COMPUTER SCIENCE

15 - Requirements Analysis

Rick Adrion

COMPUTER SCIENCE

But first, let's discuss Project 2

- Goal - Develop a Software Requirements Specification using the RUP template
  - Vision Document
  - SRS
    - Introduction Section 1: Purpose, Scope, Definitions, Acronyms and Abbreviations, and provide References
    - Overall Description Section 2: a list of names and brief descriptions of all use cases and actors, along with applicable diagrams and relationships
    - In Section 3, for each use case diagram in Section 2 define a use-case report, making sure that each feature or requirement is clearly labeled and traceable to the Vision document.
    - Appendices, including: a) Table of contents, b) Index, and c) use-case storyboards or user-interface prototypes, if needed.
- Process
  - Develop questionnaires
  - Plan and carry out interviews
  - For selected subset, define Vision Document
  - For selected subset, define Use-Cases
  - Complete the SRS

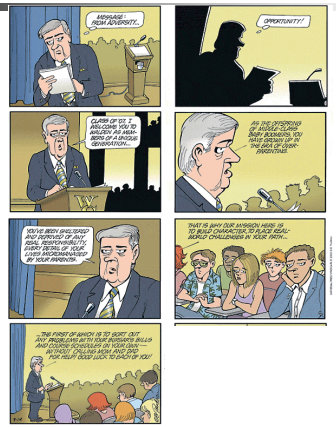
COMPUTER SCIENCE

How shall we do this?

- Pick the subset
  - Common or different for each group?
  - Project 2 prelim: each group email me the subset
- Develop questionnaires
  - At present, 4 groups have "registration & records" and one has "bursar"
  - Stakeholders to be interviewed
    - Department staff & associate chair
    - OIT
    - Registrar
    - Bursar
    - Yourselves
  - Sample "portal" questions/interview outline
- Plan and carry out interviews
  - 30 minutes/stakeholder-group
  - In class -- 1st weeks of November?
  - At another scheduled time
- For selected subset, define Vision Document
  - Assign one member of the group for this?
- For selected subset, define Use-Cases
  - Assign rest of the group for this?
- Complete the SRS

COMPUTER SCIENCE

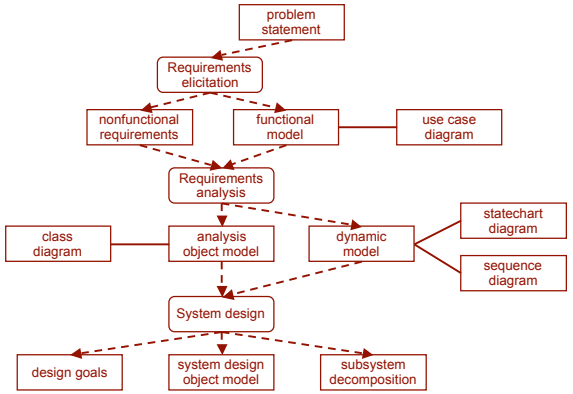
SIS



Credits [Reuse is good]

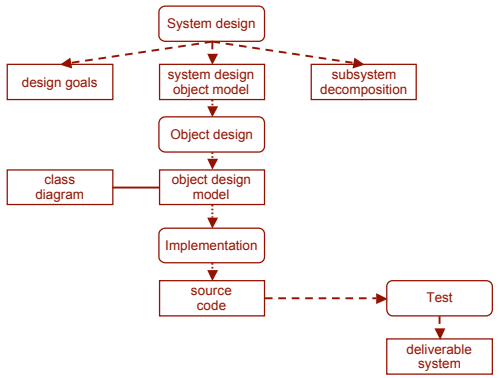
- Many of the following are derived from:
  - Kenneth M. Anderson © University of Colorado, 2003
  - Leszek Maciaszek **Requirements Analysis and System Design** © Addison Wesley, 2000
  - Bernard Bruegge and Allan Dutoit, **Object-Oriented Software Engineering (2nd Edition)** © Prentice Hall, 2003

O-O Sytem Development



adapted from Bruegge/Dutoit O-O SW Engr

O-O System Development



adapted from Bruegge/Dutoit O-O SW Engr

Object Modeling

- Main goal: Find the important abstractions
- What happens if we find the wrong abstractions?
  - Iterate and correct the model
- Steps during object modeling
  - Class identification
    - Based on the fundamental assumption that we can find abstractions
  - Find the attributes
  - Find the associations between classes
  - Find the methods
- Order of steps
  - Goal: get the desired abstractions
  - Order of steps secondary, only a heuristic
  - Iteration is important

Class Identification

- Identify the boundaries of the system
- Identify the important entities in the system
- Class identification is crucial to object-oriented modeling
- Approaches:
  - find the classes for a new software system (Forward Engineering)
  - identify the classes in an existing system (Reverse Engineering)

An ancient problem

- Objects are not just found by taking a picture of a scene or domain
- The application domain has to be analyzed.
- Depending on the purpose of the system different objects might be found
  - How can we identify the purpose of a system?
  - Scenarios and use cases
- Another important problem: Define system boundary.
  - What object is inside, what object is outside?

Pieces of an Object Model

- Classes
- Associations (Relations)
  - Part of- Hierarchy (Aggregation)
  - Kind of-Hierarchy (Generalization)
- Attributes
  - Detection of attributes
  - Application specific
  - Attributes in one system can be classes in another system
  - Turning attributes to classes
- Methods
  - Detection of methods
  - Generic methods: General world knowledge, design patterns
  - Domain Methods: Dynamic model, Functional model

Object vs Class

- Object (instance): Exactly one thing
  - The lecture on September 7 on Software Engineering from 9:05 -10:20
- A class describes a group of objects with similar properties
  - Author, Corrosion, Work order
- Object diagram: A graphic notation for modeling objects, classes and their relationships ("associations"):
  - Class diagram: Template for describing many instances of data. Useful for taxonomies, patters, schemata...
  - Instance diagram: A particular set of objects relating to each other. Useful for discussing scenarios, test cases and examples

Operation, Signature or Method?

- ***Operation***: A function or transformation applied to objects in a class. All objects in a class share the same operations *(Analysis Phase)*
- ***Signature***: Number & types of arguments, type of result value. All methods of a class have the same signature *(Object Design Phase)*
- ***Method***: Implementation of an operation for a class *(Implementation Phase)*

Polymorphic operation: The same operation applies to many different classes.

Workorder
File_name: String Size_in_bytes: integer Last_update: date Stickies: array[max]
print() delete() open() close() write() read()

Discovering Classes

- Learn about problem domain: Observe your client
- Apply general world knowledge and intuition
- Take the flow of events and find participating objects in use cases
- Apply design patterns
- Try to establish a taxonomy
- Do a textual analysis of scenario or flow of events
- Four Approaches(discussed last time)
  - Noun Phrase Approach
  - Common Class Patterns
  - Use Case Driven
  - CRC (Class-Responsibility-Collaboration)

Mapping parts of speech

Part of speech	*Model component*	*Example*
Proper noun	object	Jim Smith
Improper noun	class	Toy, doll
Doing verb	method	Buy, recommend
being verb	inheritance	is-a (kind-of)
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	3 years old
transitive verb	method	enter
intransitive verb	method (event)	depends on

Abbot 1983

Maciaszek's Guidelines

- Each class must have a statement of purpose in the system
- Each class is a template for a set of objects
  - avoid singleton classes
- Each class must house a set of attributes
- Each class should be distinguished from an attribute
  - e.g. Color may be an attribute of a Car class, but may be needed as a full class in a paint program
- Each class houses a set of operations that represents the interface of the class
  - operations can be derived from the statement of purpose

University Enrolment - Maciaszek

- A course can be part of any number of majors
- Each major specifies minimum total credits required
- Students may combine course offerings into programs of study suited to their individual needs and leading to the degree/major in which enrolled
- A student's choice of courses may be restricted by timetable clashes and by limitations on the number of students who can be enrolled in the current course offering.
- A student's proposed program of study is entered on on-line enrollment system via the SPIRE interface

University Enrolment - Maciaszek

- The system checks the program's consistency and reports any problems
- The problems need to be resolved with the help of an academic adviser
- The final program of study is subject to academic approval by the Department head (or delegate) and it is then forwarded to the Registrar
- The student's academic record to be available on demand
- The record to include information about the student's grades in each course that the student enrolled in (and has not withdrawn without penalty)
- Each course has one professor in charge of a course, but additional professors (inc instructors) may also teach in it
- There may be a different professor in charge of a course each semester
- There may be professor for each course each semester

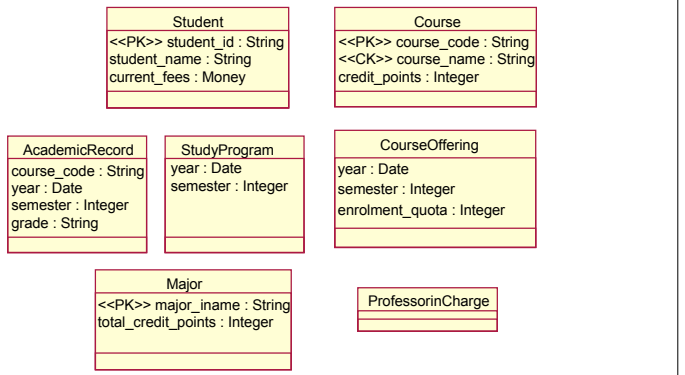
University Enrolment - Maciaszek

Relevant classes	Fuzzy classes
Course	CompulsoryCourse
Major	ElectiveCourse
Student	
CourseOffering	
	<-StudyProgram
Professor -> ProfessorinCharge	
AcademicRecord	

Specifying Attributes

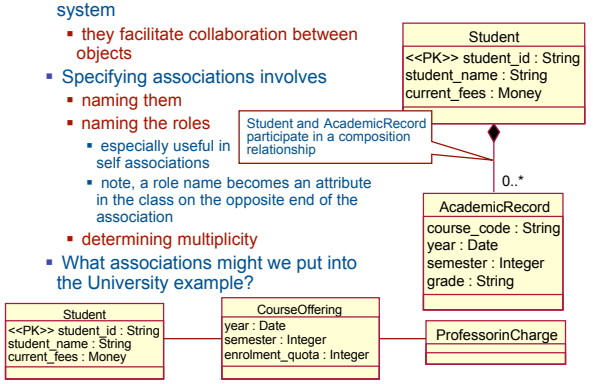
- Attributes are specified in parallel with classes
  - initial set of attributes will be “obvious”
  - important to initially select attributes that help to determine the states of the class
  - additional attributes can be added in subsequent iterations

Maciaszek Solution



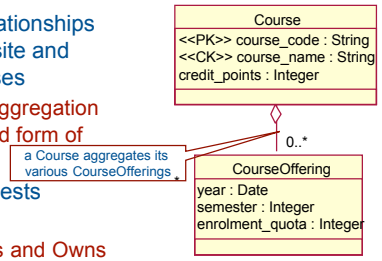
Specifying Associations

- Associations connect objects in the system
  - they facilitate collaboration between objects
- Specifying associations involves
  - naming them
  - naming the roles
    - especially useful in self associations
    - note, a role name becomes an attribute in the class on the opposite end of the association
  - determining multiplicity
- What associations might we put into the University example?

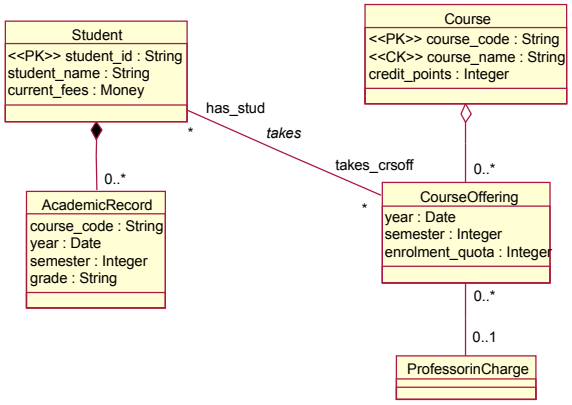


Specifying Aggregation/Composition

- "Whole-part" relationships between composite and component classes
  - UML models aggregation as a constrained form of association
- Maciaszek suggests additional power
  - ExclusiveOwns and Owns
  - Has and Member
- Litmus test: "has" or "is-part-of" is needed to explain relationship

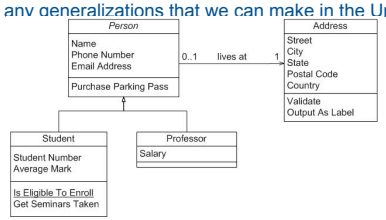


Maciaszek Solution



Specifying Generalizations

- Looking for common features among classes
  - Move common features up a class hierarchy and specialized features down
- Apart from inheritance, generalization has two objectives
  - substitutability and polymorphism
- Litmus test: "can be" and "is-a-kind-of" required to explain relationship
- Are there any generalizations that we can make in the University example?



Behavior Specification

- Behavior of a system, as it appears to an outside user, is specified in use cases
  - During analysis, use cases specify "what" a system needs to do (**not "how"**)
- Use cases require computations to be performed
- Computations are divided into activities
  - can be modeled using activity diagrams
- Activities are carried out by interacting objects
  - interactions are modeled using sequence diagrams

Behavior Specification

- Provide an operational view of the system
- Main Tasks
  - Define use cases and determine which classes are used to execute a use case
  - Identify operations on classes
- State specifications in analysis typically reveal entity classes
- Behavior specifications will often reveal controller classes and boundary classes (user interface classes)

Maciaszek's Take: Use Cases

- A use case represents
  - a complete piece of functionality
    - including main and alternate flows of logic
  - a piece of externally visible functionality
  - an orthogonal piece of functionality
    - use cases can share objects but execute independently from each other
  - a piece of functionality initiated by an actor
  - a piece of functionality that delivers value to an actor

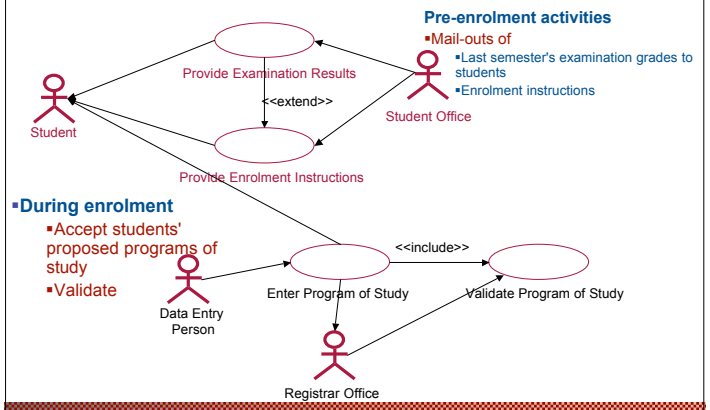
Finding Use Cases

- Use cases are discovered via analysis of
  - requirements in the requirements documents
  - actors and their purpose
- Jacobson suggests asking the following questions concerning actors to help identify use cases
  - What are the main tasks performed by the actor
  - Will an actor access or modify information in the system
  - Will an actor inform the system about changes in other systems?
  - Should an actor be informed about unexpected changes in the system?

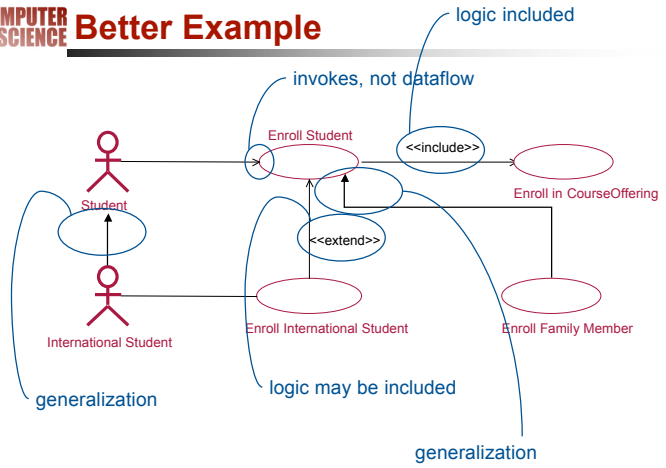
Use Case Relationships

- Association
  - a communication path
- Generalization
  - a specialized use case can change any aspect of the base use case
- Include
  - directly includes steps of another use case
- Extend
  - customize an extension point

Example 4.12



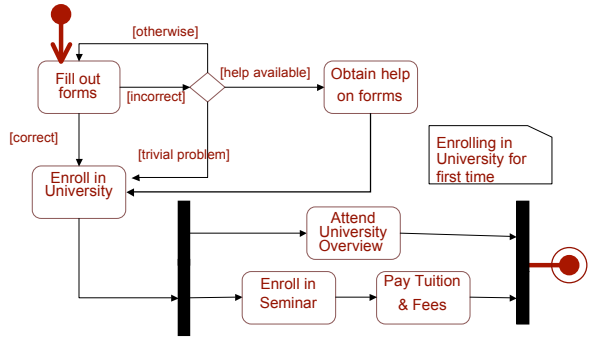
Better Example



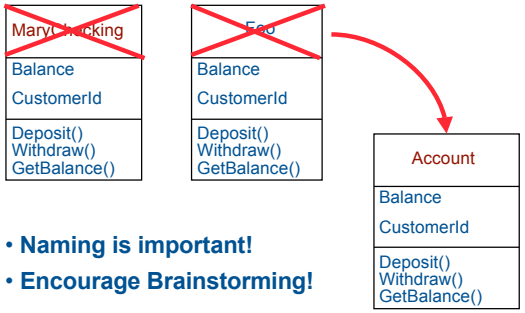
Modeling Activities

- Activities capture the flow of logic within a system
  - both sequential and parallel control can be modeled
- Since activities do not reference classes, they can be created without the need for a class diagram
- Most often used to graphically represent the steps of a use case
  - can show main flow and extensions at once
- Activities are best discovered by analyzing the action steps of use cases, with verbs indicating candidate activities

University enrollment example

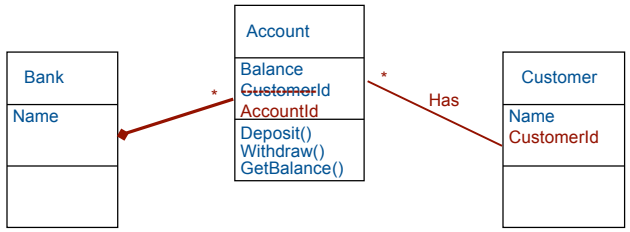


Object Modeling in Practice



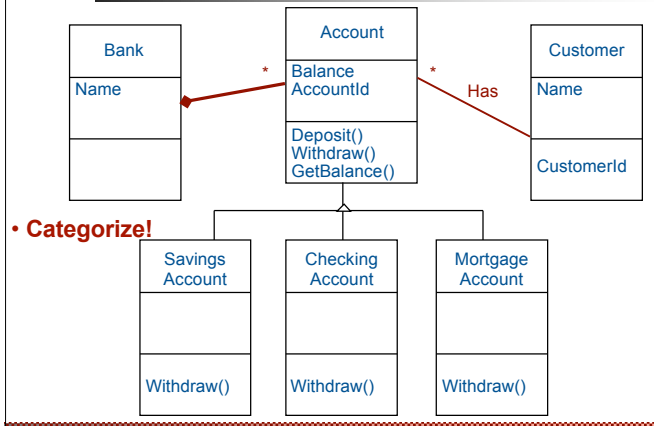
- **Naming is important!**
- **Encourage Brainstorming!**

Object Modeling in Practice



- **Find New Objects**
- **Iterate on Names, Attributes and Methods**
- **Find Associations between Objects**
- **Label the associations**
- **Determine the multiplicity of the associations**

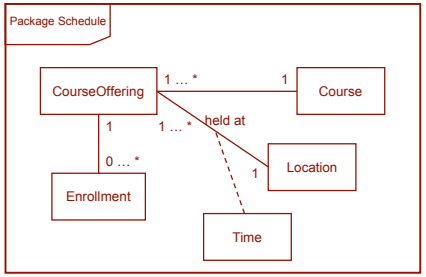
Object Modeling in Practice:



- **Categorize!**

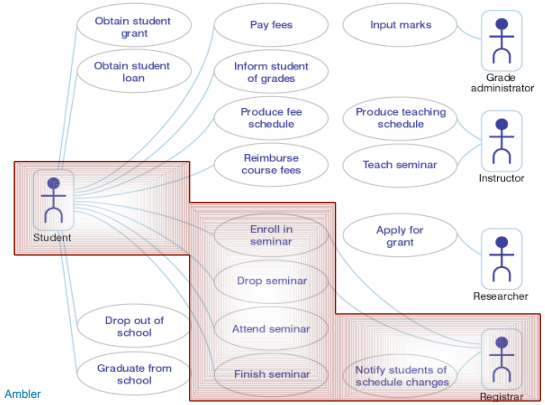
Using packages

- Avoid Ravioli Models -- don't put too many classes into the same package:
  - Rule: 7+-2 (or even 5+-2)



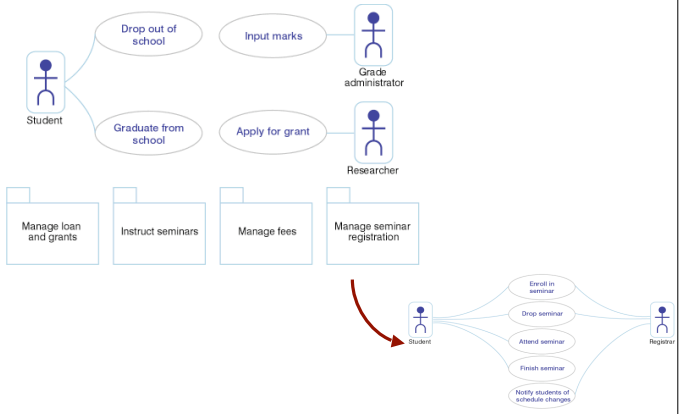
Using packages

Using packages with Use Cases



Scott W. Ambler

Using packages with Use Cases



Heuristics

- Explicitly schedule meetings for object identification
- Try to differentiate between entity, boundary and control objects
- Find associations and their multiplicity
  - Unusual multiplicities usually lead to new objects or categories
- Identify Aggregation
- Identify Inheritance: Look for a Taxonomy, Categorize
- Allow time for brainstorming , **Iterate, iterate**

Modeling Interactions

- One level of abstraction below activities
- Interaction models require at least one iteration of state specification to be performed
  - Since we need to have classes to which each object belongs
- Interaction diagrams do not model object state changes; however they may show the actions that lead to a state change
- Interactions can help determine operations; any message to an object in a interaction must be handled by an operation (actually a method)
  - Recall that a method implements an operation; indeed there may be many methods available for a single operation

Dynamic Modeling with UML

- Diagrams for dynamic modeling
  - Interaction diagrams describe the dynamic behavior between objects
  - Statecharts describe the dynamic behavior of a single object
- Interaction diagrams
  - Sequence Diagram:
    - Dynamic behavior of a set of objects arranged in time sequence.
    - Good for real-time specifications and complex scenarios
  - Collaboration Diagram :
    - Shows the relationship among objects. Does not show time
- State Charts:
  - A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).
- Activity Diagram:
  - Special type of statechart where all states are action states

Start with Flow of Events

- Get events from Use Case
- What is an Event?
  - something that happens at a point in time
- Relation of events to each other:
  - causally related: Before, after,
  - causally unrelated: concurrent
- An event sends information from one object to another
- Events can be grouped in event classes with a hierarchical structure.
- Event is often used in two ways:
  - Instance of an event class
  - Attribute of an event class

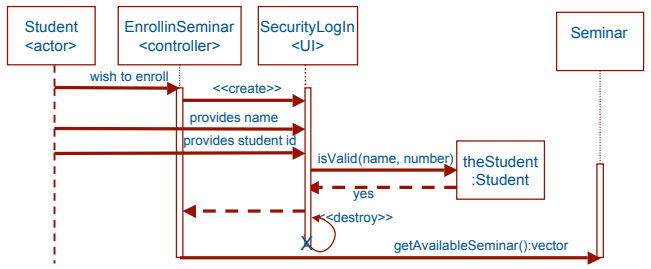
Specifying message sequences

- Useful to distinguish between
  - signals
    - asynchronous inter-object communication
    - often shown with "half-arrow notation"
  - Calls
    - synchronous inter-object communication control returns to caller (usually)

Sequence Diagram

- From the flow of events in the use case or scenario proceed to the sequence diagram
- A sequence diagram is a graphical description of objects participating in a use case or scenario using a DAG notation
- Relation to object identification:
  - Objects/classes have already been identified during object modeling
  - Objects are identified as a result of dynamic modeling
- Heuristic:
  - An event always has a sender and a receiver. Find them for each event => These are the objects participating in the use case

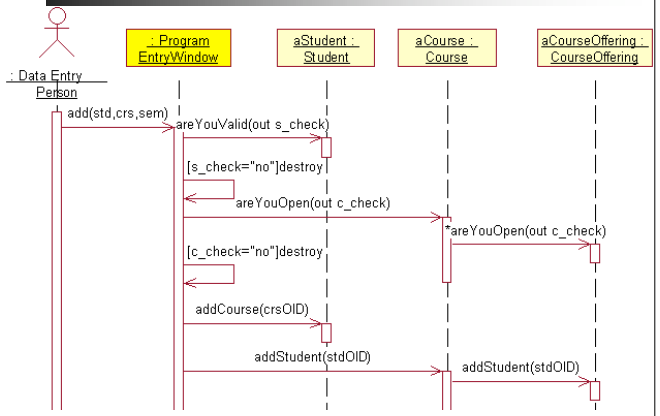
Sequence Diagram



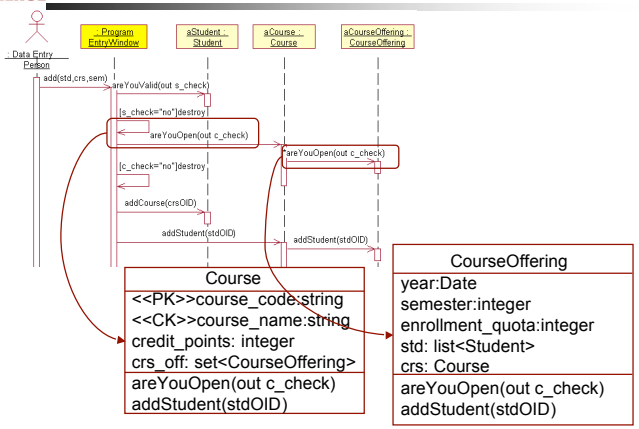
Defining Operations

- A public interface of a class consists of operations that offer services to entities external to the class
  - operations are best discovered from sequence diagrams, since every message must be serviced by an operation
- Other operations can be found using the CRUD (create, read, update, delete) paradigm; classes need to provide these services regardless of their domain specific functionality

Example 4.17 – Maciaszek



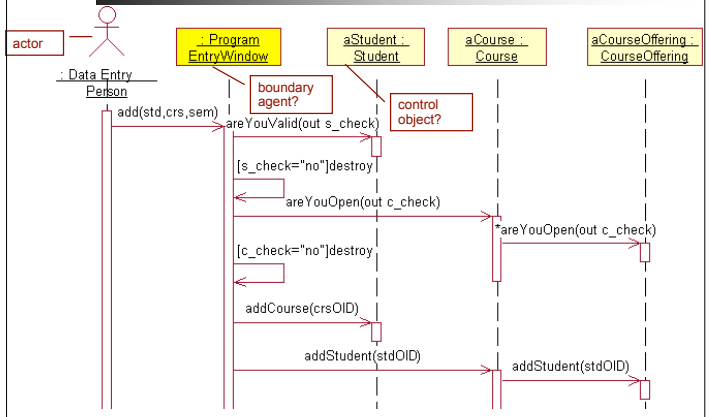
Example 4.17 – Maciaszek



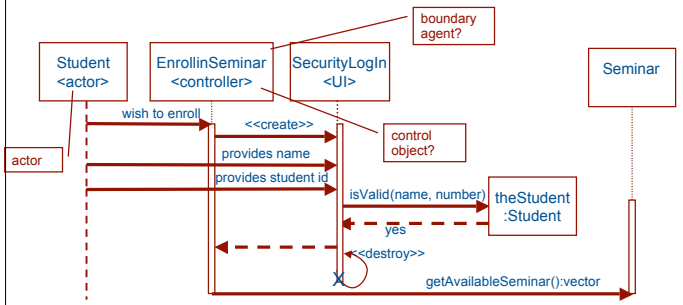
Heuristics for Sequence Diagrams

- Layout:
  - 1st column: Should correspond to the actor who initiated the use case
  - 2nd column: Should be a boundary object
  - 3rd column: Should be the control object that manages the rest of the use case
- Creation:
  - Control objects are created at the initiation of a use case
  - Boundary objects are created by control objects
- Access:
  - Entity objects are accessed by control and boundary objects,
  - Entity objects should never call boundary or control objects: This makes it easier to share entity objects across use cases and makes entity objects resilient against technology-induced changes in boundary objects.

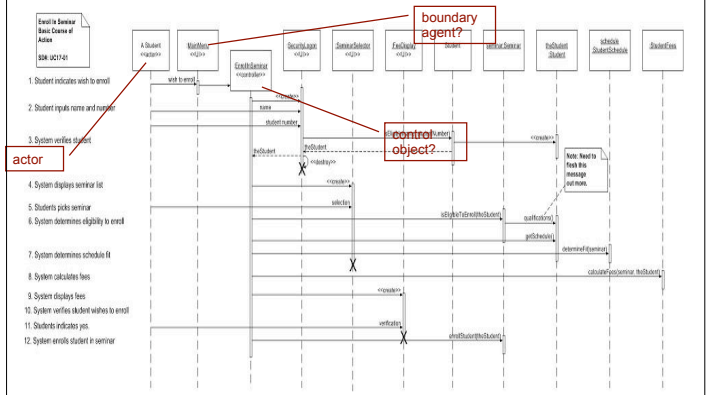
Is this a good sequence diagram?



Perhaps better ….

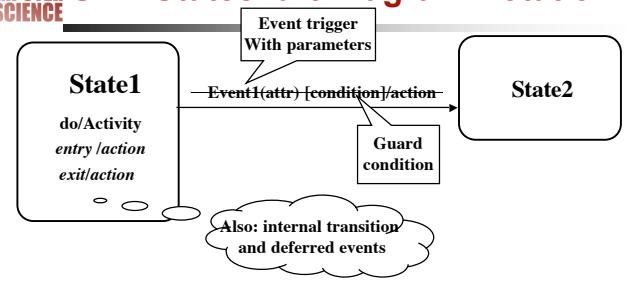


Perhaps even better ….



Fix after this!

UML Statechart Diagram Notation



- Notation based on work by Harel
  - Added are a few object-oriented modifications
- A UML statechart diagram can be mapped into a finite state machine

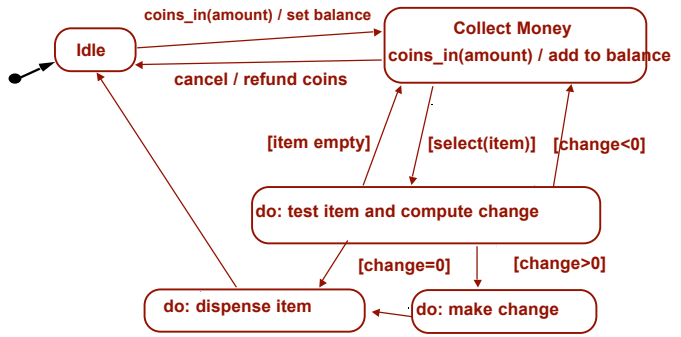
Statechart Diagrams

- Graph whose nodes are states and whose directed arcs are transitions labeled by event names.
- Distinguish between two types of operations:
  - Activity: Operation that takes time to complete
    - associated with states
  - Action: Instantaneous operation
    - associated with events
    - associated with states (reduces drawing complexity): Entry, Exit, Internal Action
- A statechart diagram relates events and states for one class
  - An object model with a set of objects has a set of state diagrams

State

- An abstraction of the attribute of a class
  - State is the aggregation of several attributes a class
- Basically an equivalence class of all those attribute values and links that do no need to be distinguished as far as the control structure of the system is concerned
  - Example: State of a bank
    - A bank is either solvent or insolvent
- State has duration

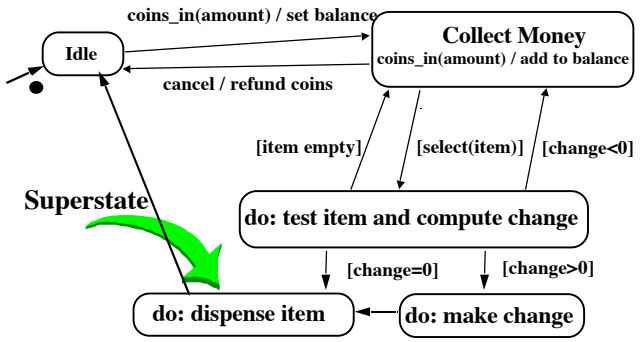
Example of a StateChart Diagram



Nested State Diagram

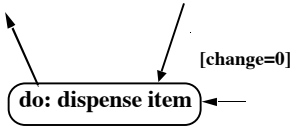
- Activities in states are composite items denoting other lower-level state diagrams
- A lower-level state diagram corresponds to a sequency of lower-level states and events that are invisible in the higher-level diagram.
- Sets of substates in a nested state diagram denoting a superstate are enclosed by a large rounded box, also called contour.

Example of a Nested Statechart Diagram

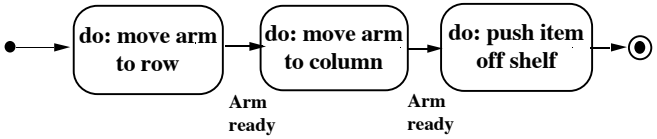


Expanding activity "do:dispense item"

'Dispense item' as an atomic activity:



'Dispense item' as a composite activity:



Superstates

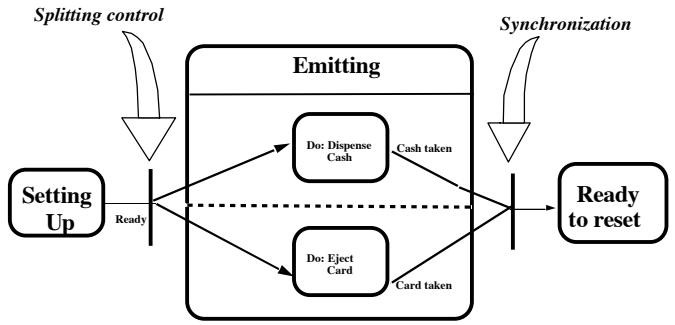
- Goal:
  - Avoid spaghetti models
  - Reduce the number of lines in a state diagram
- Transitions from other states to the superstate enter the first substate of the superstate.
- Transitions to other states from a superstate are inherited by all the substates (state inheritance)

Modeling Concurrency

Two types of concurrency

1. System concurrency
   - State of overall system as the aggregation of state diagrams, one for each object. Each state diagram is executing concurrently with the others.
2. Object concurrency
   - An object can be partitioned into subsets of states (attributes and links) such that each of them has its own subdiagram.
   - The state of the object consists of a set of states: one state from each subdiagram.
   - State diagrams are divided into subdiagrams by dotted lines.

Example of Concurrency within an Object



State Chart Diagram vs Sequence Diagram

- State chart diagrams help to identify:
  - *Changes* to objects over time
- Sequence diagrams help to identify
  - The *temporal relationship* of between objects over time
  - *Sequence of operations* as a response to one ore more events

Practical Tips for Dynamic Modeling

- Construct dynamic models only for classes with significant dynamic behavior
  - Avoid "analysis paralysis"
- Consider only relevant attributes
  - Use abstraction if necessary
- Look at the granularity of the application when deciding on actions and activities
- Reduce notational clutter
  - Try to put actions into state boxes (look for identical actions on events leading to the same state)