CHAPTER 12
A picture is worth 1024 words

Objectives

- After finish this chapter, student should understand the purpose of every type of diagrams, when and how to build them.
- Student could choice the plan to model requirements.

Contents

1. A useful of picture in requirements representation
2. Modeling the requirements
3. From voice of the customer to analysis models
4. Selecting the right representations
5. Data flow diagram
6. Swimlane diagram
7. State-transition diagram and state table
8. Decision tables and decision trees
9. Modeling on agile projects

A useful of picture in requirements representation

- Diagrams communicate certain types of information more efficiently than text can.
- Pictures help bridge language and vocabulary barriers among team members

Modeling the requirements

- Visual requirements models can help you identify missing, extraneous, and inconsistent requirements.
- Given the limitations of human short-term memory, analyzing a list of one thousand requirements for inconsistencies, duplication, and extraneous requirements is nearly impossible.

From voice of the customer to analysis models

- By listening carefully to how customers present their requirements, the BA can pick out keywords that translate into specific model elements.

TABLE 12-1 Relating the customer's voice to analysis model components

Type of word	Examples	Analysis model components
Noun	People, organizations, software systems, data elements, or objects that exist	■ External entities, data stores, or data flow (DFD) ■ Actors (use case diagram) ■ Entities or their attributes (ERD) ■ Lanes (swimlane diagram) ■ Objects with states (STD)
Verb	Actions, things a user or system can do, or events that can take place	■ Processes (DFD) ■ Process steps (swimlane diagram) ■ Use cases (use case diagram) ■ Relationships (ERD) ■ Transitions (STD) ■ Activities (activity diagram) ■ Events (event-response table)
Conditional	Conditional logic statements, such as if/then	■ Decisions (decision tree, decision table, or activity diagram) ■ Branching (swimlane diagram or activity diagram)

Selecting the right representations

TABLE 12-2 Choosing the most appropriate representation techniques

Information depicted	Representation techniques
System external interfaces	■ *The context diagram* and *use case diagram* identify objects outside the system that connect to it. The context diagram and *data flow diagrams* illustrate the system inputs and outputs at a high level of abstraction. The *ecosystem map* identifies possible systems that interact, but includes some that do not interface directly as well. *Swimlane diagrams* show what happens in the interactions between systems. ■ External interface details can be recorded in input and output *file formats* or *report layouts*. Products that include both software and hardware components often have *interface specifications* with data attribute definitions, perhaps in the form of an application programming interface or specific input and output signals for a hardware device.
Business process flow	■ A top-level *data flow diagram* represents how a business process handles data at a high level of abstraction. *Swimlane diagrams* show the roles that participate in executing the various steps in a business process flow. ■ Refined levels of *data flow diagrams* or *swimlane diagrams* can represent business process flows in considerable detail. Similarly, *flowcharts* and *activity diagrams* can be used at either high or low levels of abstraction, although most commonly they are used to define the details of a process.
Data definitions and data object relationships	■ The *entity-relationship diagram* shows the logical relationships between data objects (entities). *Class diagrams* show the logical connections between object classes and the data associated with them. ■ The *data dictionary* contains detailed definitions of data structures and individual data items. Complex data objects are progressively broken down into their constituent data elements.

Selecting the right representations

Information depicted	Representation techniques
System and object states	■ *State-transition diagrams* and *state tables* represent a high-abstraction view of the possible states of a system or object and the changes between states that can take place under certain circumstances. These models are helpful when multiple use cases can manipulate (and change the state of) certain objects. ■ Some analysts create an *event-response table* as a scoping tool, identifying external events that help define the product's scope boundary. You can also specify individual functional requirements with an event-response table by detailing how the system should behave in response to each combination of external event and system state. ■ *Functional requirements* provide the details that describe exactly what user and system behaviors lead to status changes.
Complex logic	■ A *decision tree* shows the possible outcomes from a set of related decisions or conditions. A *decision table* identifies the unique functional requirements associated with the various combinations of true and false outcomes for a series of decisions or conditions.
User interfaces	■ The *dialog map* provides a high-level view of a proposed or actual user interface, showing the various display elements and possible navigation pathways between them. ■ *Storyboards* and *low-fidelity prototypes* flesh out the dialog map by showing what each screen will contain without depicting precise details. *Display-action-response models* describe the display and behavior requirements of each screen. ■ *Detailed screen layouts* and *high-fidelity prototypes* show exactly how the display elements will look. *Data field definitions* and *user interface control descriptions* provide additional detail.

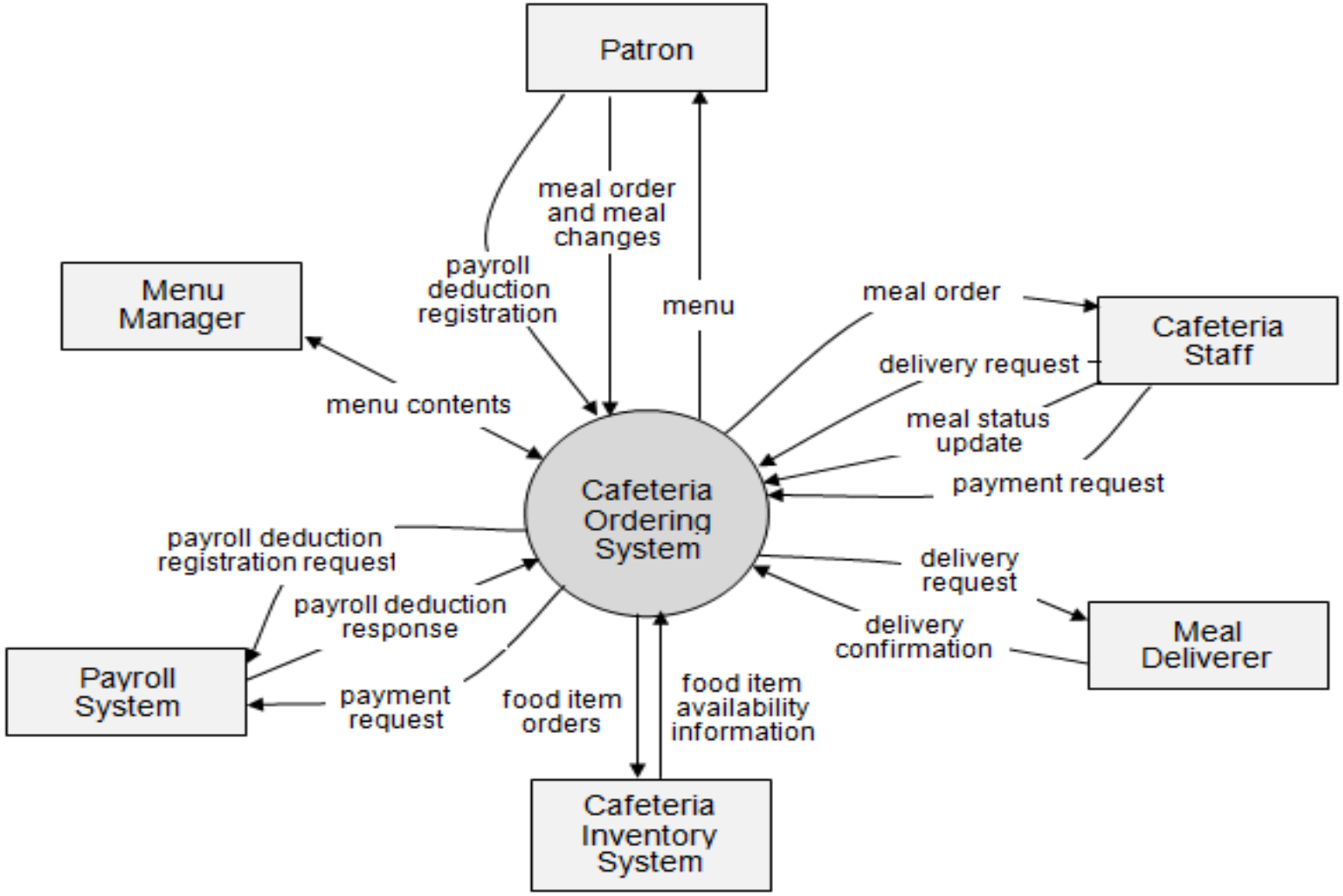
Selecting the right representations

Information depicted	Representation techniques
User task descriptions	■ *User stories*, *scenarios*, and *use case specifications* describe user tasks in various levels of detail. ■ *Swimlane diagrams* illustrate the business process or interplay between multiple actors and the system. *Flowcharts* and *activity diagrams* visually depict the flow of the use case dialog and branches into alternative flows and exceptions. ■ *Functional requirements* provide detailed descriptions of how the system and user will interact to achieve valuable outcomes. *Test cases* provide an alternative low-abstraction view, describing exactly what system behavior to expect under specific conditions of inputs, system state, and actions.
Nonfunctional requirements (quality attributes, constraints)	■ Quality attributes and constraints are usually written in the form of *natural language text*, but that often results in a lack of precision and completeness. Chapter 14, "Beyond functionality" describes a definitive technique for precisely specifying nonfunctional requirements called *Planguage* (Gilb 2005).

Data flow diagram – context diagram

- Purpose
  - DFDs provide a big-picture view of how data moves through a system, which other models don't show well
  - A gives context to the functional requirements regarding how the user performs specific tasks.
  - DFDs can be used as a technique to identify missing data requirements

Dataflow diagram: example



Swimlane diagram

- Purpose:
  - Provide a way to represent the steps involved in a business process or the operations of a proposed software system.
  - They are a variation of flowcharts, subdivided into visual subcomponents called lanes. The lanes can represent different systems or actors that execute the steps in the process.
  - Swimlane diagrams are most commonly used to show business processes, workflows, or system and user interactions.
  - They are similar to UML activity diagrams. Swimlane diagrams are sometimes called cross-functional diagrams.

Swimlane example

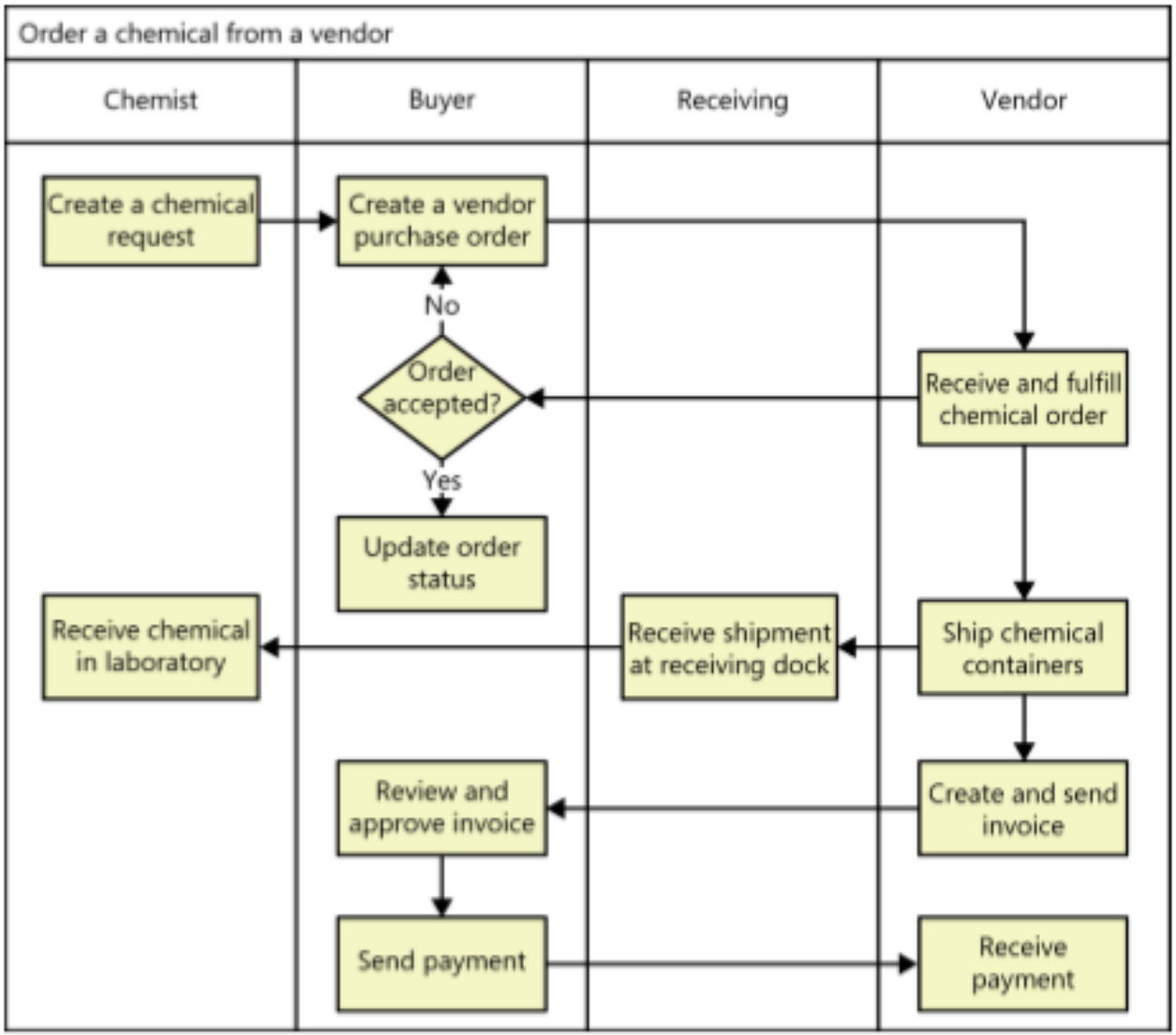


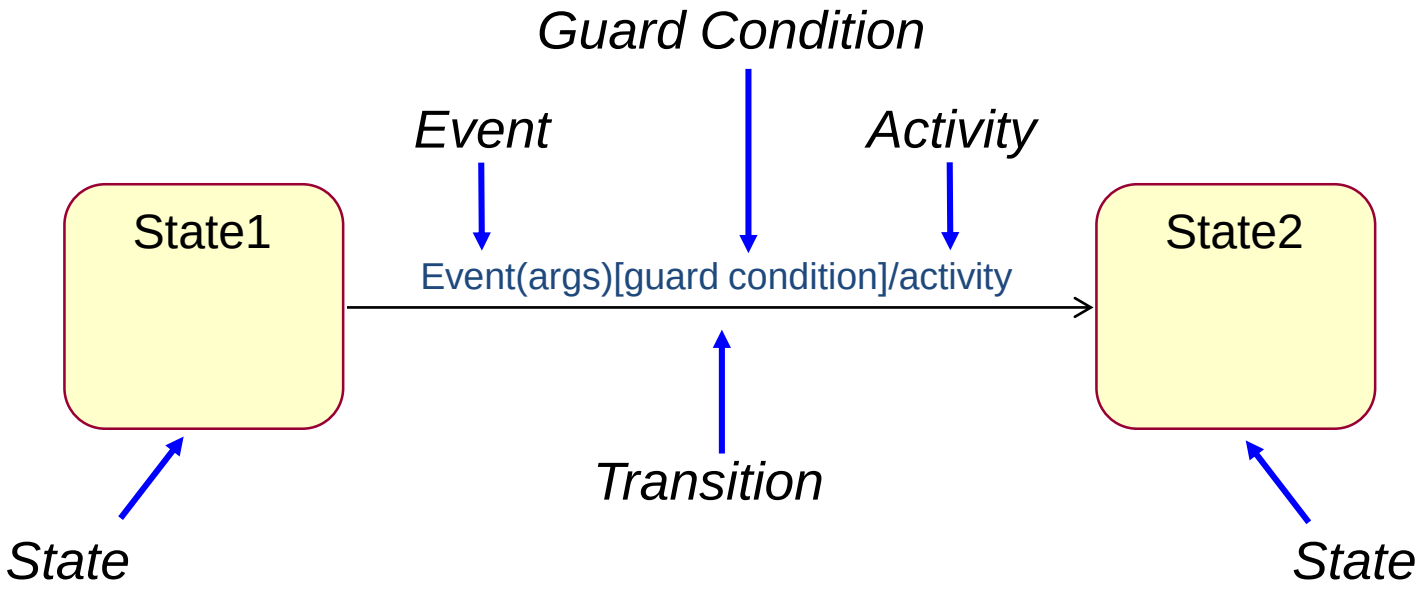
FIGURE 12-2 Partial swimlane diagram for a process in the Chemical Tracking System.

State-transition diagram and state table

- Purpose: Software systems involve a combination of functional behavior, data manipulation, and state changes. Real-time systems and process control applications can exist in one of a limited number of states at any given time.
- A state change can take place only when well-defined criteria are satisfied, such as receiving a specific input stimulus under certain conditions
- State transition diagram contains three type of elements:
  - Possible system states, shown as rectangles
  - Allowed state changes or transitions, shown as arrows connecting pairs of rectangles
  - Events or conditions that cause each transition to take place, shown as text labels on each transition arrow

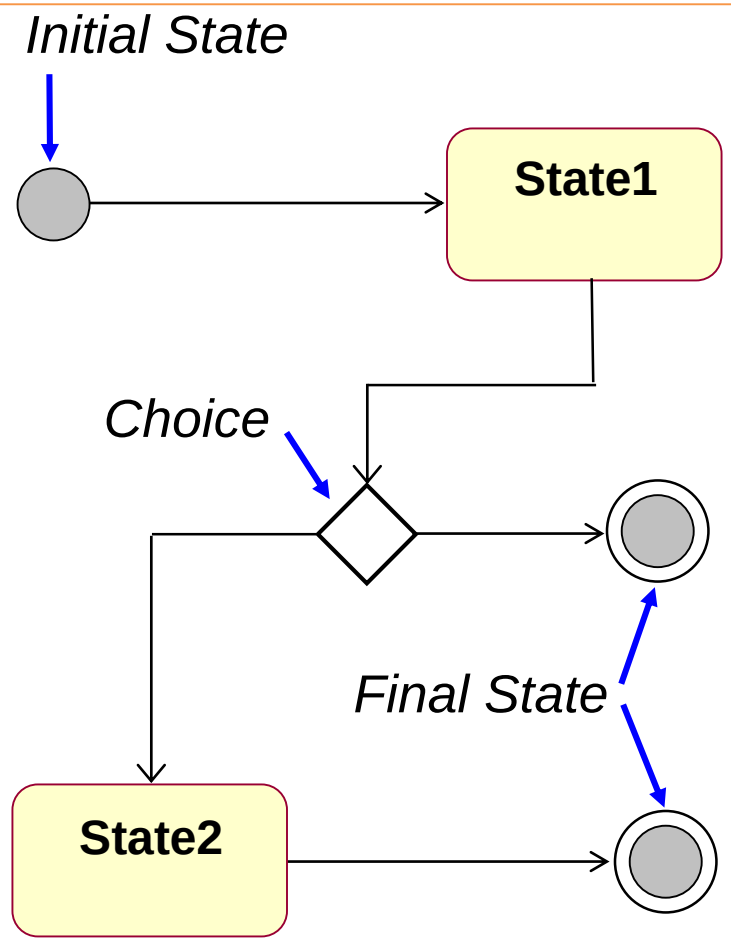
What is a State Machine?

- A directed graph of states (nodes) connected by transitions (directed arcs)
- Describes the life history of a reactive object



Pseudo States

- Initial state
  - The state entered when an object is created
  - Mandatory, can only have one initial state
- Choice
  - Dynamic evaluation of subsequent guard conditions
  - Only first segment has a trigger
- Final state
  - Indicates the object's end of life
  - Optional, may have more than one



Identify and Define the States

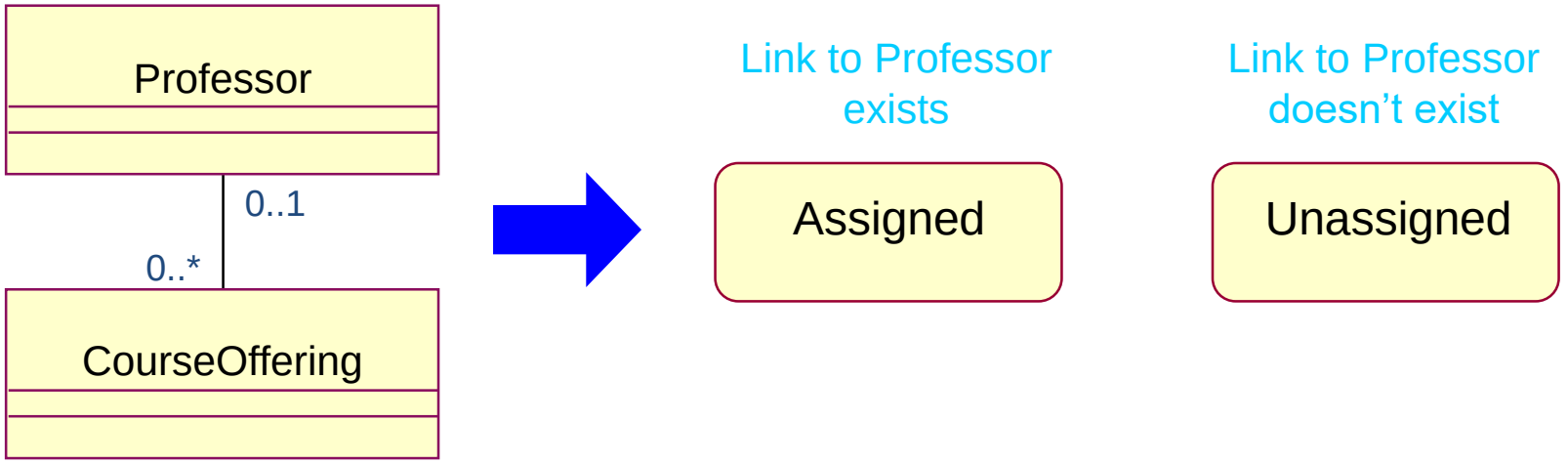
- Significant, dynamic attributes

The maximum number of students per course offering is 10

numStudents < 10 → Open

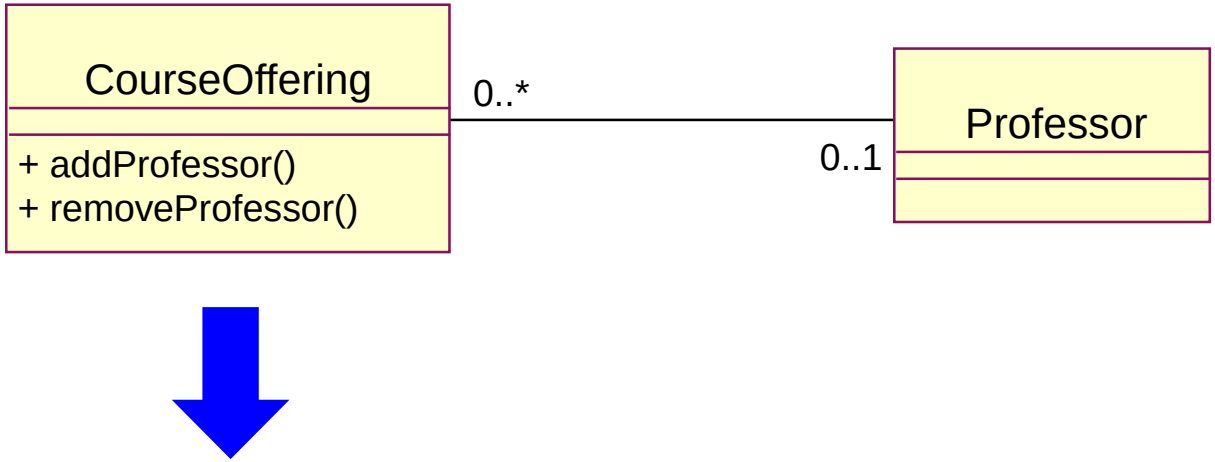
numStudents > = 10 → Closed

- Existence and non-existence of certain links



Identify the Events

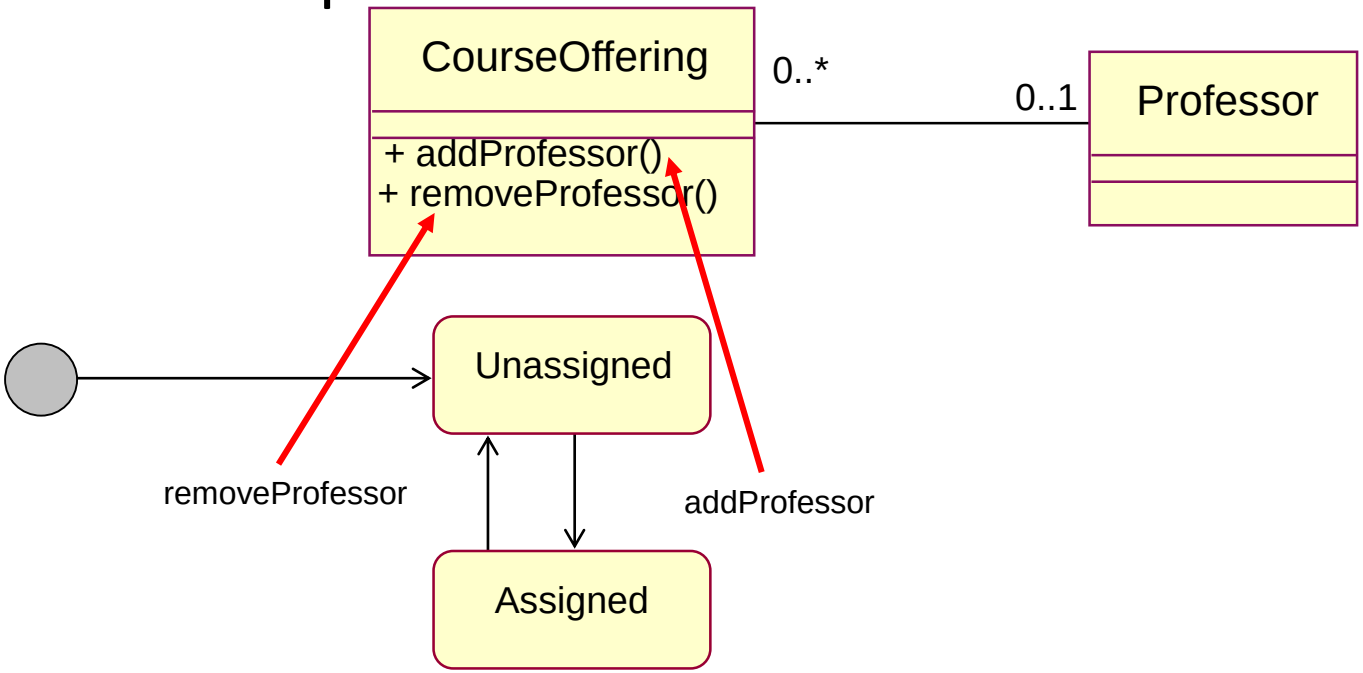
- Look at the class interface operations



Events: addProfessor,
removeProfessor

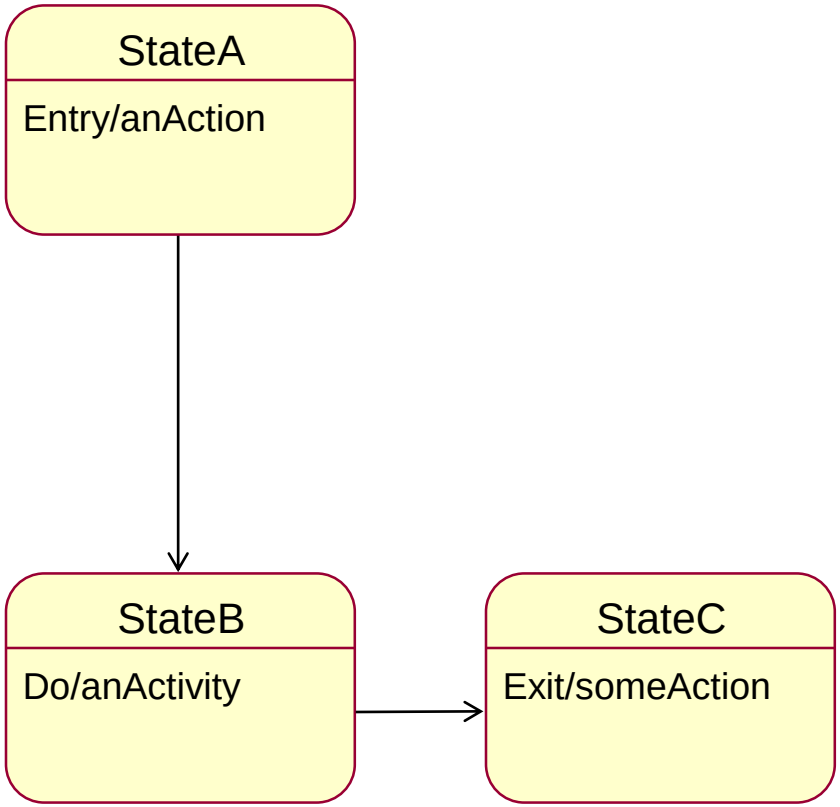
Identify the Transitions

- For each state, determine what events cause transitions to what states, including guard conditions, when needed
- Transitions describe what happens in response to the receipt of an event

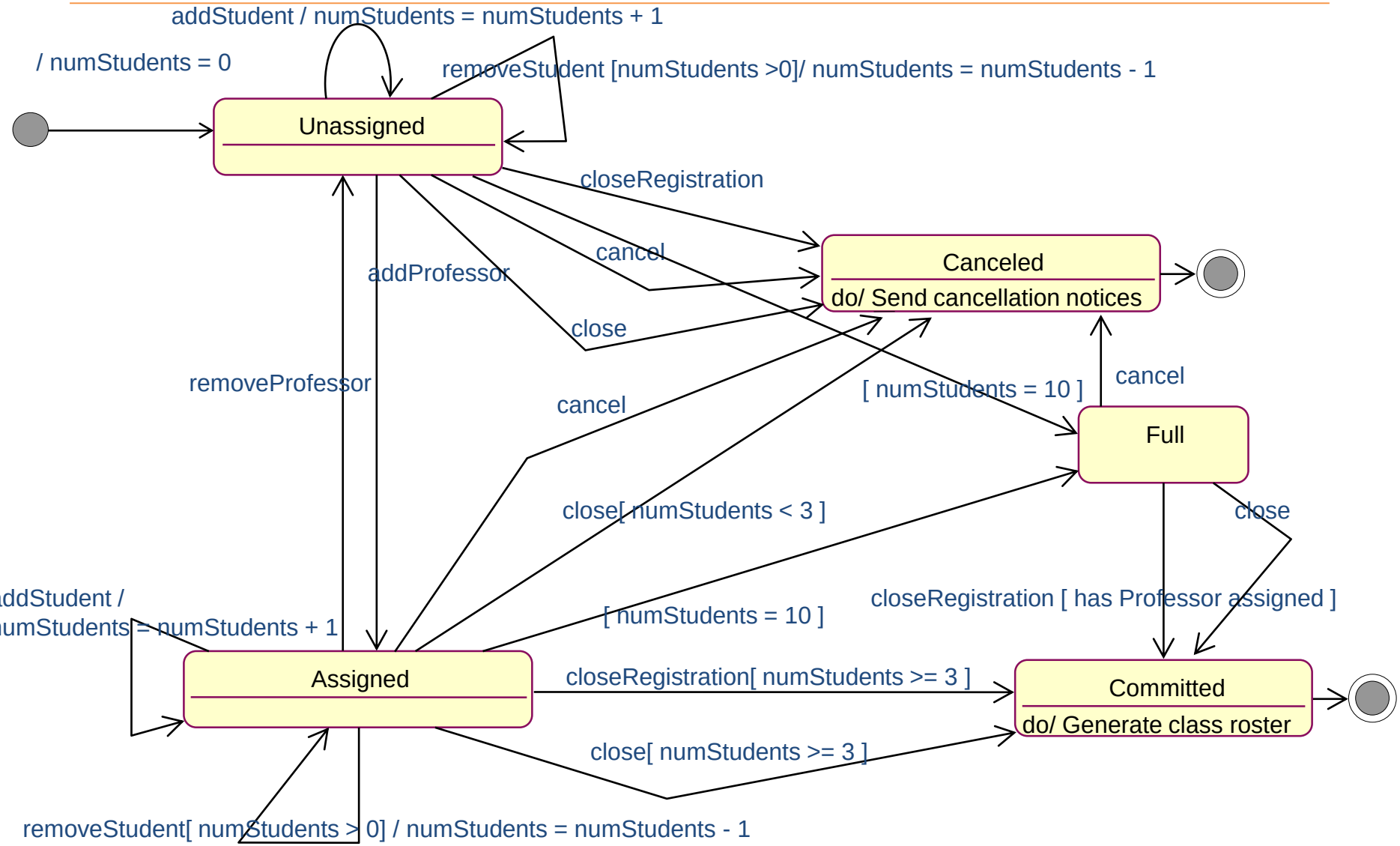


Add Activities

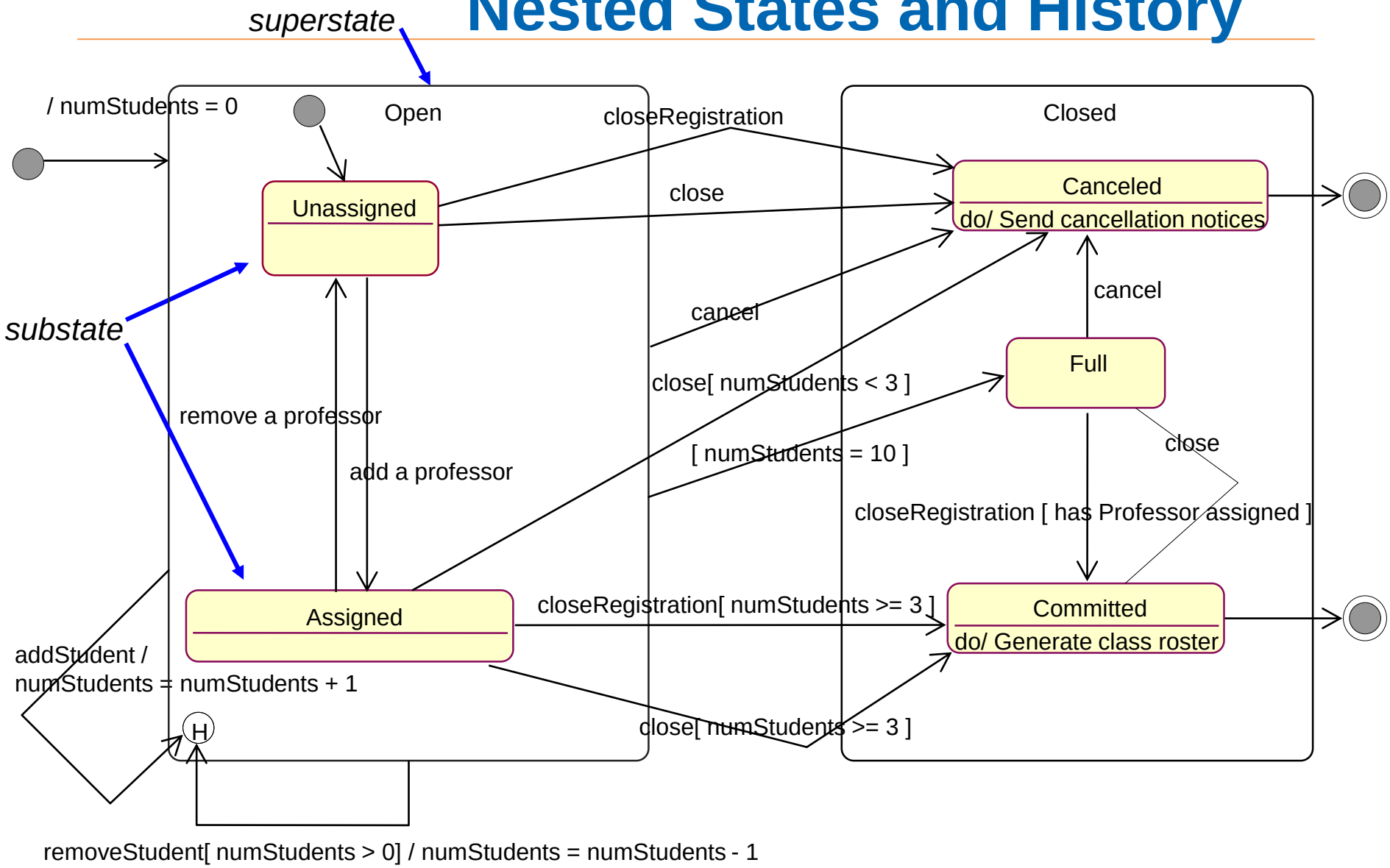
- Entry
  - Executed when the state is entered
- Do
  - Ongoing execution
- Exit
  - Executed when the state is exited



Example: State machine



Example: State Machine with Nested States and History

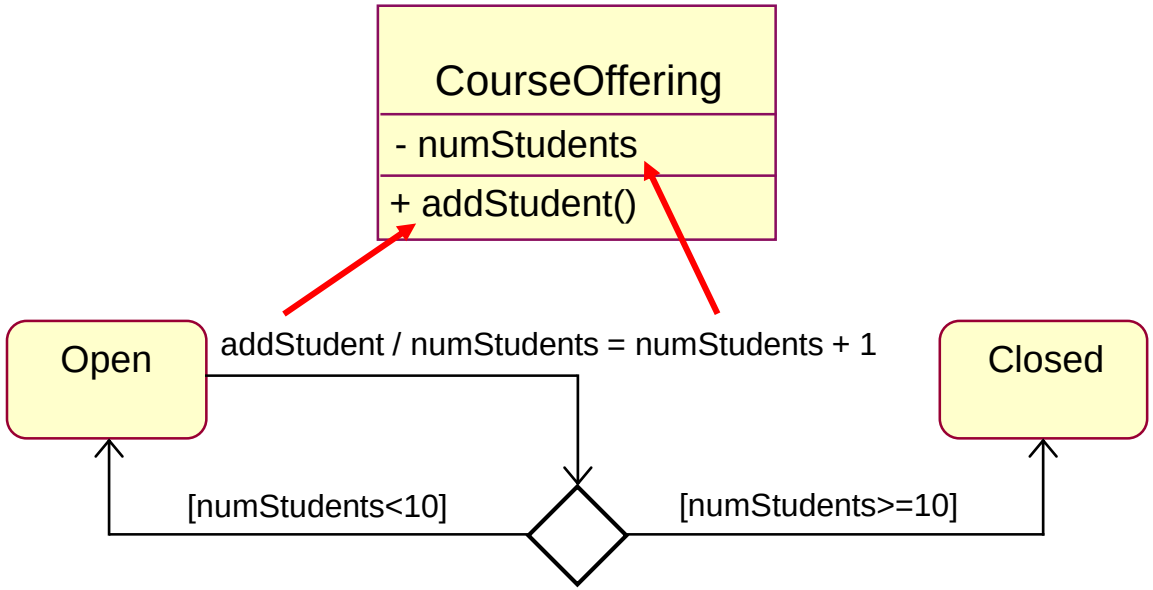


Which Objects Have Significant State?

- Objects whose role is clarified by state transitions
- Complex use cases that are state-controlled
- It is not necessary to model objects such as:
  - Objects with straightforward mapping to implementation
  - Objects that are not state-controlled
  - Objects with only one computational state

How Do State Machines Map to the Rest of the Model?

- Events may map to operations
- Methods should be updated with state-specific information
- States are often represented using attributes
  - This serves as input into the "*Define Attributes*" step



State table example

	In preparation	Postponed	Accepted	Placed	Back-Ordered	Fulfilled	Canceled
In Preparation	no	user saves incomplete request	system accepts valid request	no	no	no	no
Postponed	user retrieves incomplete request	no	no	no	no	no	no
Accepted	no	no	no	buyer places order with vendor	no	chemical stockroom fills request	requester cancels request
Placed	no	no	no	no	vendor places chemical on back-order	chemical received from vendor	buyer cancels vendor order
Back-Ordered	no	no	no	no	no	chemical received from vendor	buyer cancels vendor order
Fulfilled	no	no	no	no	no	no	no
Canceled	no	no	no	no	no	no	no

FIGURE 12-4 State table for a chemical request in the Chemical Tracking System.

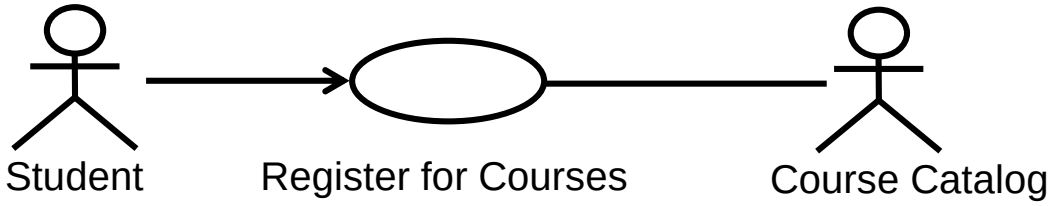
Several diagrams for self-studying

- Dialog map
- Decision tables and decision trees
- Event-response tables

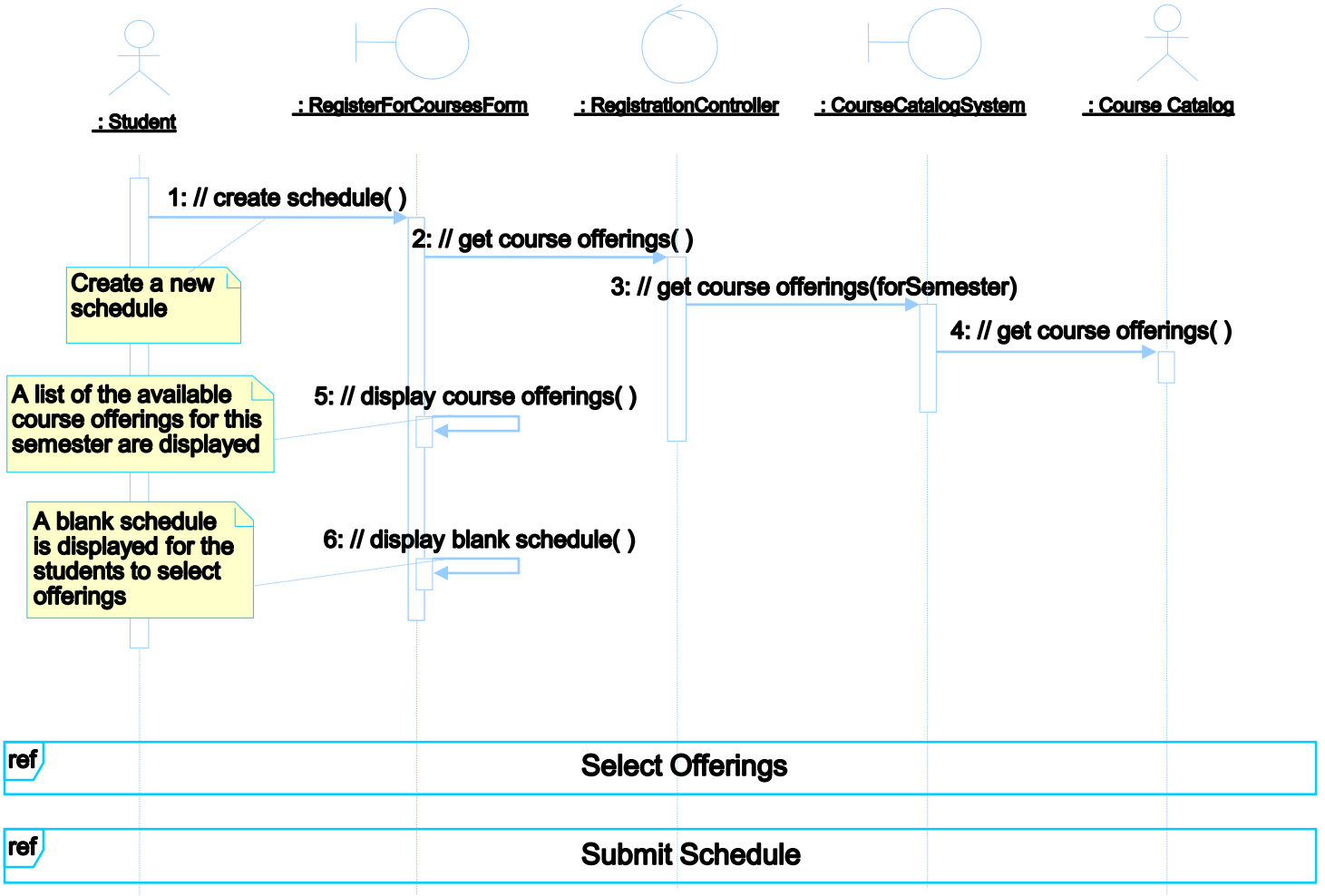
A few words about UML diagrams

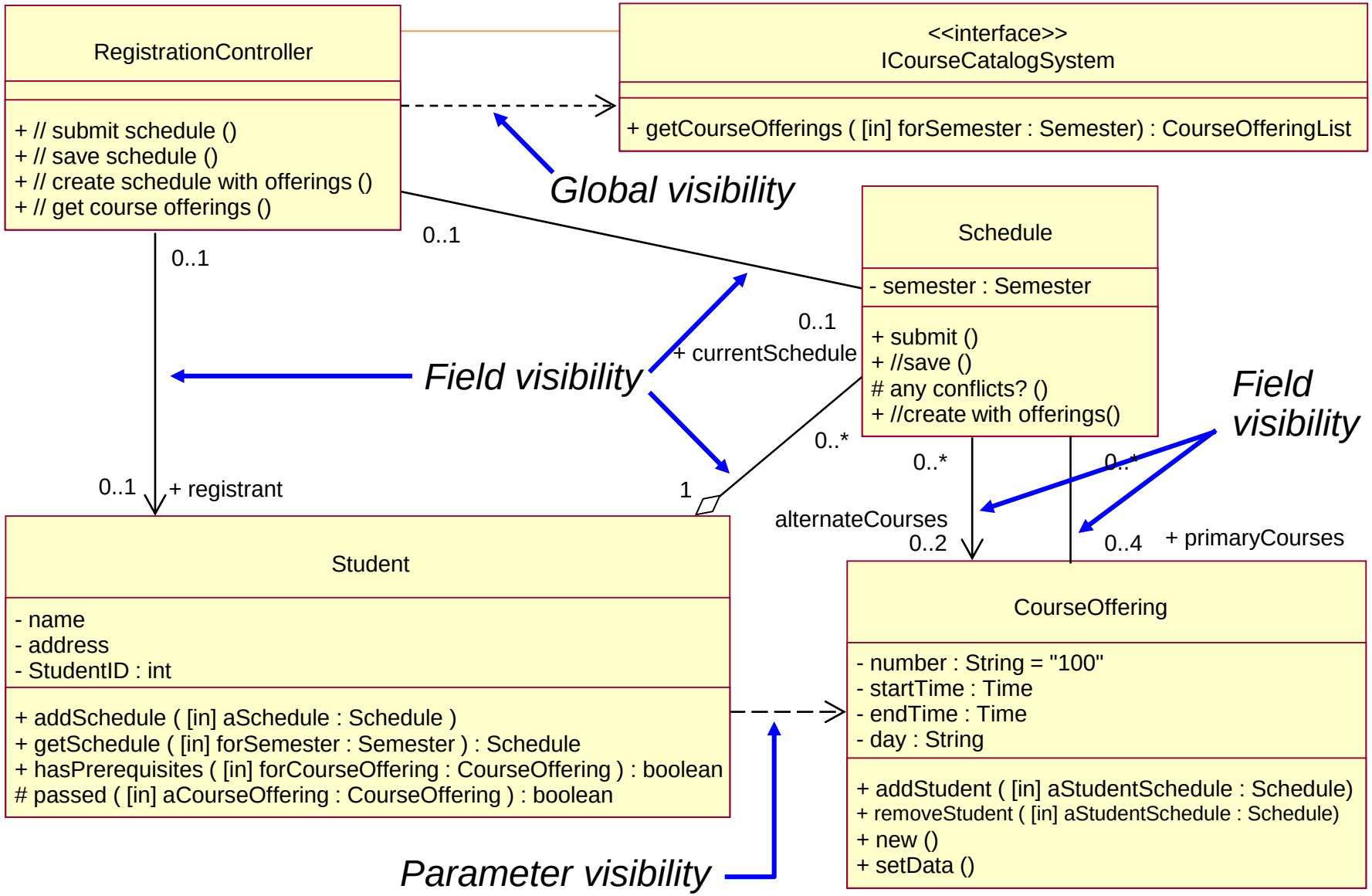
- UML diagrams: representation requirements
  - Why ?
  - When?
  - What ?

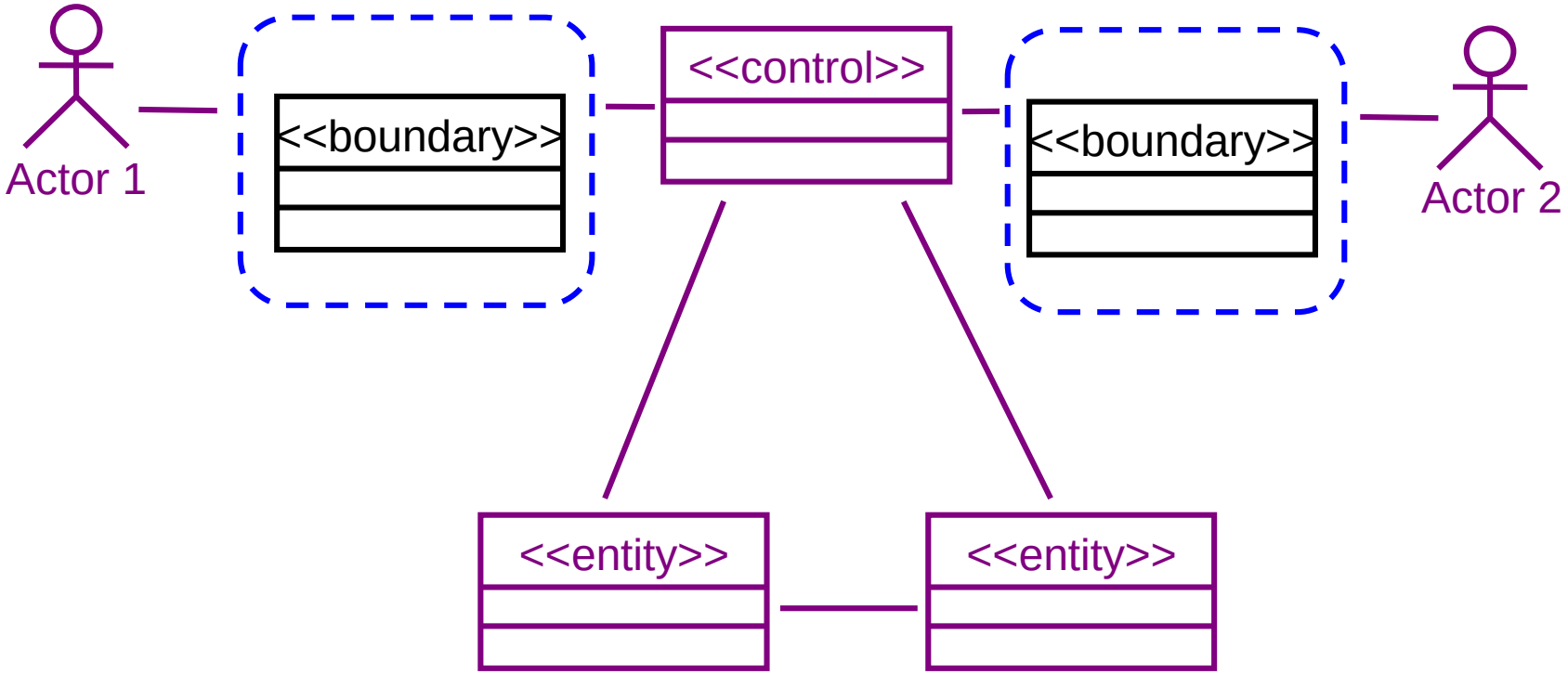
Use case example



Example: Sequence Diagram



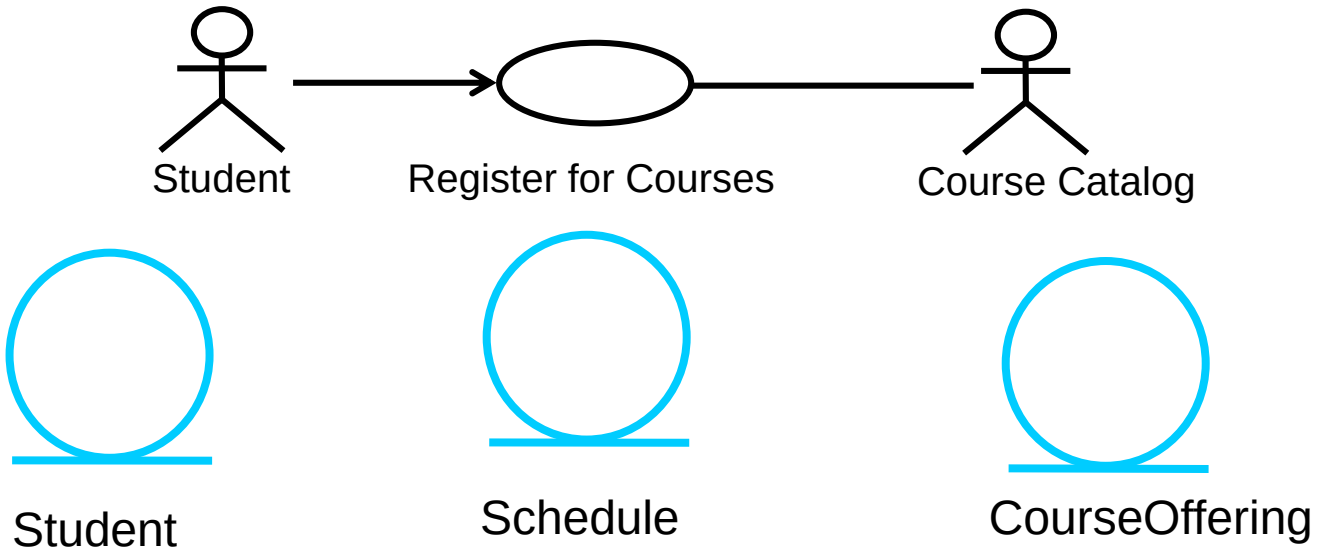

RegistrationController
+ // submit schedule ()
+ // save schedule ()
+ // create schedule with offerings ()
+ // get course offerings ()
<<interface>>
ICourseCatalogSystem
+ getCourseOfferings ([in] forSemester : Semester) : CourseOfferingList
Global visibility
0..1
0..1
Schedule
- semester : Semester
+ submit ()
+ //save ()
# any conflicts? ()
+ //create with offerings()
0..1
+ currentSchedule
Field visibility
Field visibility
0..*
0..*
0..*
0..1
+ registrant
1
alternateCourses
0..2
0..4
+ primaryCourses
Student
- name
- address
- StudentID : int
+ addSchedule ([in] aSchedule : Schedule)
+ getSchedule ([in] forSemester : Semester) : Schedule
+ hasPrerequisites ([in] forCourseOffering : CourseOffering) : boolean
# passed ([in] aCourseOffering : CourseOffering) : boolean
CourseOffering
- number : String = "100"
- startTime : Time
- endTime : Time
- day : String
+ addStudent ([in] aStudentSchedule : Schedule)
+ removeStudent ([in] aStudentSchedule : Schedule)
+ new ()
+ setData ()
Parameter visibility



Model interaction between the system and its environment.

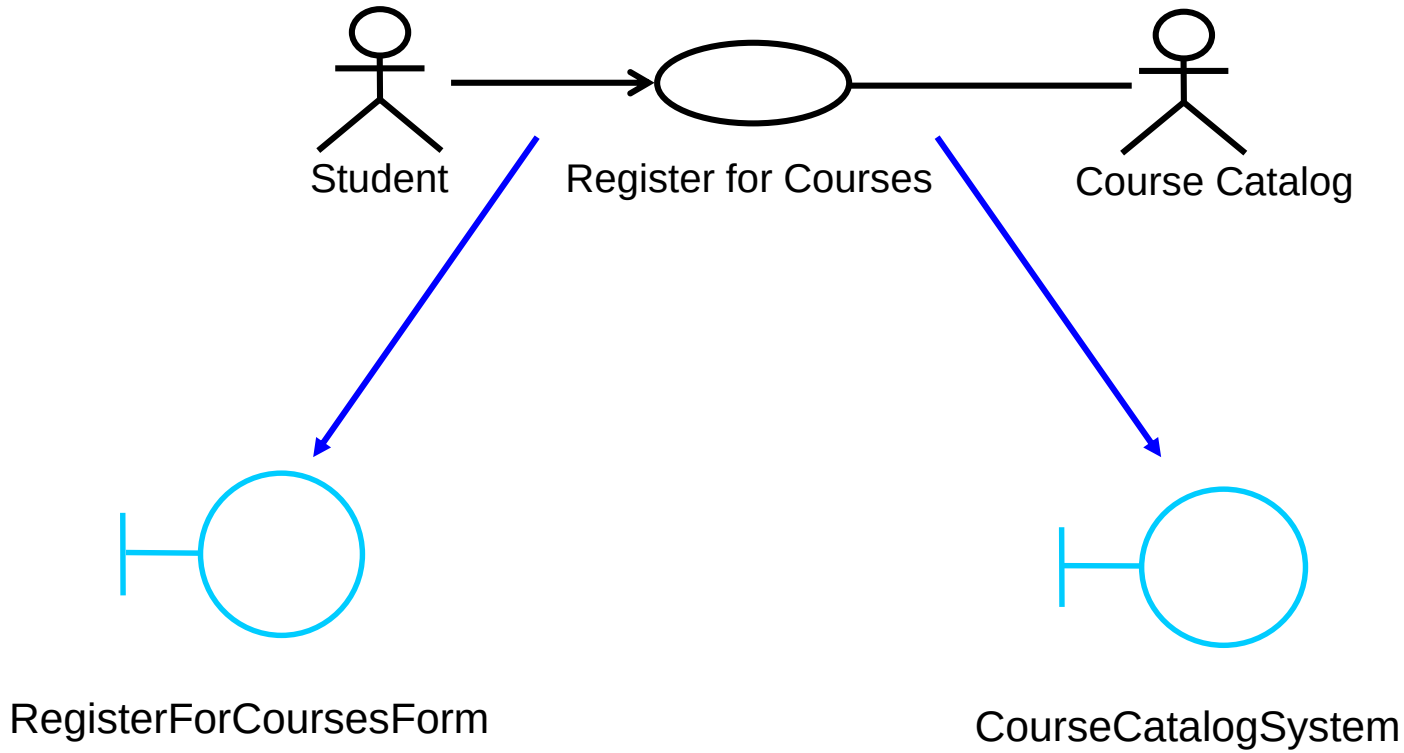
Example: Candidate Entity Classes

- Register for Courses (Create Schedule)



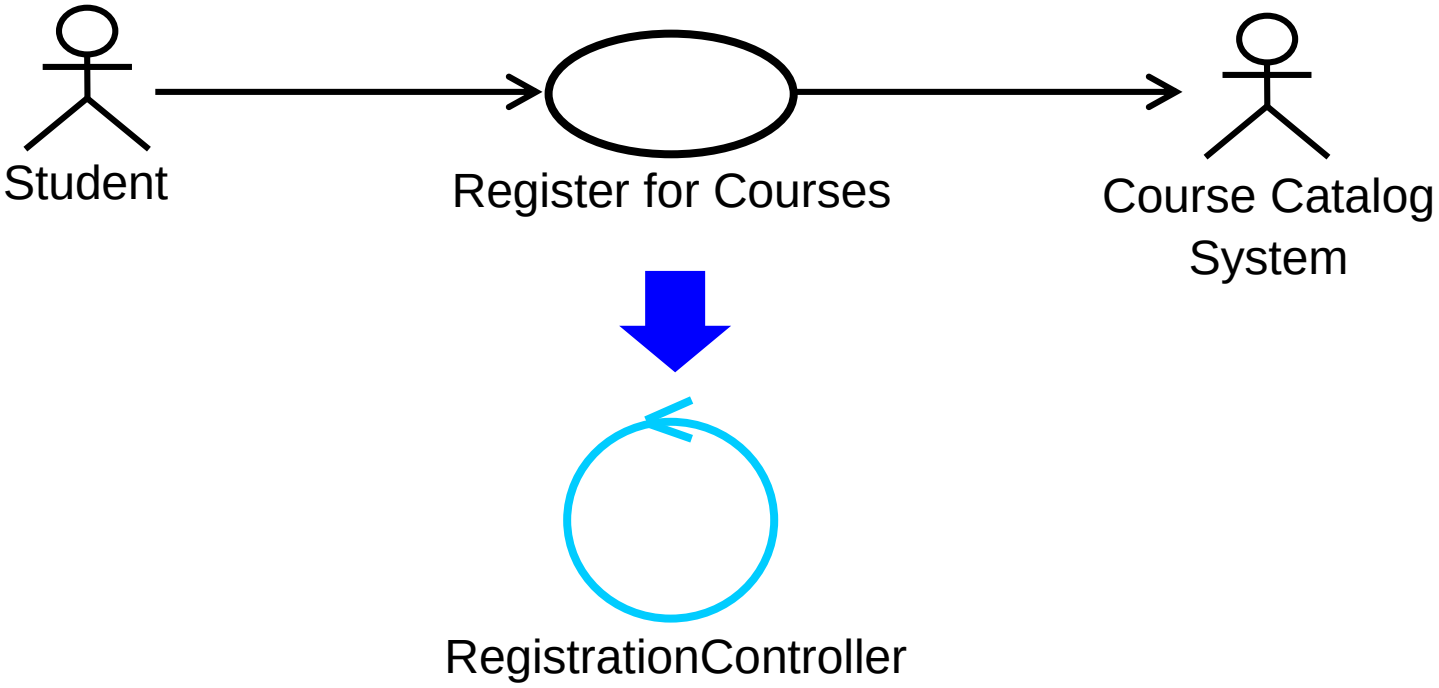
Example: Finding Boundary Classes

- One boundary class per actor/use case pair



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



Modeling on agile projects

- The difference in how traditional and agile projects perform modeling is related to when the models are created and the level of detail in them
- The key point in using analysis models on agile projects—or really, on any project—is to focus on creating only the models you need, only when you need them, and only to the level of detail you need to make sure project stakeholders adequately understand the requirements.