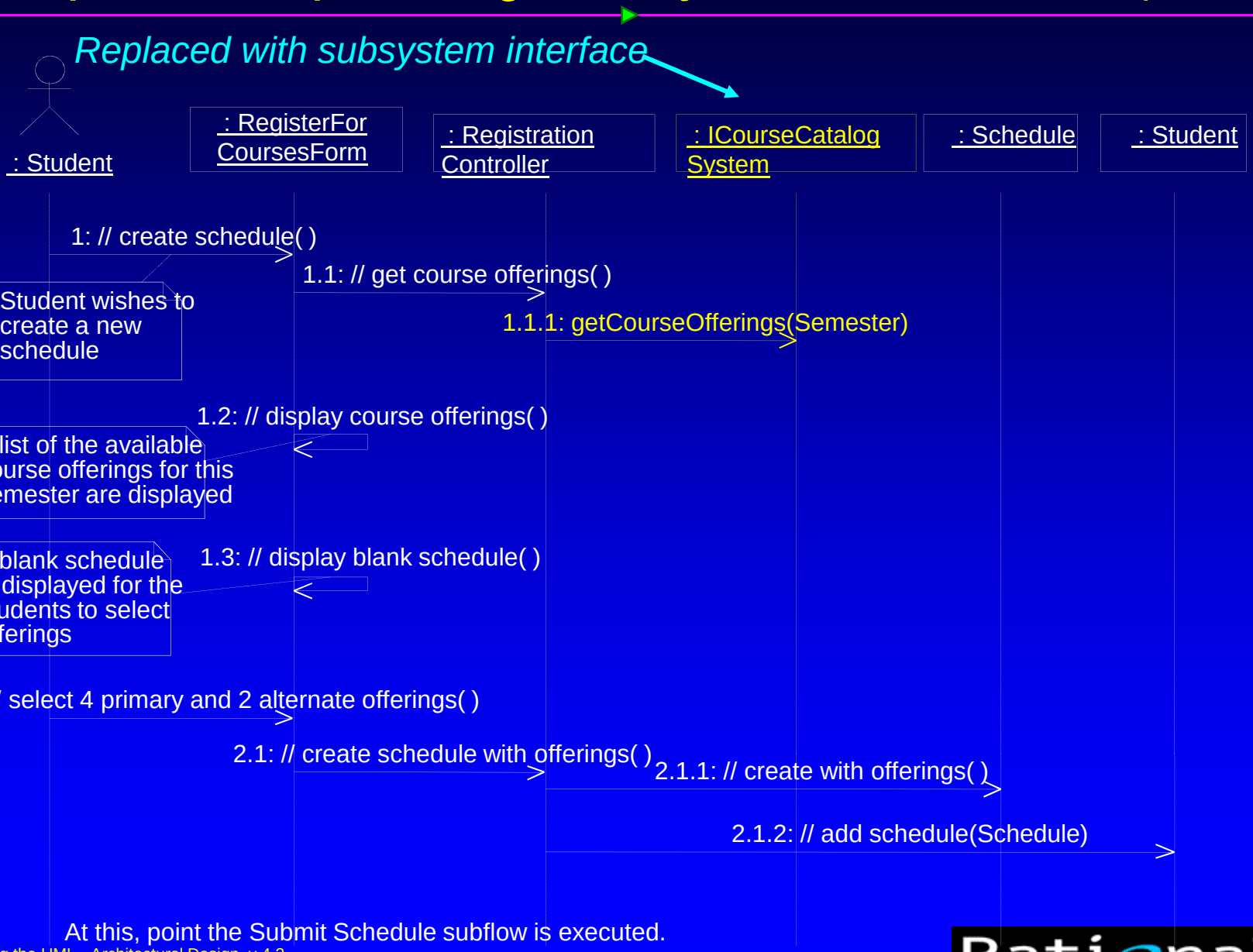
All other analysis classes mapped directly to design classes

Example: Incorporating Subsystem Interfaces (Before)

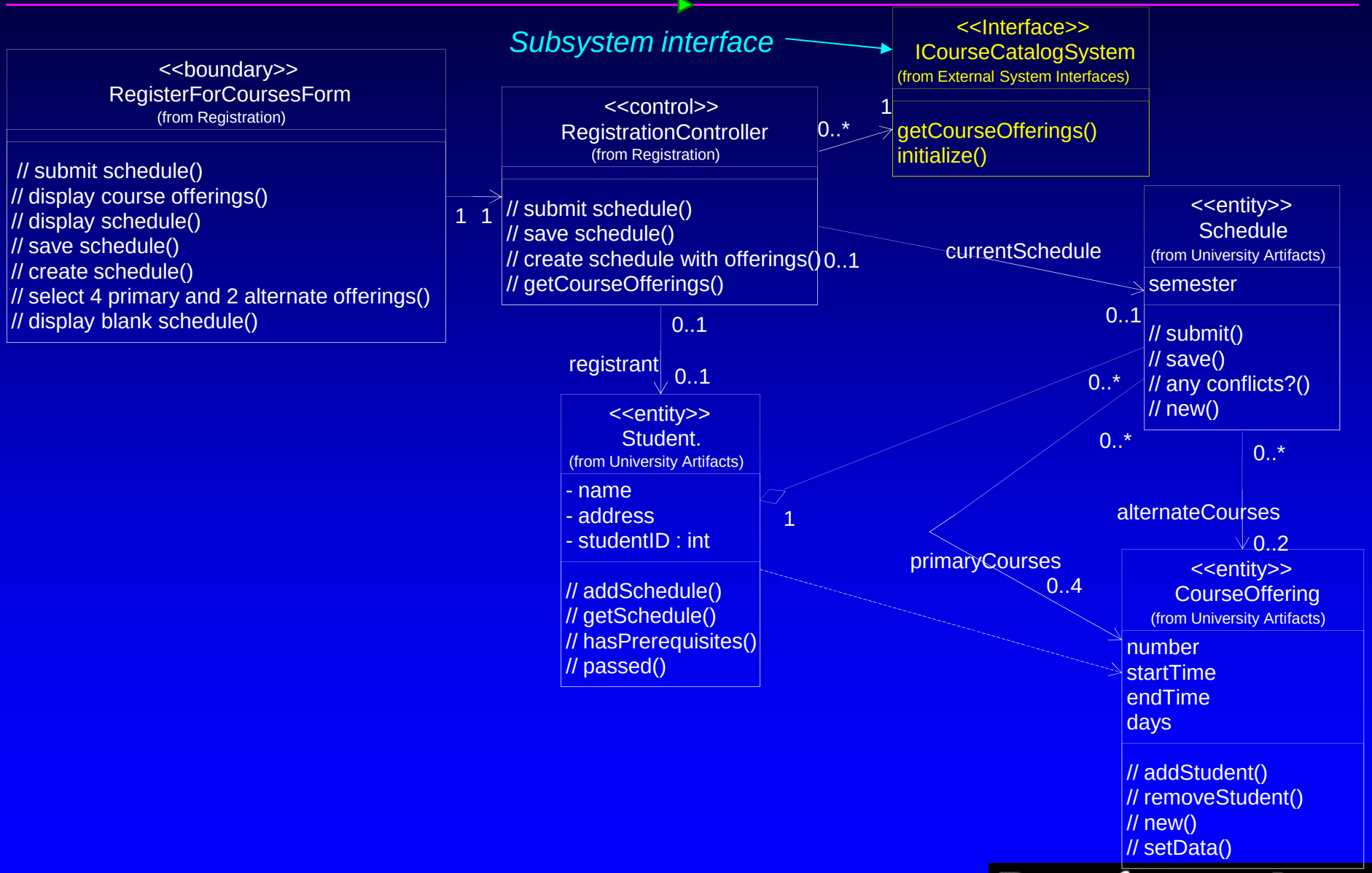


At this point, the Submit Schedule subflow is executed

Example: Incorporating Subsystem Interfaces (After)



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

The details of incorporating the security mechanism are provided in the Additional Information Appendix in the Security Mechanism section.

Details in Appendix

Incorporating Architectural Mechanisms: Distribution

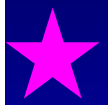
- Analysis-Class-to-Architectural-Mechanism Map from Use-Case Analysis

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

The details of incorporating the distribution (RMI) mechanism are provided in the Additional Information Appendix in the RMI Mechanism section.

Details in Appendix

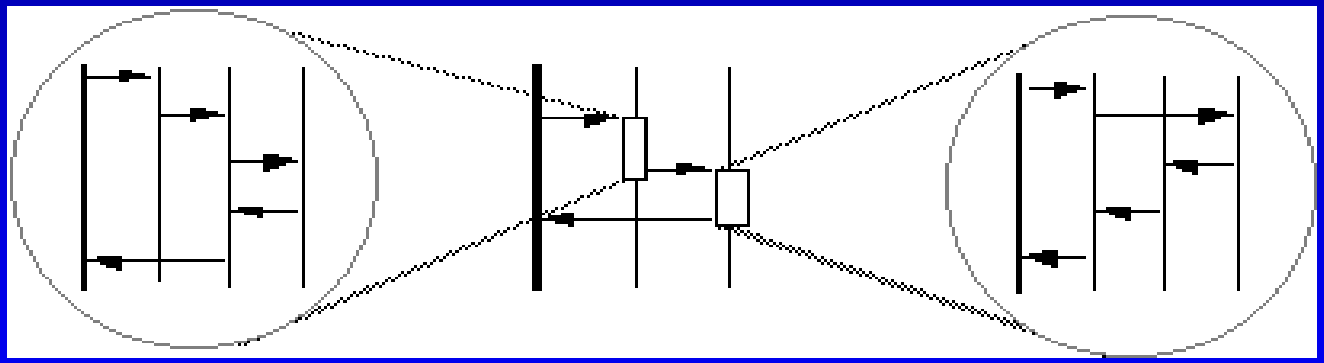
Use-Case Design Steps



- Describe Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

Encapsulating Subsystem Interactions

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



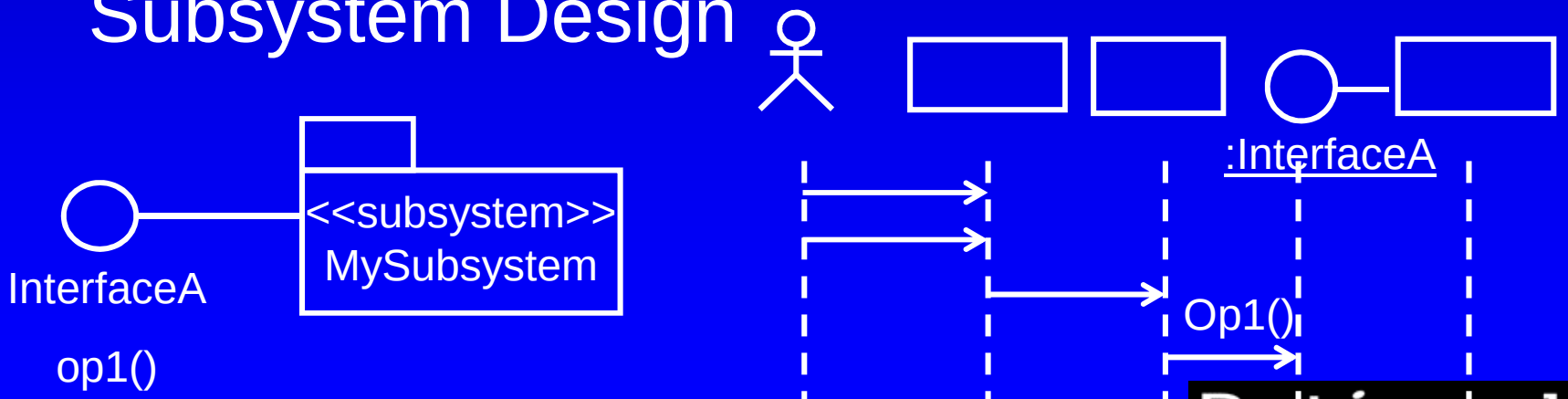
Raises the level of abstraction

When to Encapsulate Sub-Flows in a Subsystem

- Sub-flow occurs in multiple use-case realizations
- Sub-flow has reuse potential
- Sub-flow is complex and easily encapsulated
- Sub-flow is responsibility of one person/team
- Sub-flow produces a well-defined result
- Sub-flow is encapsulated within a single Implementation Model component

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 modeled in Subsystem Design



Advantages of Encapsulating Subsystem Interactions

- Use-case realizations are less cluttered
- Use-case realizations can be created before the internal designs of subsystems are created (parallel development)
- Use-case realizations are more generic and easy 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 subsystem interfaces as synchronization points

Use-Case Design Steps

- Describe Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

Use-Case Design Steps: Describe Persistence-related Behavior

- Describe Persistence-related Behavior
  - Modeling Transactions
  - Writing Persistent Objects
  - Reading Persistent Objects
  - Deleting Persistent Objects

Modeling Transactions

- What is a Transaction?
  - Atomic operation invocations
  - “All or nothing”
  - Provide consistency
- Modeling Options
  - Textually (scripts)
  - Explicit messages
- Error conditions
  - May require separate interaction diagrams
  - Rollback
  - Failure modes

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) deferred to Subsystem Design

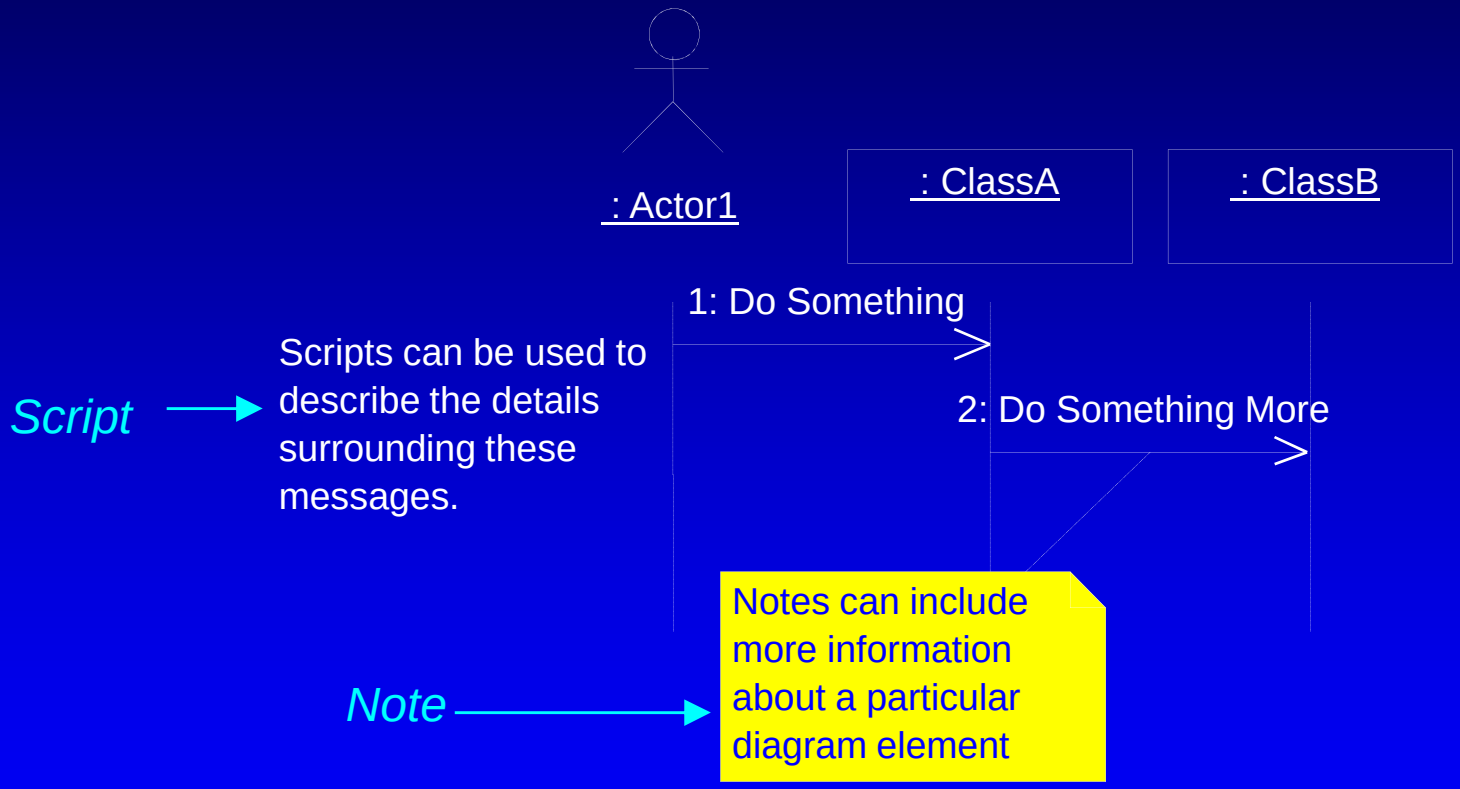
Details in Appendix

Use-Case Design Steps

- Describe Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

Detailed Flow of Events Description Options

- Annotate the interaction diagrams

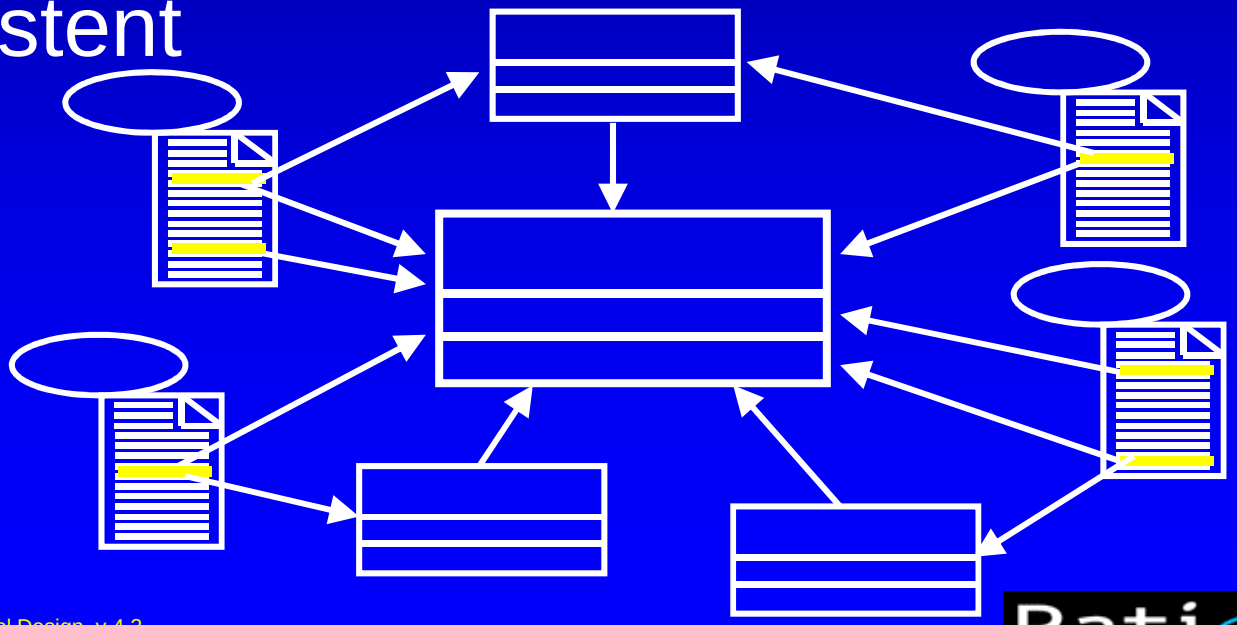


Use-Case Design Steps

- Describe Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

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



Use-Case Design Steps

- Describe Interactions Between Design Objects
- Simplify Interaction Diagrams Using Subsystems (optional)
- Describe Persistence-Related Behavior
- Refine the Flow of Events Description
- Unify Classes and Subsystems
- Checkpoints

Checkpoints: Design Model

- 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 which can be separated into an independent package/subsystem?
- Is the ratio between the number of packages/subsystems and the number of classes appropriate?

Checkpoints: Use-Case Realizations

- Have all the main and/or sub-flows 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 collaboration 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?

Exercise: Use-Case Design, Part 1

- Given the following:
  - Analysis use-case realizations (VOPCs and interaction diagrams)
  - The analysis-class-to-design-element map
  - *The analysis-class-to-analysis-mechanism map*
  - *Analysis-to-design-mechanism map*
  - *Patterns of use for the architectural mechanisms*

(continued)

Exercise: Use-Case Design, Part 1 (cont.)

- Identify the following for a particular use case:
  - The design elements that replaced the analysis classes in the analysis use-case realizations
  - *The architectural mechanisms that affect the use-case realizations*
  - The design element collaborations needed to implement the use case
  - The relationships between the design elements needed to support the collaborations

(continued)

Exercise: Use-Case Design, Part 1 (cont.)

- Produce the following for a particular use case:
  - Design use-case realization
    - Interaction diagram(s) per use-case flow of events that describes the DESIGN ELEMENT collaborations required to implement the use case
    - Class diagram (VOPC) that includes the DESIGN ELEMENTS that must collaborate to perform the use case, and their relationships

Exercise: Use-Case Design, Part 2 (optional)

- Given the following:
  - The architectural layers, their packages, and their dependencies
  - All design use-case realization VOPCs (design elements, their packages, and their relationships)

(continued)

Exercise: Use-Case Design, Part 2 (optional) (cont.)

- Identify the following:
  - Any updates to the package relationships needed to support the class relationships
- Produce the following diagrams:
  - Refined class diagram that contains all packages and their dependencies (organized by layer)