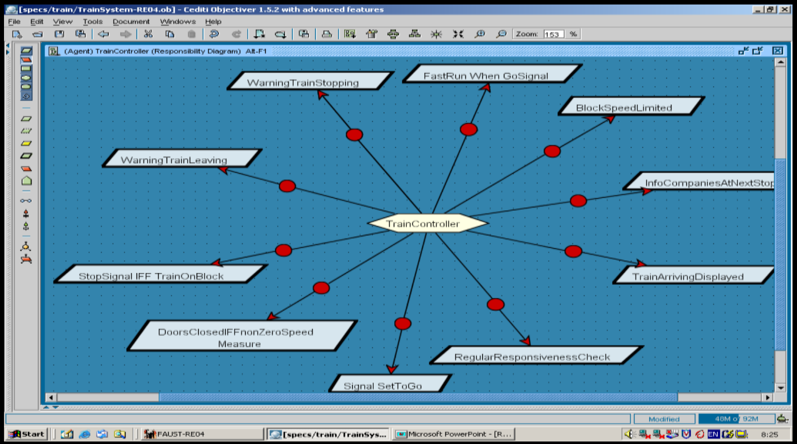
Building models for RE – Chapter 13

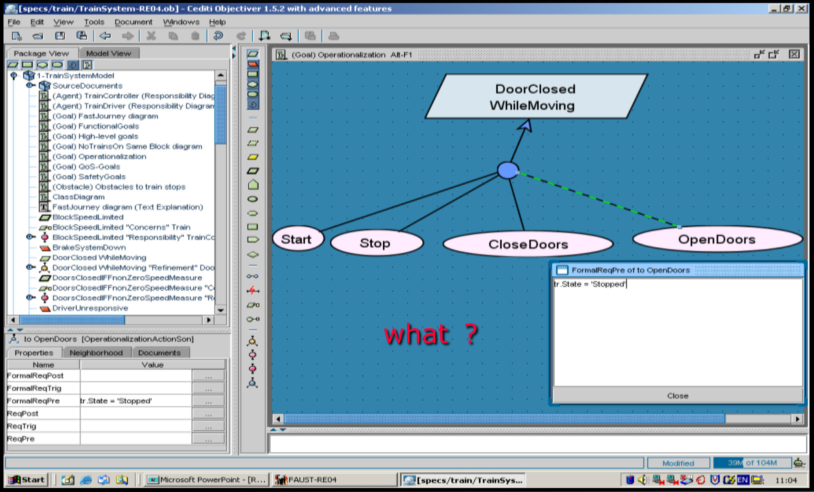
Chap.11: Agents & responsibilities



[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features

(Agent) TrainController (Responsibility Diagram) Alt-F1

Chap.12: Operations



[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features

- 1-TrainSystemModel
  - SourceDocuments
    - (Agent) TrainController (Responsibility Diag
    - (Agent) TrainDriver (Responsibility Diagram
    - (Goal) FastJourney diagram
    - (Goal) FunctionalGoals
    - (Goal) High-level goals
    - (Goal) NoTrainsOn Same Block diagram
    - (Goal) Operationalization
    - (Goal) QoS-Goals
    - (Goal) SafetyGoals
    - (Obstacle) Obstacles to train stops
    - ClassDiagram
    - FastJourney diagram (Text Explanation)
  - BlockSpeedLimited
  - BlockSpeedLimited "Concerns" Train
  - BlockSpeedLimited "Responsibility" TrainCo
  - BrakeSystemDown
  - DoorClosed WhileMoving
  - DoorClosed WhileMoving "Refinement" Doo
  - DoorsClosedIFFnonZeroSpeedMeasure
  - DoorsClosedIFFnonZeroSpeedMeasure "C
  - DoorsClosedIFFnonZeroSpeedMeasure "R
  - DriverUnresponsive

to OpenDoors [OperationalizationActionSon]

Name	Value
FormalReqPost	
FormalReqTrig	
FormalReqPre	tr.State = 'Stopped'
ReqPost	
ReqTrig	
ReqPre	

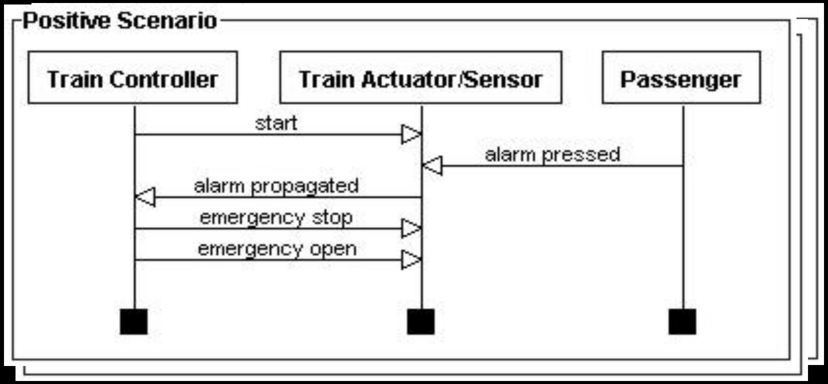
(Goal) Operationalization Alt-F1

FormalReqPre of to OpenDoors

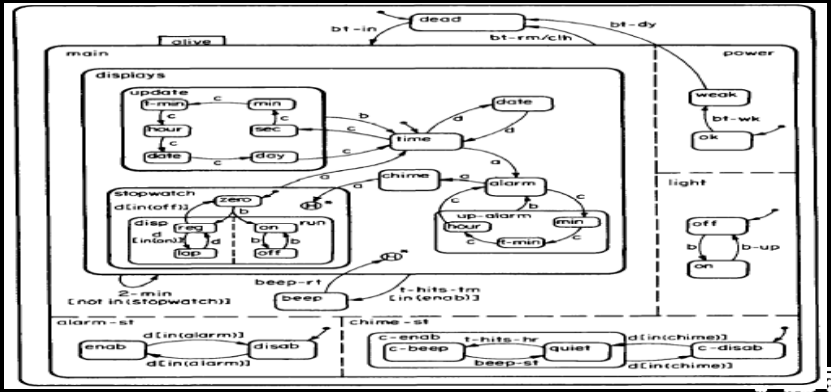
tr.State = 'Stopped'

what ?

Chap.13: Behaviors - *Scenarios*



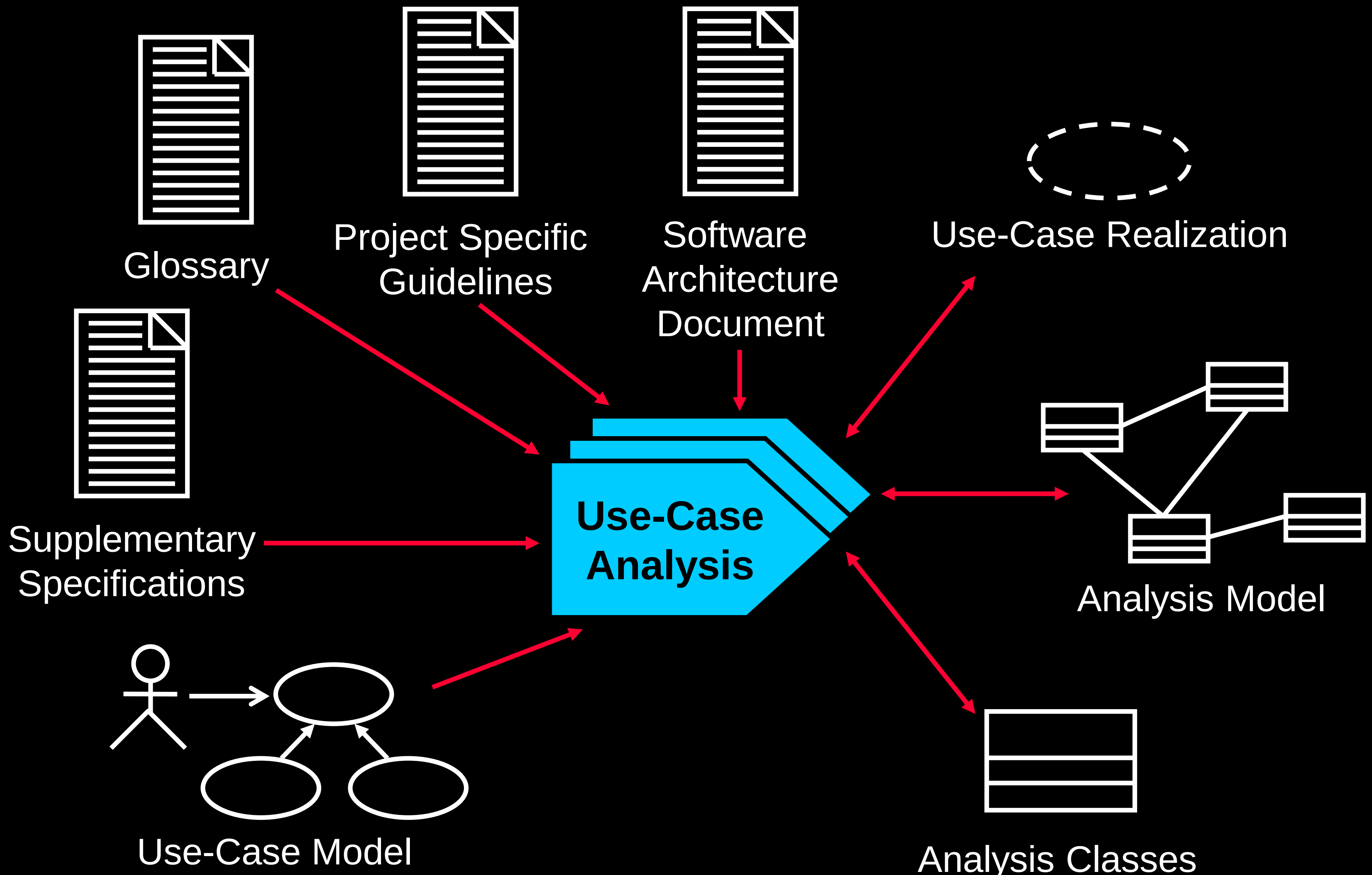
Chap.13: Behaviors - *State machines*



How to build Sequence Diagram - Steps

1. Use case analysis based on specification of use case
   - Determine analysis class
   - Determine behavior for every object
2. Draw sequence

Use-Case Analysis Overview



Glossary

Project Specific Guidelines

Software Architecture Document

Use-Case Realization

Supplementary Specifications

Use-Case Analysis

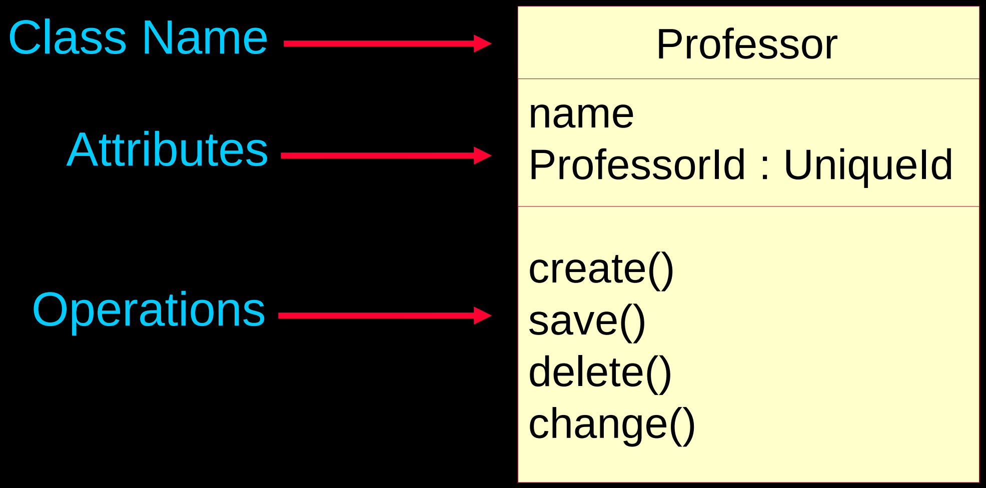
Analysis Model

Use-Case Model

Analysis Classes

Review: Class

- An abstraction
- Describes a group of objects with common:
  - Properties (attributes)
  - Behavior (operations)
  - Relationships
  - Semantics



Class Name

Attributes

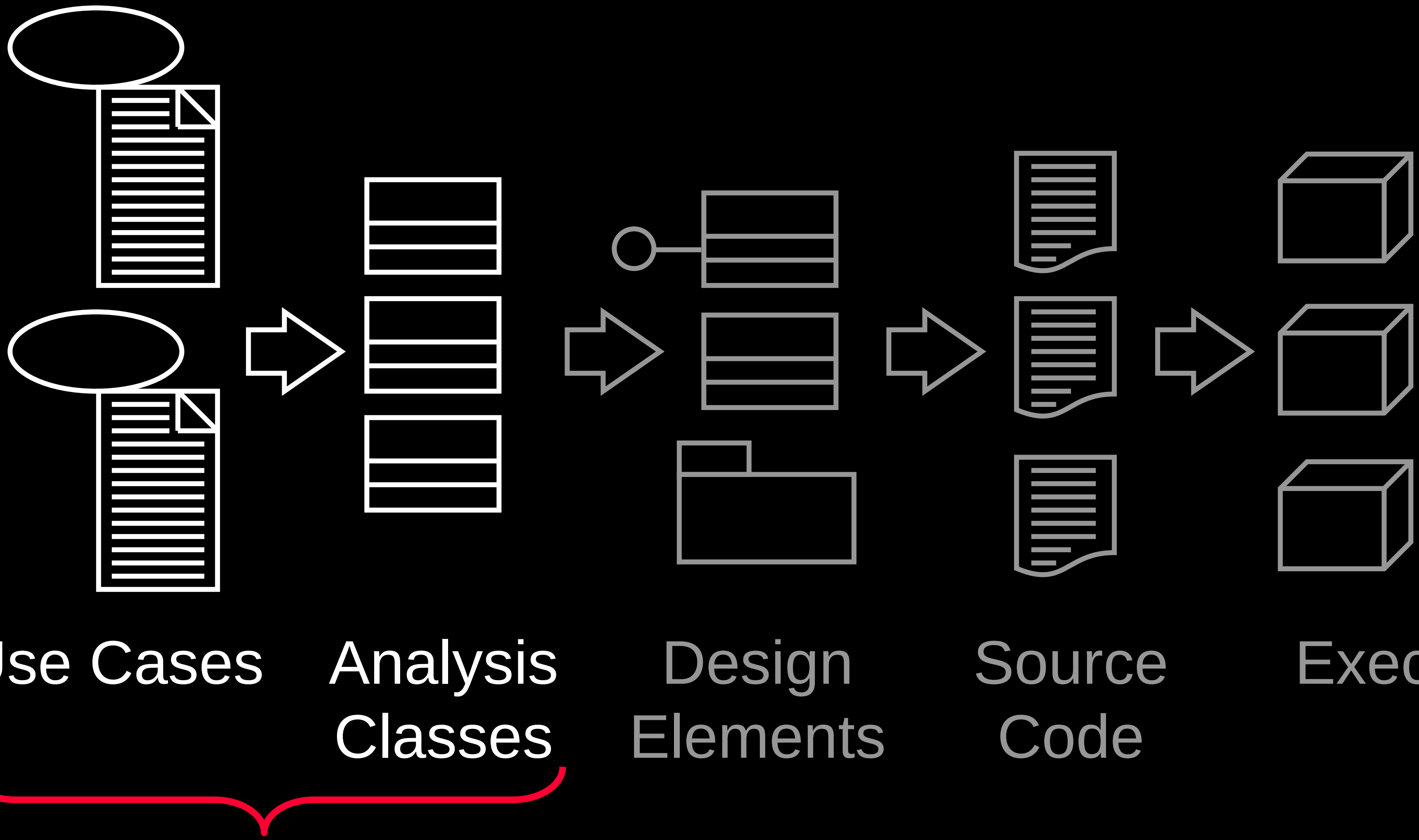
Operations

Professor

name
ProfessorId : UniqueId

create()
save()
delete()
change()

Analysis Classes: A First Step Toward Executables



Use Cases

Analysis Classes

Design Elements

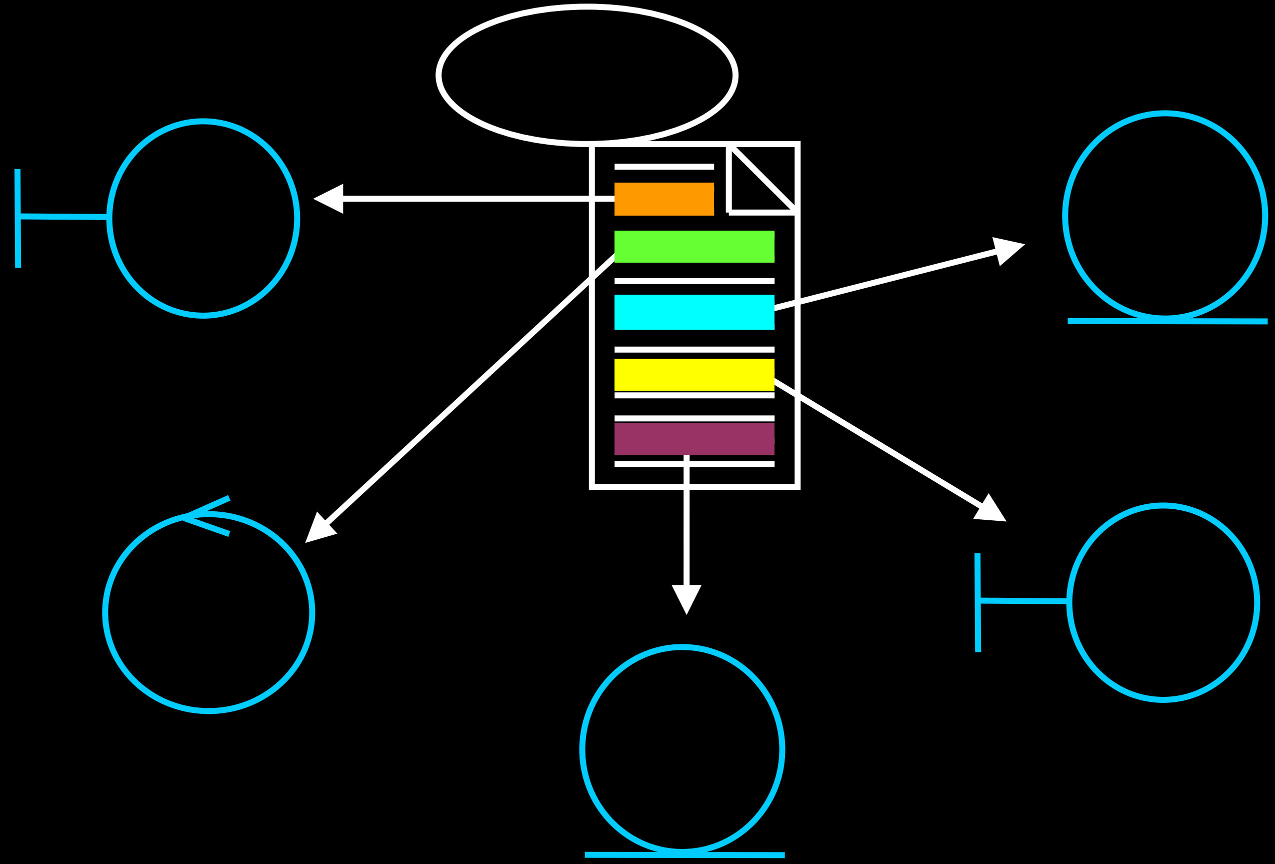
Source Code

Exec

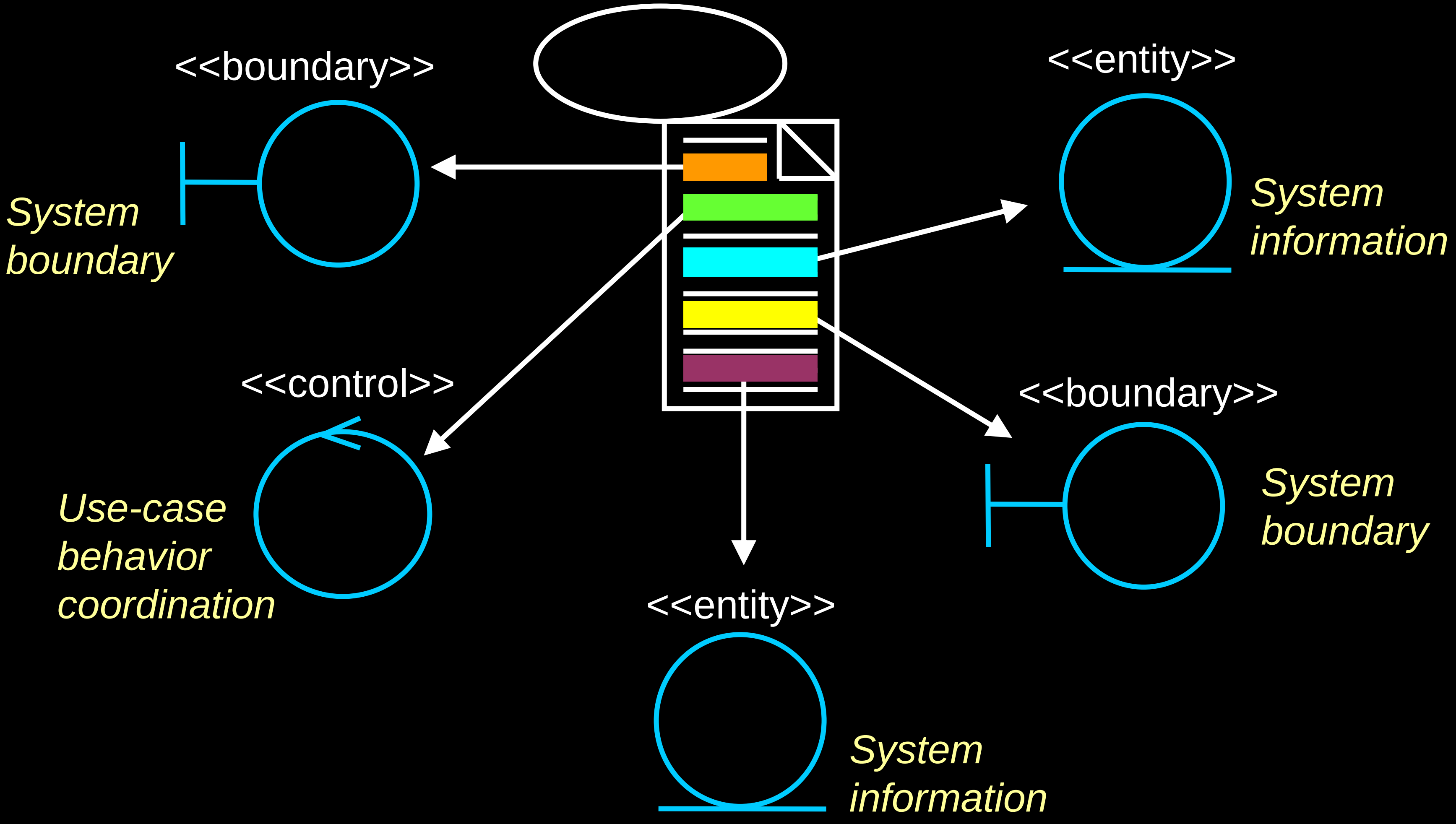
Use-Case Analysis

Find Classes from Use-Case Behavior

- The complete behavior of a use case has to be distributed to analysis classes



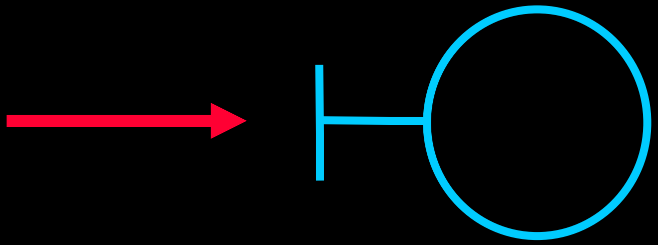
What Is an Analysis Class?



What Is a Boundary Class?

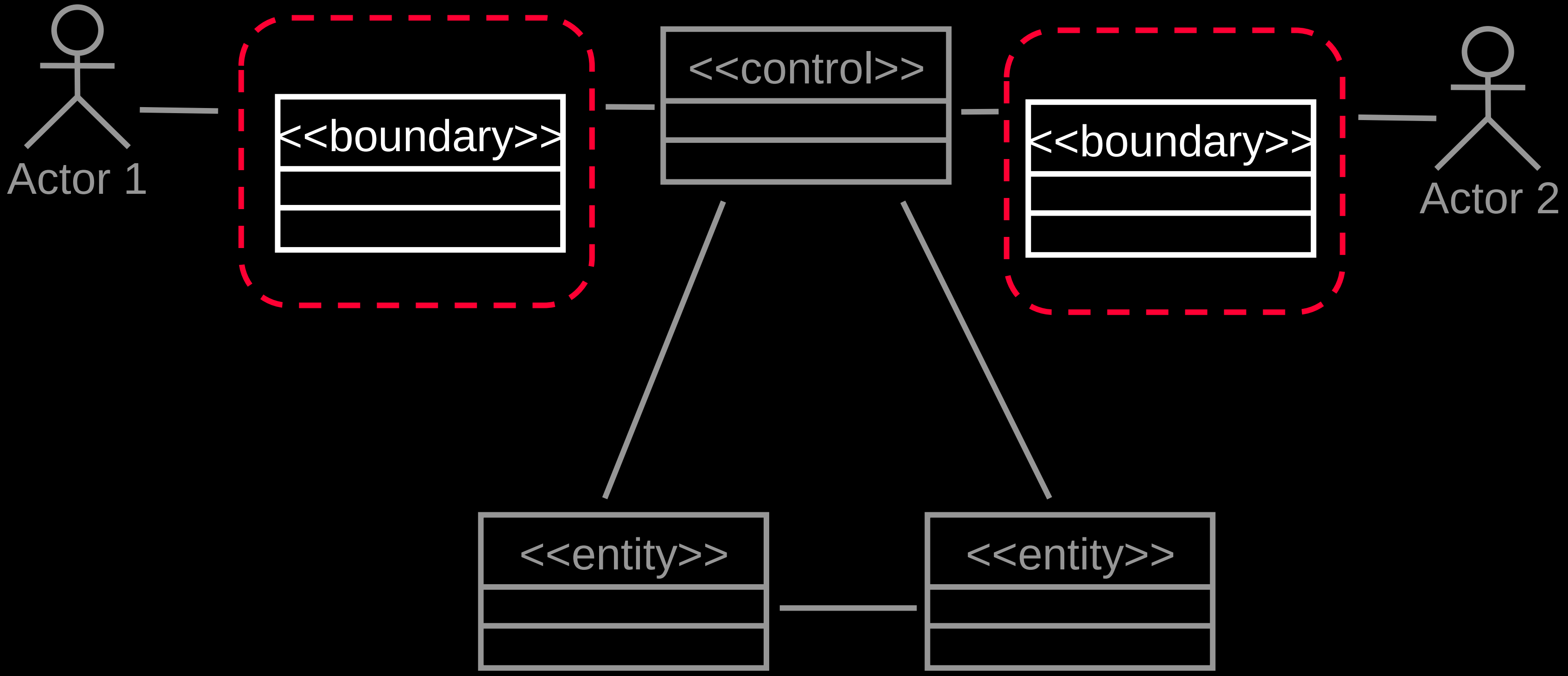
- Intermediates between the interface and something outside the system
- Several Types
  - User interface classes
  - System interface classes
  - Device interface classes
- *One boundary class per actor/use-case pair*

Analysis class stereotype



Environment dependent.

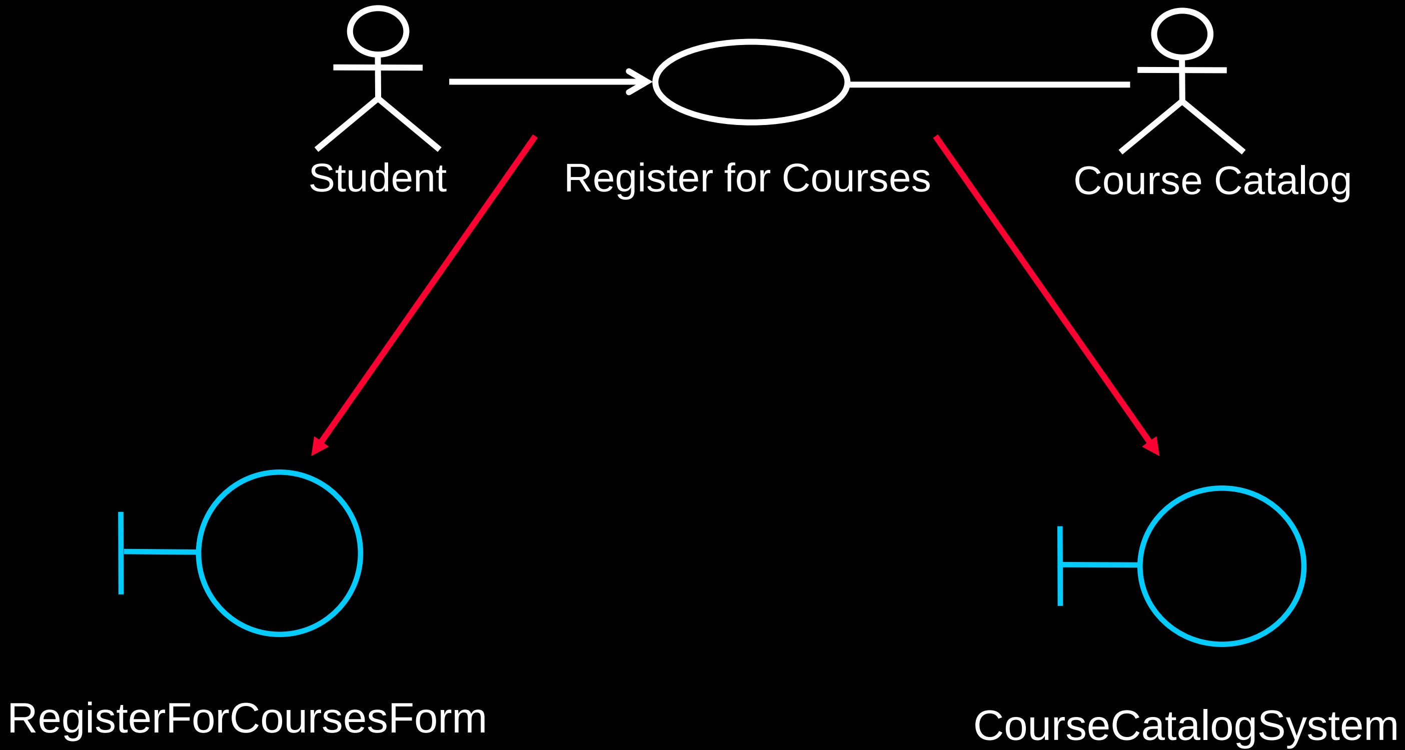
The Role of a Boundary Class



Model interaction between the system and its environment.

Example: Finding Boundary Classes

- One boundary class per actor/use case pair



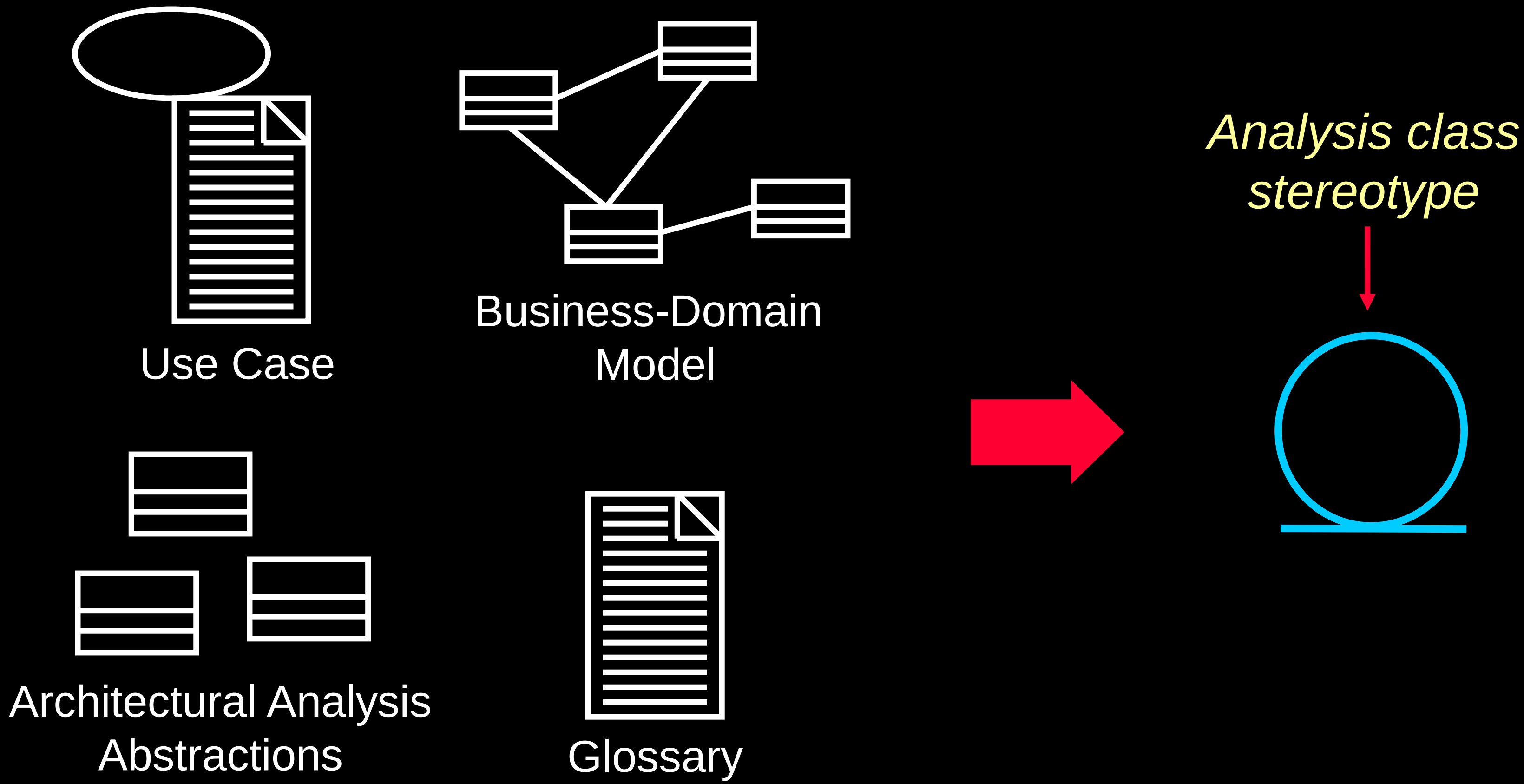
Guidelines: Boundary Class

- User Interface Classes
  - Concentrate on what information is presented to the user
  - Do NOT concentrate on the UI details
- System and Device Interface Classes
  - Concentrate on what protocols must be defined
  - Do NOT concentrate on how the protocols will be implemented

Concentrate on the responsibilities, not the details!

What Is an Entity Class?

- Key abstractions of the system



Use Case

Business-Domain Model

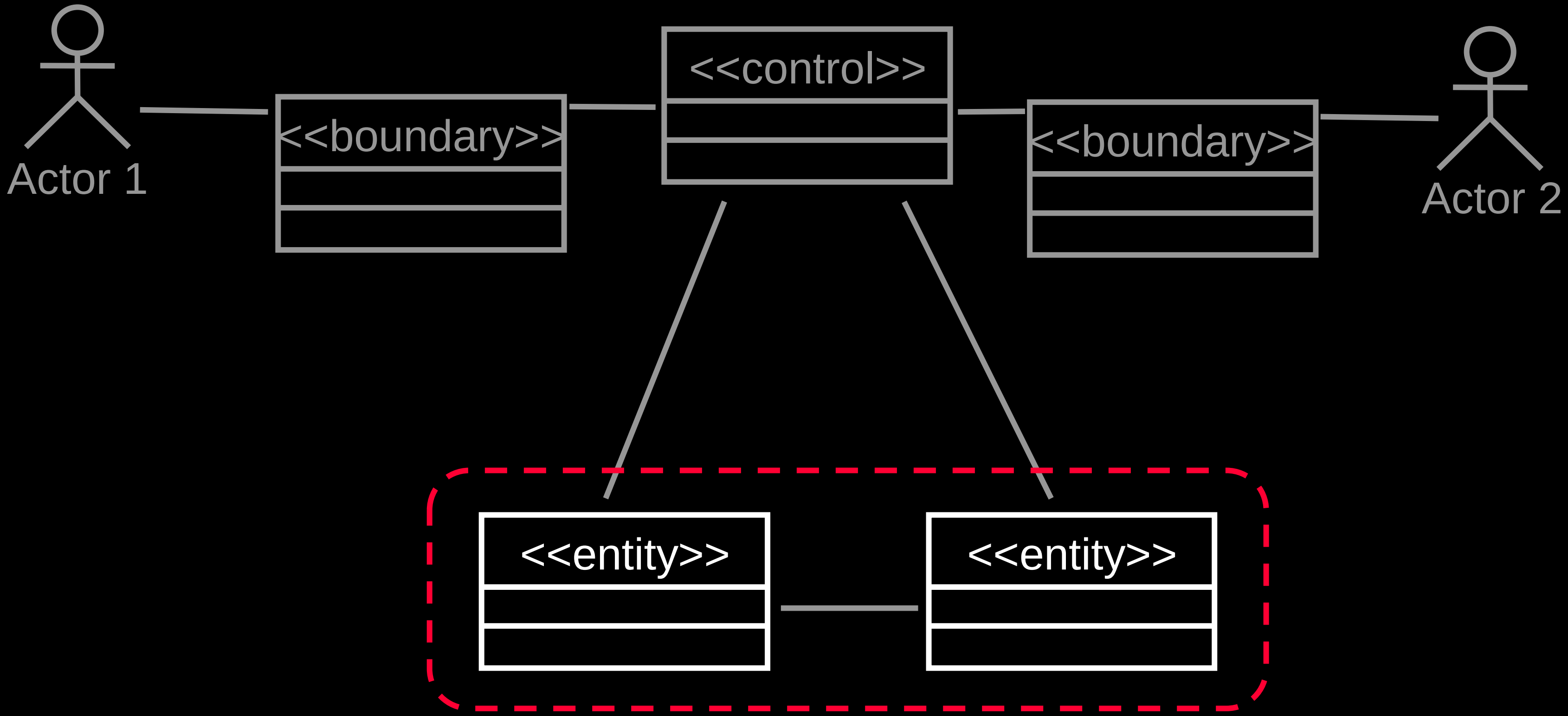
Architectural Analysis Abstractions

Glossary

Analysis class stereotype

Environment independent.

The Role of an Entity Class



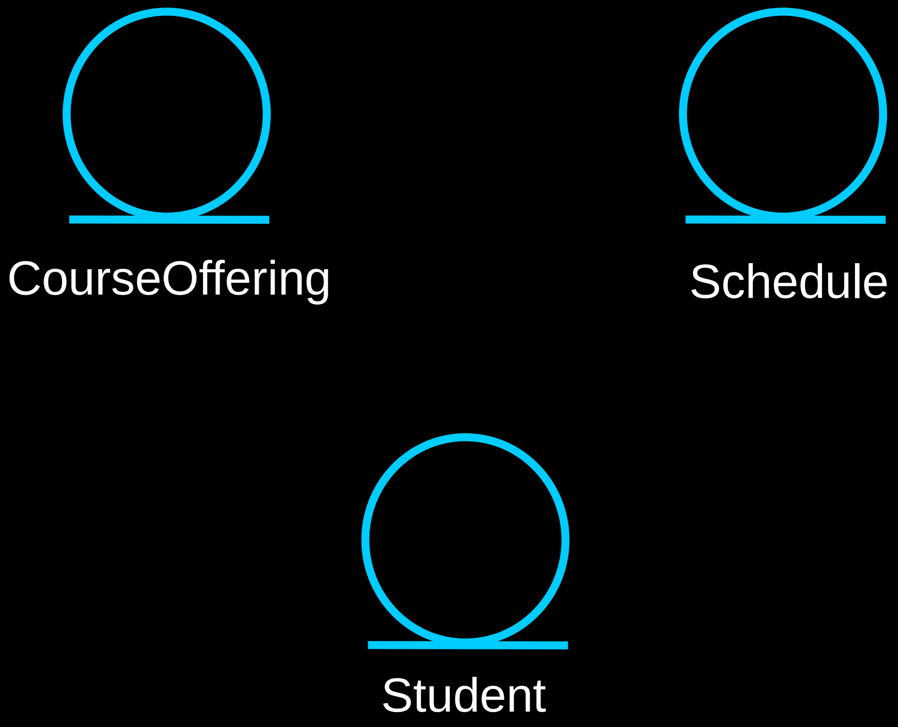
Store and manage information in the system.

Example: Finding Entity Classes

- Use use-case flow of events as input
- Key abstractions of the use case
- Traditional, filtering nouns approach
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations

Example: Candidate Entity Classes

- Register for Courses (Create Schedule)



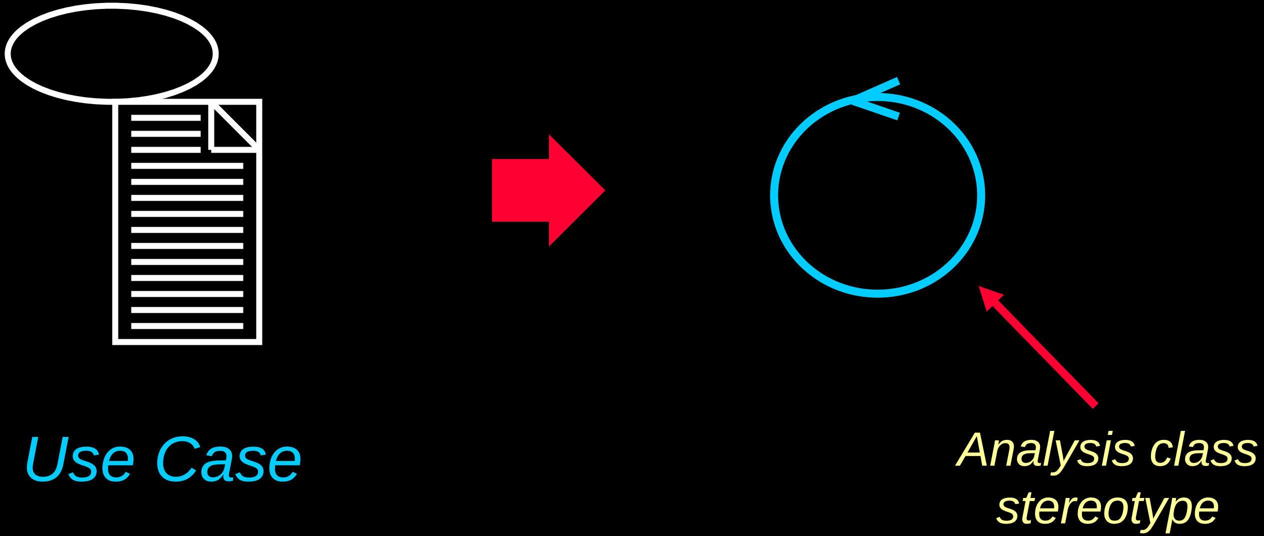
CourseOffering

Schedule

Student

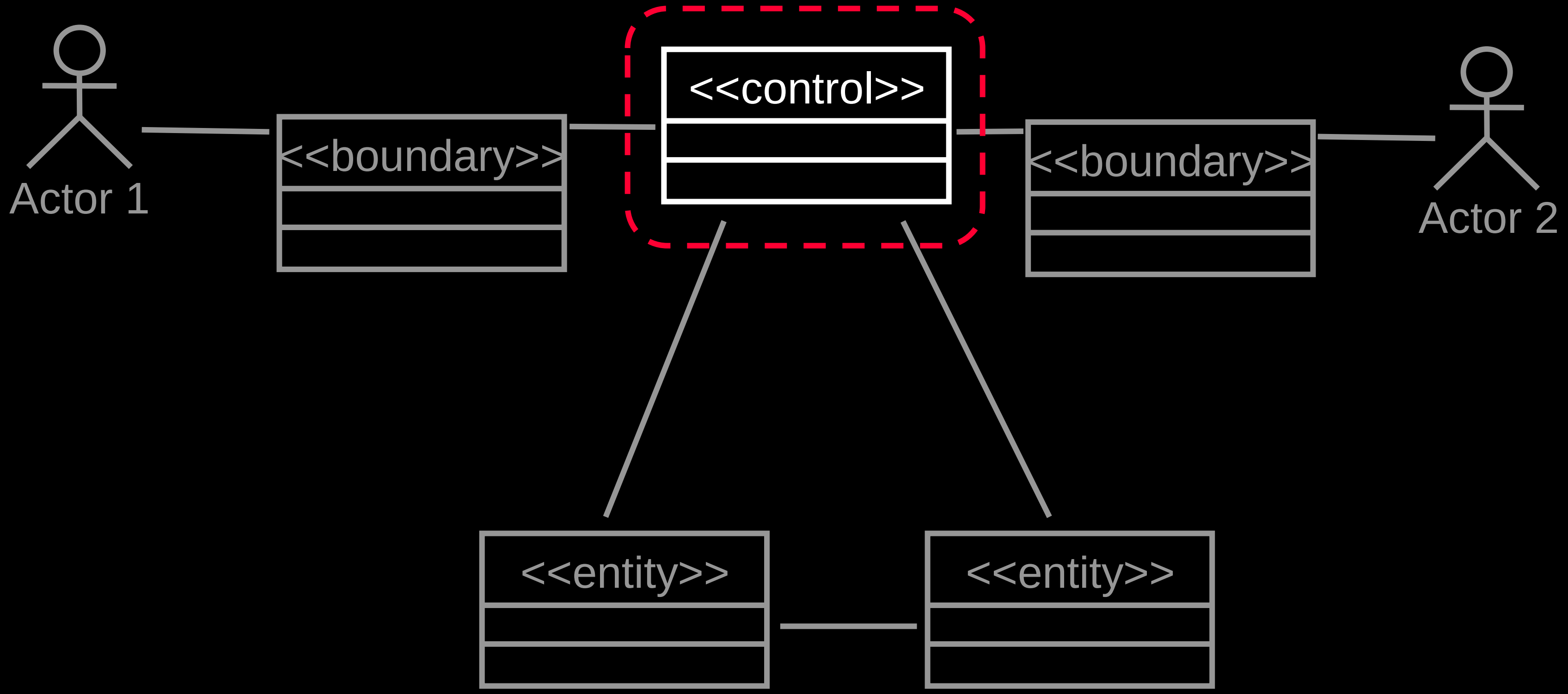
What Is a Control Class?

- Use-case behavior coordinator
  - More complex use cases generally require one or more control cases



Use-case dependent. Environment independent.

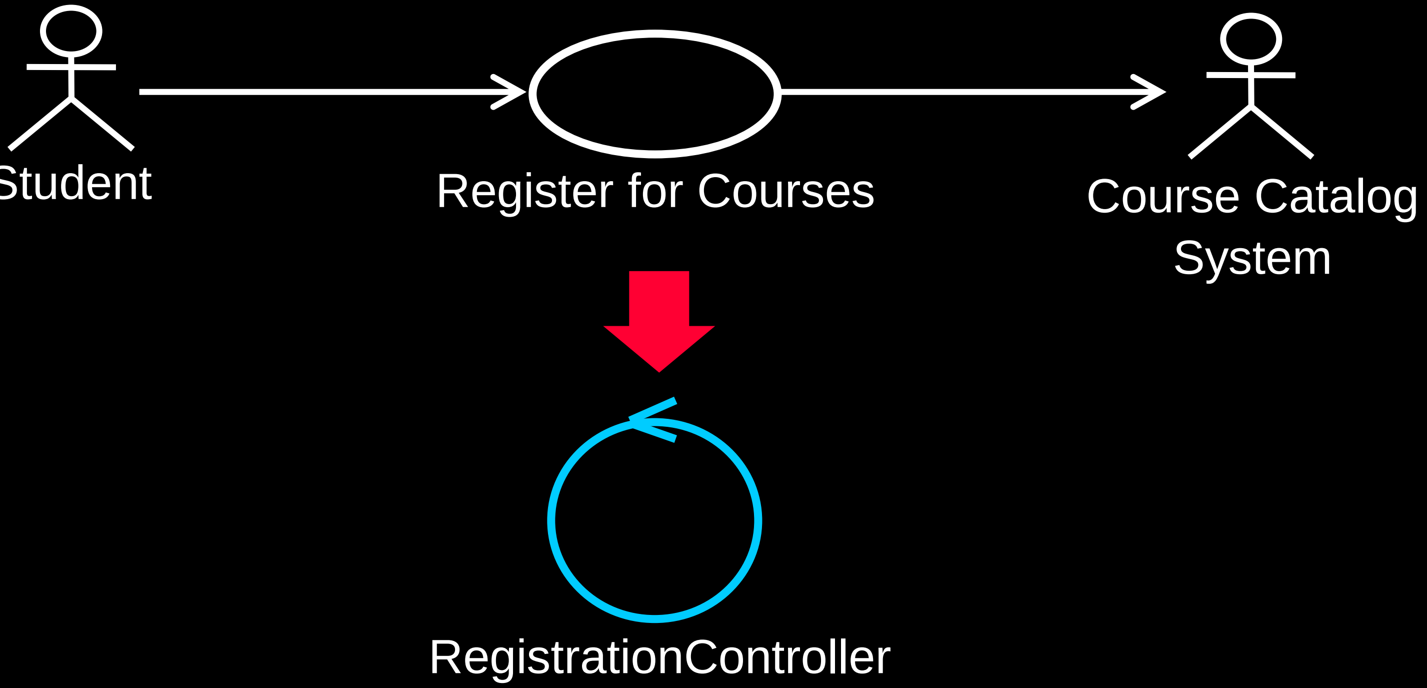
The Role of a Control Class



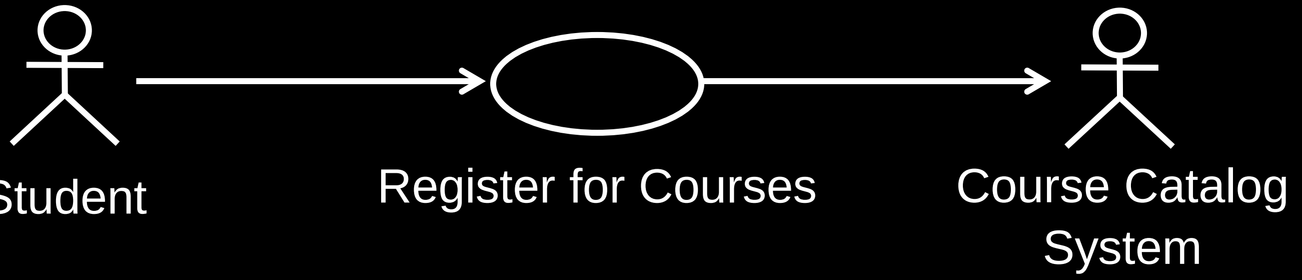
Coordinate the use-case behavior.

Example: Finding Control Classes

- In general, identify one control class per use case.
  - As analysis continues, a complex use case's control class may evolve into more than one class



Example: Summary: Analysis Classes



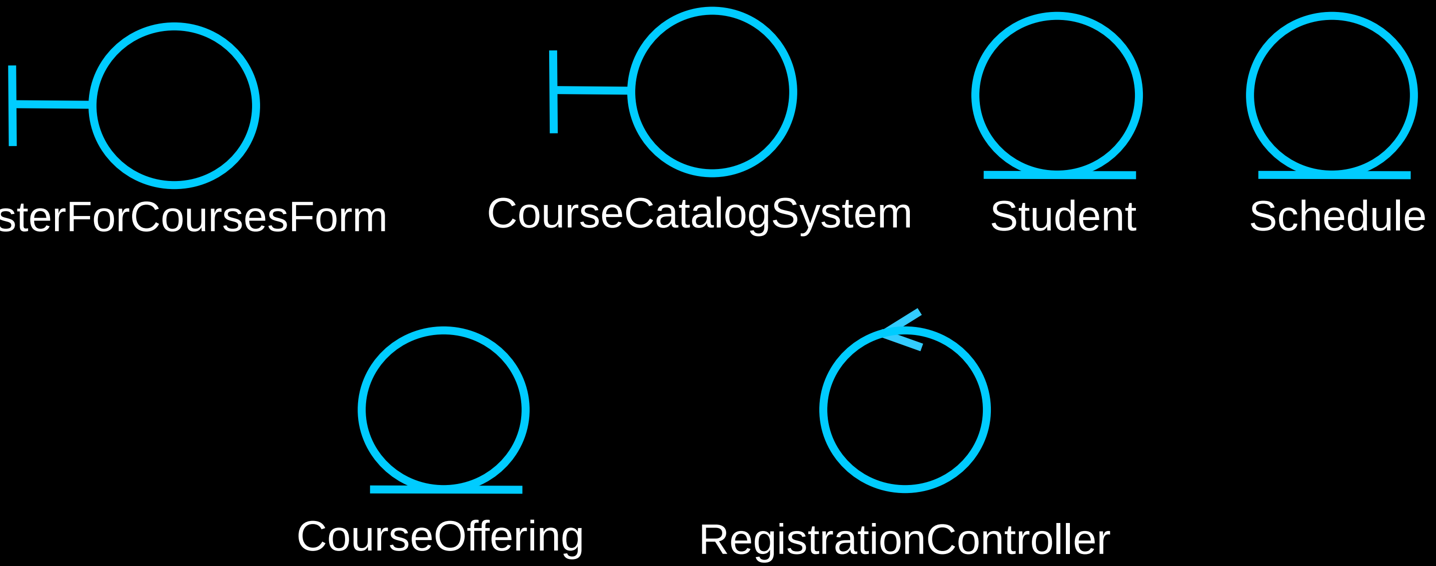
Student

Register for Courses

Course Catalog System

Use-Case Model

Design Model



RegisterForCoursesForm

CourseCatalogSystem

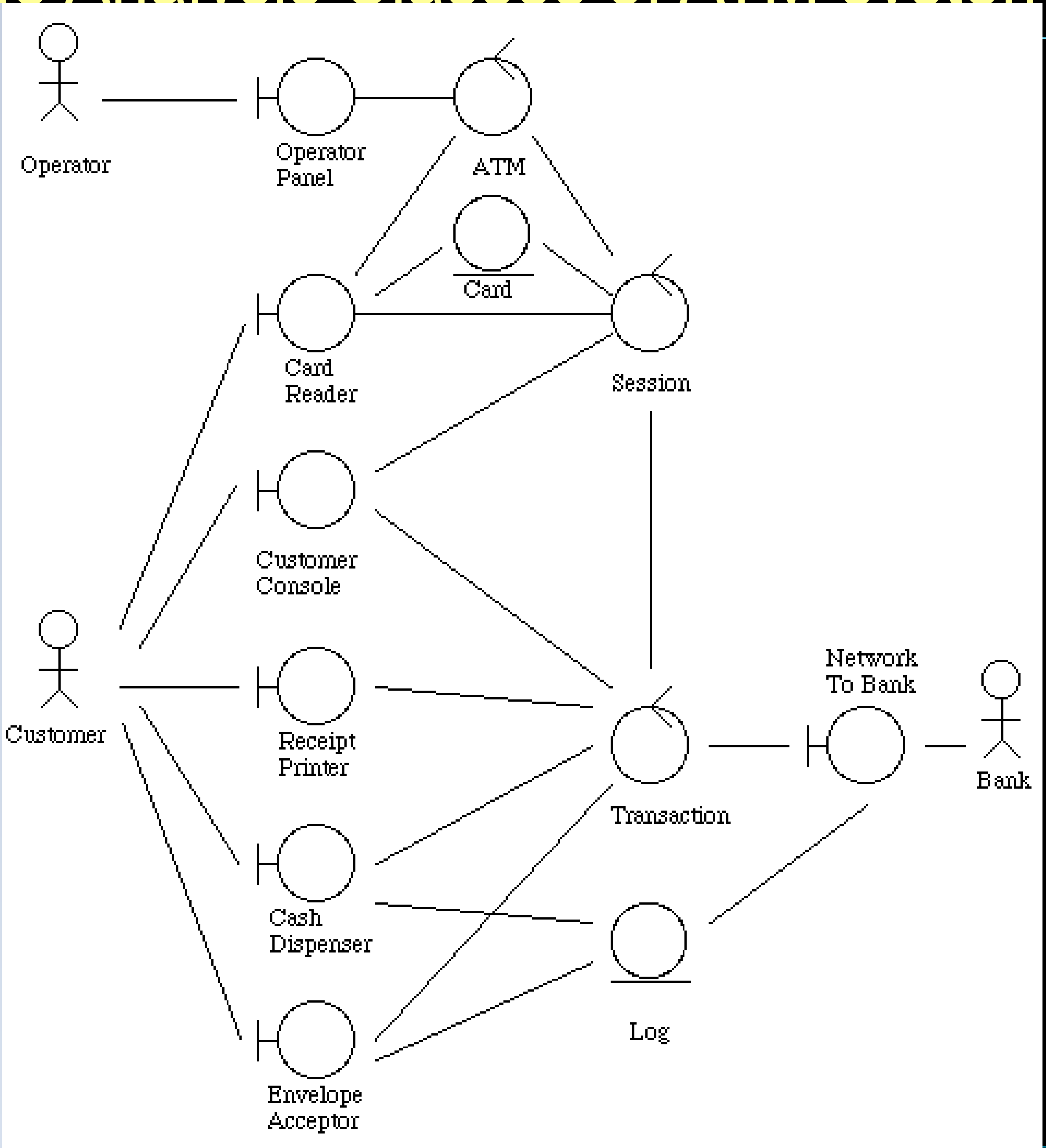
Student

Schedule

CourseOffering

RegistrationController

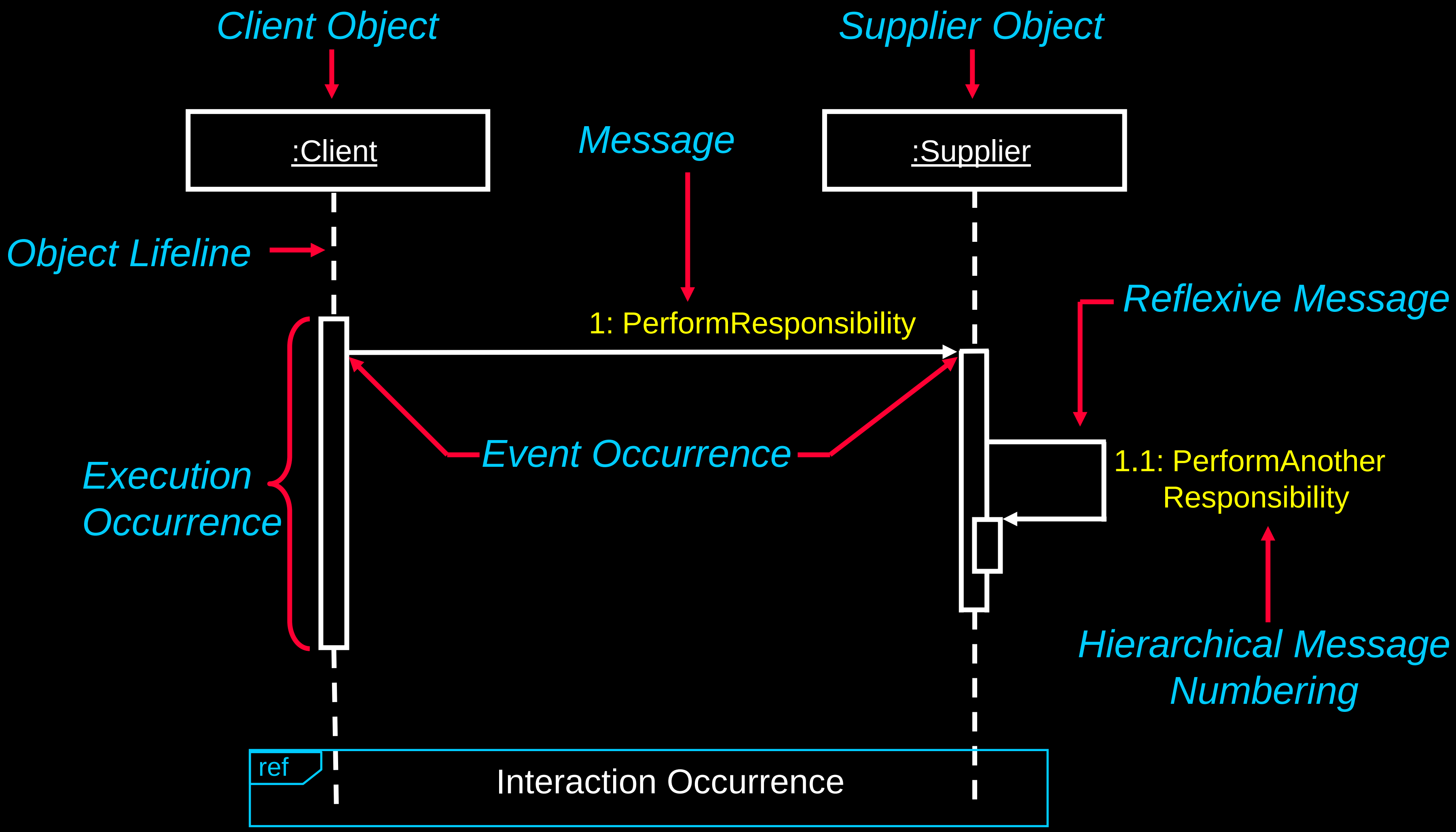
Example Analysis Classes of ATM system- withdraw

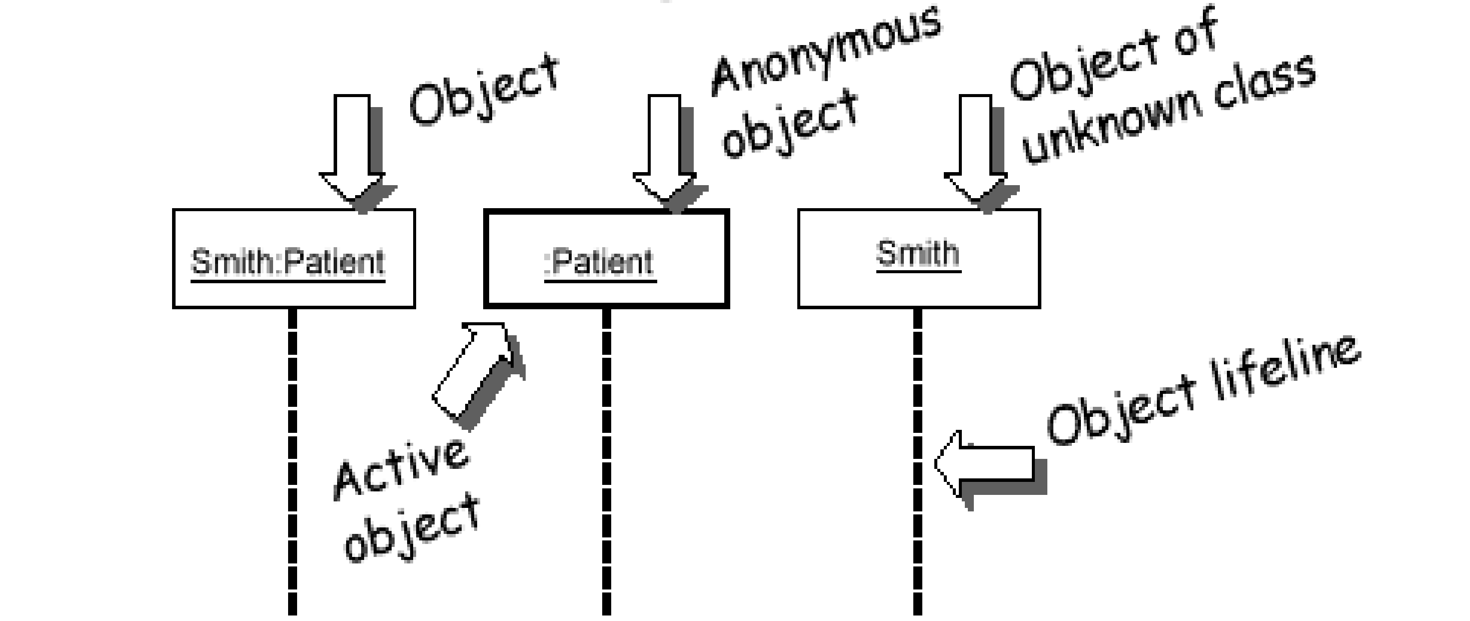


Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - ★ Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

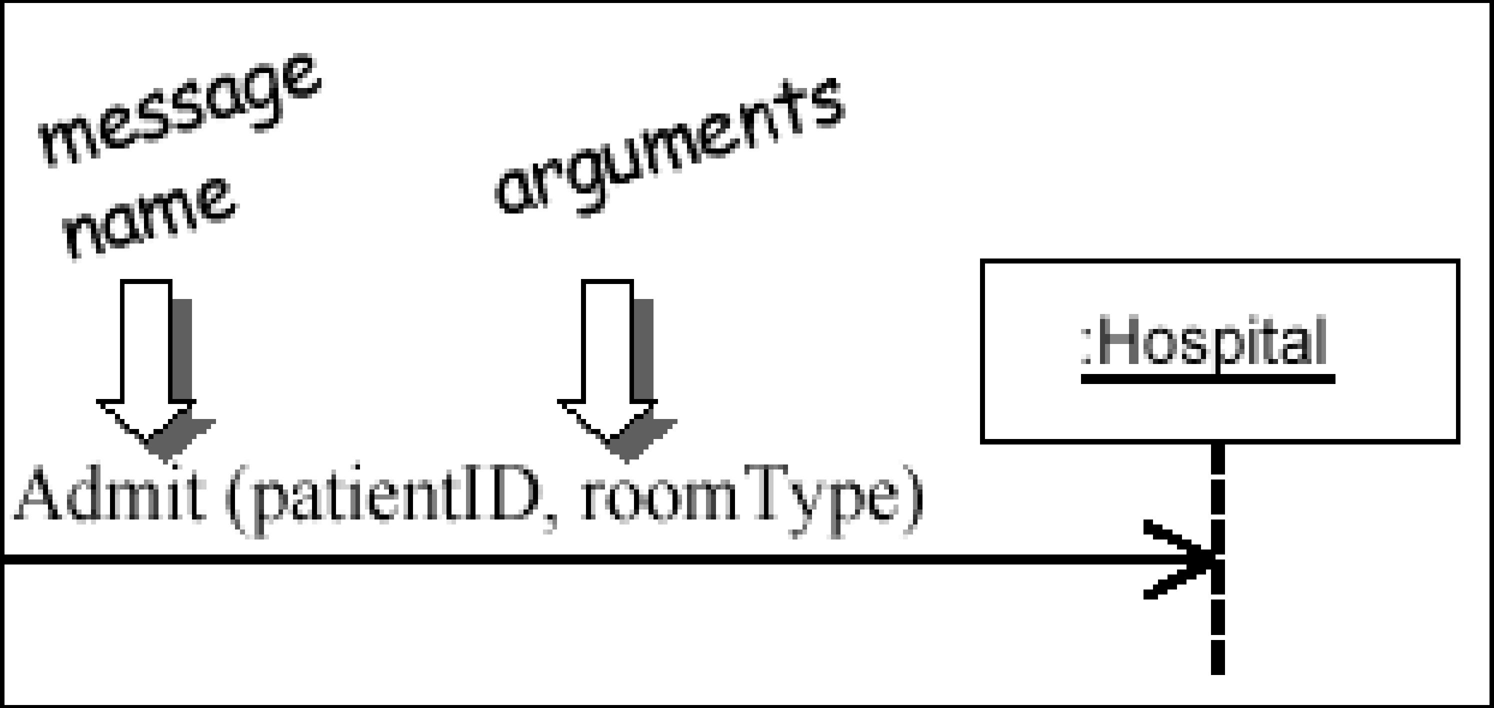
The Anatomy of Sequence Diagrams

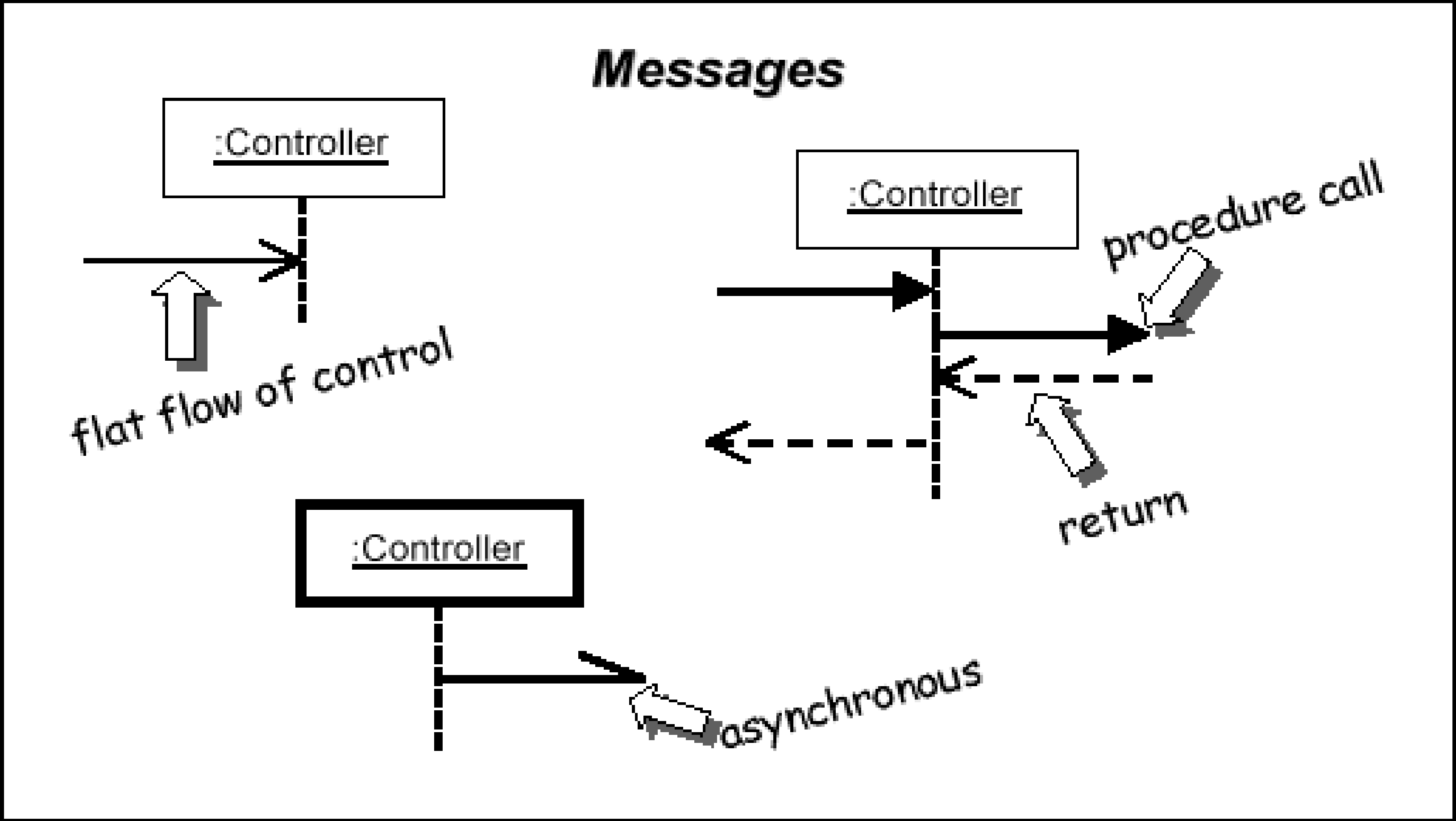


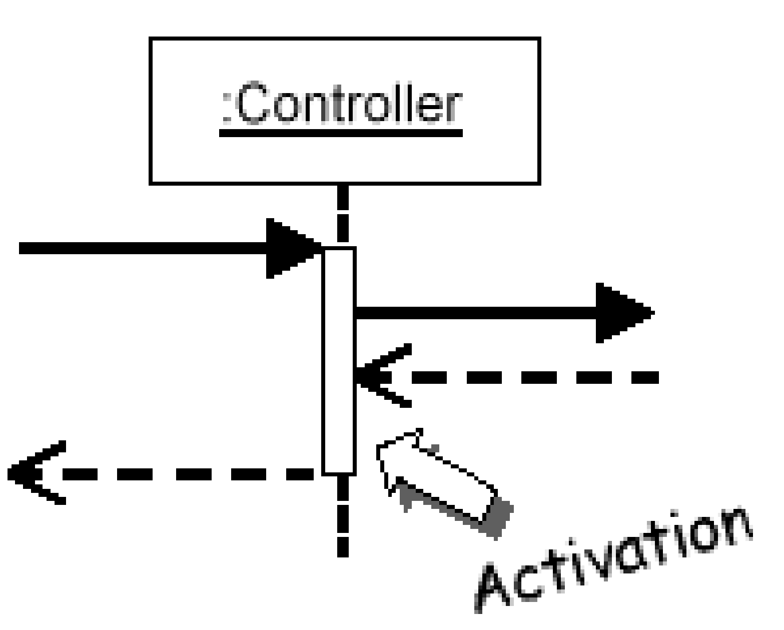


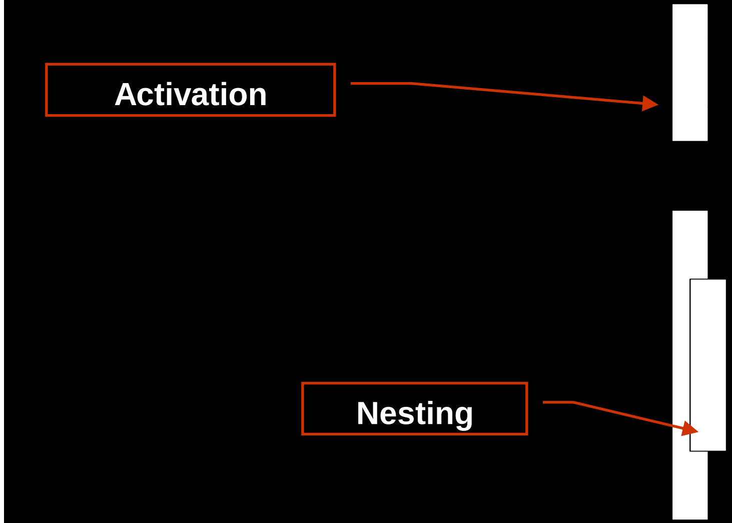
Object
Anonymous object
Object of unknown class
Smith:Patient
:Patient
Smith
Active object
Object lifeline

Name syntax: <objectname>:<classname>


message name
arguments
:Hospital
Admit (patientID, roomType)


Messages
:Controller
flat flow of control
:Controller
procedure call
return
:Controller
asynchronous

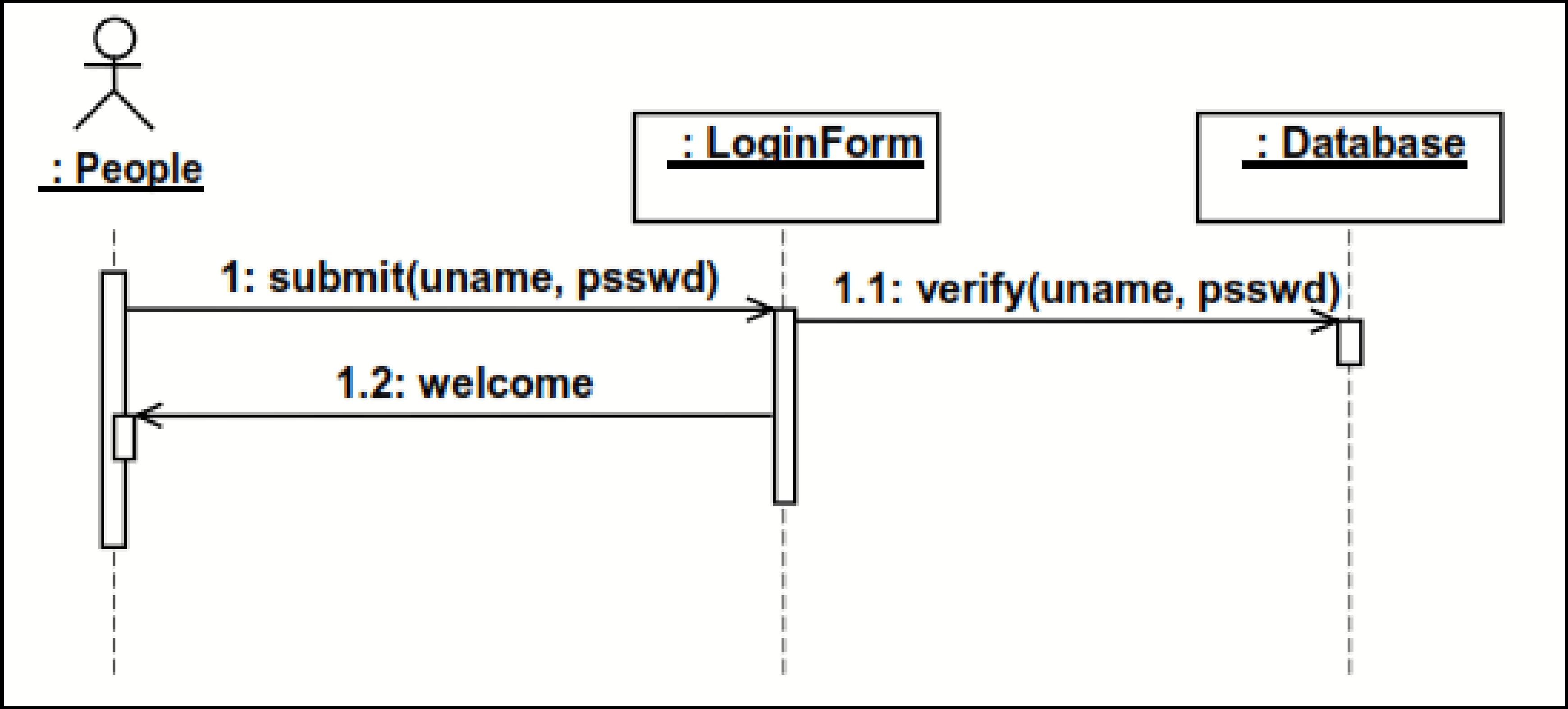




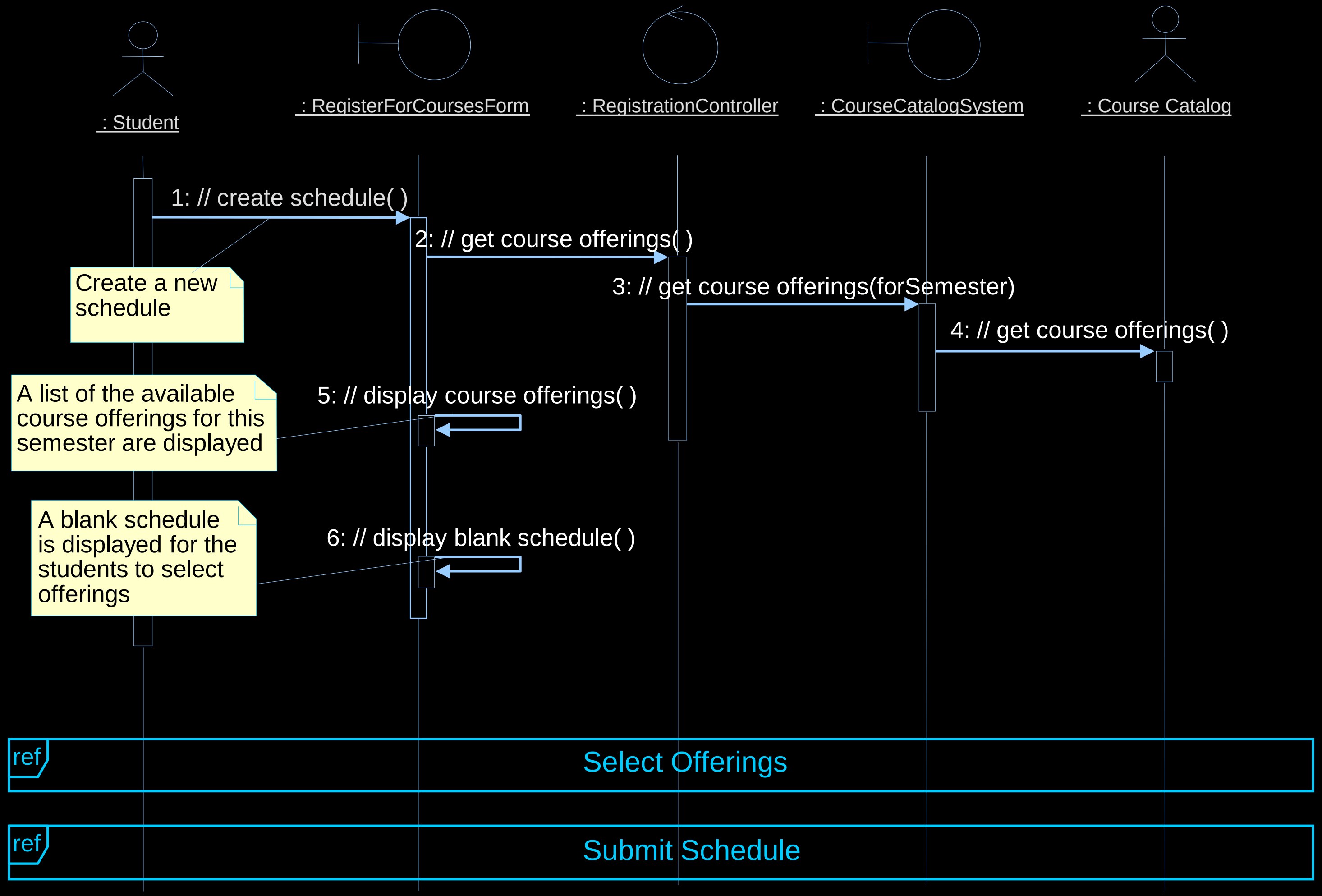
Activation

Nesting

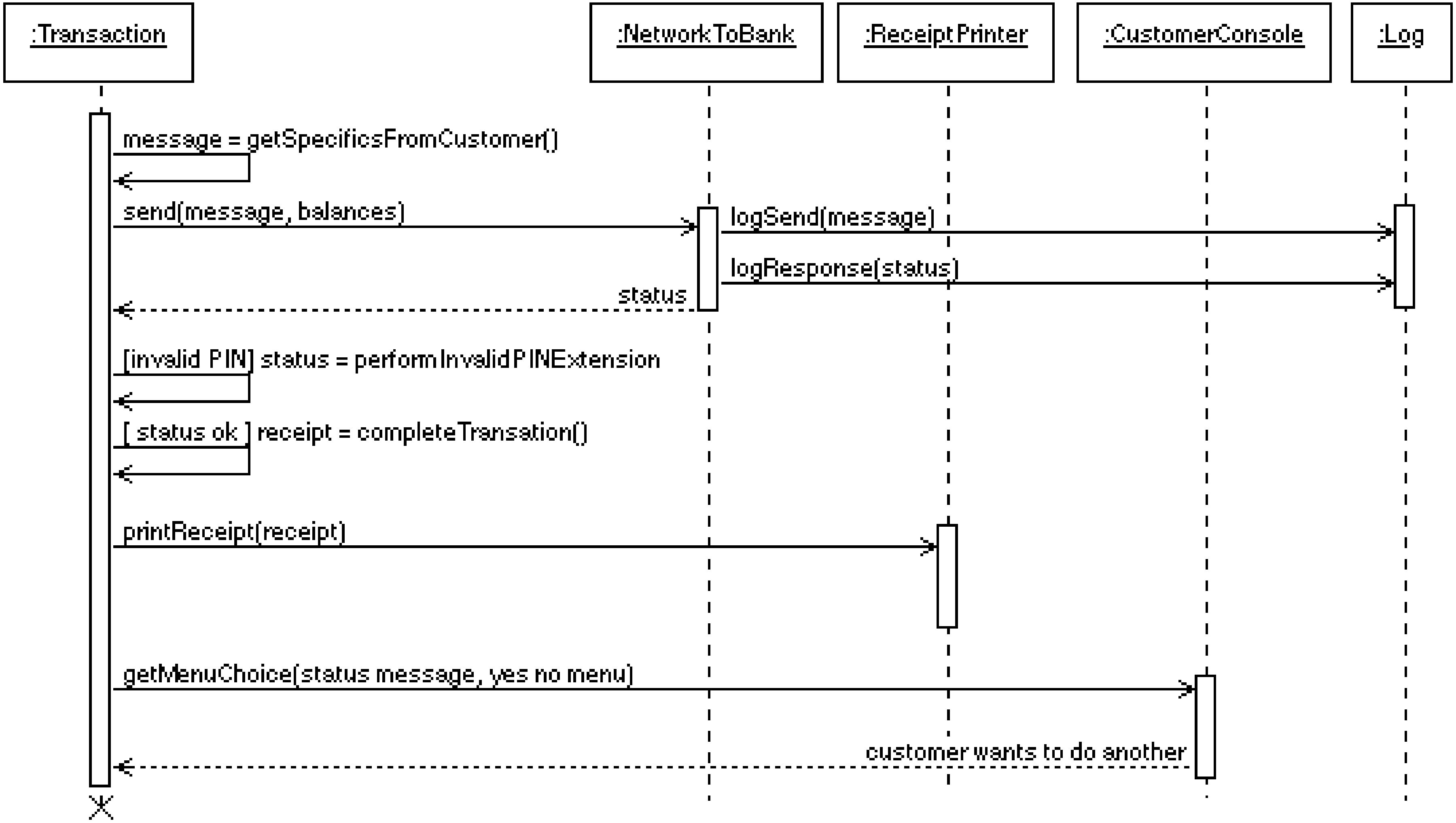
- Login sequence



Example: Sequence Diagram


: Student
: RegisterForCoursesForm
: RegistrationController
: CourseCatalogSystem
: Course Catalog
1: // create schedule( )
2: // get course offerings( )
3: // get course offerings(forSemester)
4: // get course offerings( )
5: // display course offerings( )
6: // display blank schedule( )
Create a new schedule
A list of the available course offerings for this semester are displayed
A blank schedule is displayed for the students to select offerings
ref
Select Offerings
ref
Submit Schedule

Transaction Sequence Diagram


:Transaction
:NetworkToBank
:ReceiptPrinter
:CustomerConsole
:Log
message = getSpecificsFromCustomer()
send(message, balances)
logSend(message)
logResponse(status)
status
[invalid PIN] status = performInvalidPINExtension
[status ok] receipt = completeTransation()
printReceipt(receipt)
getMenuChoice(status message, yes no menu)
customer wants to do another