COMPUTER SCIENCE

10 - UML Overview

Rick Adrion

COMPUTER SCIENCE

What is use case modeling?

- use case model
  - a view of a system that emphasizes the behavior as it appears to outside users. A use case model partitions system functionality into transactions ('use cases') that are meaningful to users ('actors').

COMPUTER SCIENCE

Use-Case diagrams

- emphasis is on *what* a system does rather than *how*
- Use case diagrams are closely connected to scenarios
  - a **scenario** is an example of what happens when someone interacts with the system, e.g.,
    "A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot."
  - a **use case** is a summary of scenarios for a single task or goal
  - an **actor** is who or what initiates the events involved in that task

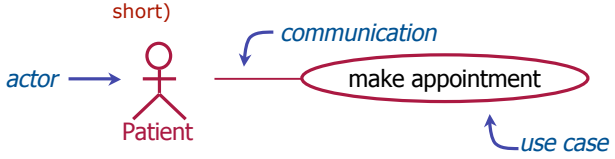
COMPUTER SCIENCE

Use Case Modeling: Core Elements

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

Example

- **Make Appointment**
  - use case for the medical clinic
    - actor is a **Patient**
    - connection between actor and use case is a **communication association** (or **communication** for short)



Use Case Modeling: Core Relationships

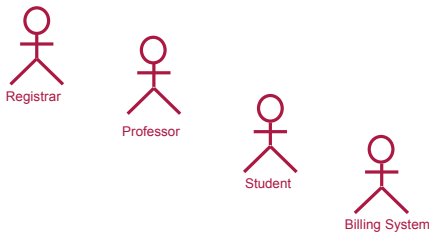
Construct	Description	Syntax
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	———
generalization	A taxonomic relationship between a more general use case and a more specific use case.	——→
extend	A relationship from an *extension* use case to a *base* use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	<<extend>> - - - - - - - - -→

An example

- The ESU University wants to computerize their registration system
  - The Registrar sets up the curriculum for a semester
    - One course may have multiple course offerings
  - Students select 4 primary courses and 2 alternate courses
  - Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
  - Students may use the system to add/drop courses for a period of time after registration
  - Professors use the system to receive their course offering rosters
  - Users of the registration system are assigned passwords which are used at logon validation

Actors

- An actor is someone or some thing that must interact with the system under development



Use Cases

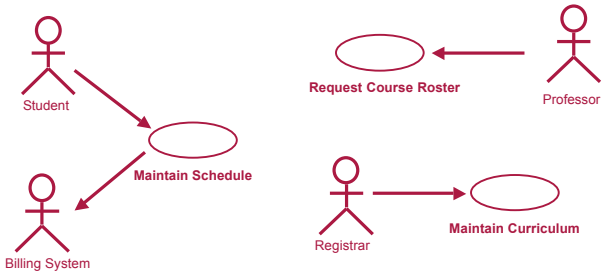
- A use case is a pattern of behavior the system exhibits
  - Each use case is a sequence of related transactions performed by an actor and the system in a dialogue
- Actors are examined to determine their needs
  - Registrar -- maintain the curriculum
  - Professor -- request roster
  - Student -- maintain schedule
  - Billing System -- receive billing information from registration

Documenting Use Cases

- A flow of events document is created for each use cases
  - Written from an actor point of view
- Details what the system must provide to the actor when the use cases is executed
- Typical contents
  - How the use case starts and ends
  - Normal flow of events
  - Alternate flow of events
  - Exceptional flow of events

Use Case Diagram

- Use case diagrams are created to visualize the relationships between actors and use cases



Maintain Curriculum Flow of Events

- This use case begins when the Registrar logs onto the Registration System and enters his/her password. The system verifies that the password is valid (E-1) and prompts the Registrar to select the current semester or a future semester (E-2). The Registrar enters the desired semester. The system prompts the Registrar to select the desired activity: ADD, DELETE, REVIEW, or QUIT.
- If the activity selected is ADD, the S-1: Add a Course subflow is performed.
- If the activity selected is DELETE, the S-2: Delete a Course subflow is performed.
- If the activity selected is REVIEW, the S-3: Review Curriculum subflow is performed.
- If the activity selected is QUIT, the use case ends.
- ...

Documenting use cases

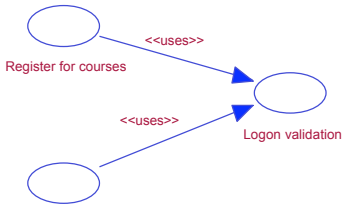
- **Brief Description**
- **Actors** involved
- **Preconditions** necessary for the use case to start
- **Detailed Description** of flow of events that includes:
  - **Main Flow** of events, that can be broken down to show:
    - **Subflows** of events (subflows can be further divided into smaller subflows to improve document readability)
  - **Alternative Flows** to define exceptional situations
- **Postconditions** that define the state of the system after the use case ends

Narrative use case specification

Use Case	Add a course to the curriculum
Brief Description	This use case allows a Registrar to enter a new course. …
Actors	`Registrar`
Preconditions	Registrar has a valid password (E-1), has selected a semester default or E-2), and has selected the `Add` (S-1) function at the system prompt
Main Flow	The system enters the `Add a Course` subflow
Alternative Flows	The `Registrar` activates the `Delete`, `Review`, or `Quit` functions
Postconditions	If the use case was successful, the Registrar has accessed the `Add a Course` function

Uses and Extends Relationships

- As the use cases are documented, other use case relationships may be discovered
  - A **uses** relationship shows behavior that is common to one or more use cases
  - An **extends** relationship shows optional behavior



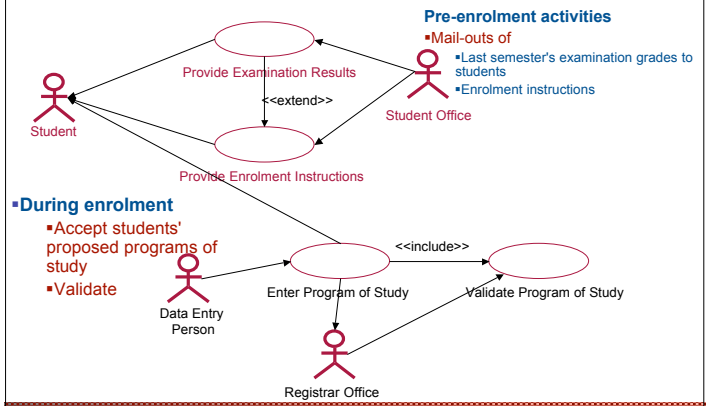
University Enrolment - Maciaszek

- The **university** offers
  - Undergraduate and postgraduate degrees
  - To full-time and part-time students
- The **university structure**
  - Divisions containing departments
  - Single division administers each degree
  - Degree may include courses from other divisions
- **University enrolment** system
  - Individually tailored programs of study
  - Prerequisite courses
  - Compulsory courses
  - Restrictions
    - Timetable clashes
    - Maximum class sizes, etc.

University Enrolment (cont)

- The system is required to
  - Assist in pre-enrolment activities
  - Handle the enrolment procedures
- **Pre-enrolment activities**
  - Mail-outs of
    - Last semester's examination grades to students
    - Enrolment instructions
- **During enrolment**
  - Accept students' proposed programs of study
  - Validate for prerequisites, timetable clashes, class sizes, special approvals, etc.
- Resolutions to some of the problems may require consultation with academic advisers or academics in charge of course offerings

Example 4.12



When to model use cases

- Model user requirements with use cases.
- Model test scenarios with use cases.
- If you are using a use-case driven method
  - start with use cases and derive your structural and behavioral models from it.
- If you are not using a use-case driven method
  - make sure that your use cases are consistent with your structural and behavioral models.

Use Case Modeling Tips

- Make sure that each use case describes a significant chunk of system usage that is understandable by both domain experts and programmers
- When defining use cases in text, use nouns and verbs accurately and consistently to help derive objects and messages for interaction diagrams (see Lecture 2)
- Factor out common usages that are required by multiple use cases
  - If the usage is required use <<include>>
  - If the base use case is complete and the usage may be optional, consider use <<extend>>
- A use case diagram should
  - contain only use cases at the same level of abstraction
  - include only actors who are required
- Large numbers of use cases should be organized into packages

Use Case Realizations

- The use case diagram presents an outside view of the system
- Interaction diagrams describe how use cases are realized as interactions among societies of objects
- Two types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

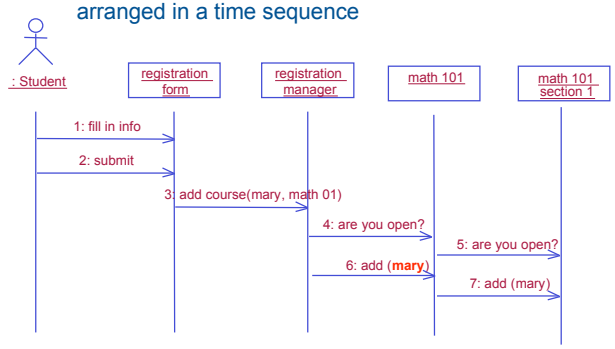
sequence diagram

- an interaction diagram that details how operations are carried out
  - what messages are sent and when
  - are organized according to time
  - time progresses as you go down the page
  - objects involved in the operation are listed from left to right according to when they take part in the message sequence.

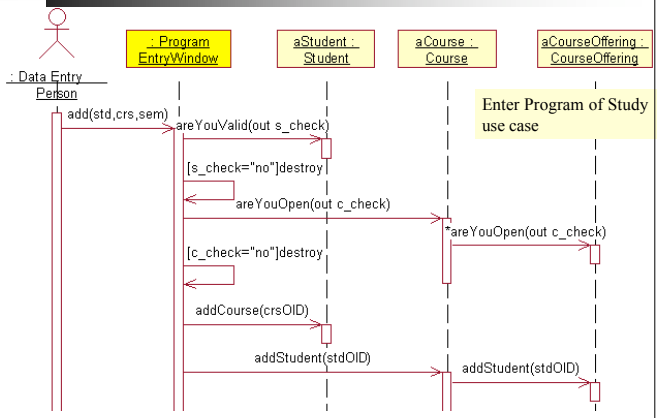
Symbol	Meaning
→	simple message which may be synchronous or asynchronous
- - →	simple message return (optional)
→	a synchronous message
→→	an asynchronous message

Sequence Diagram

- A sequence diagram displays object interactions arranged in a time sequence

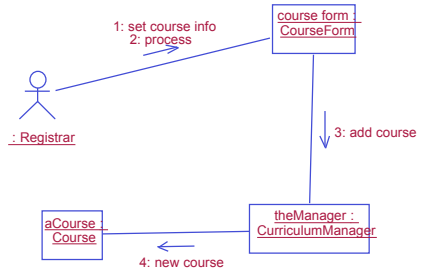


Example 4.17 – Maciaszek



Collaboration Diagrams

- also interaction diagrams
- focus on object roles instead of the times that messages are sent



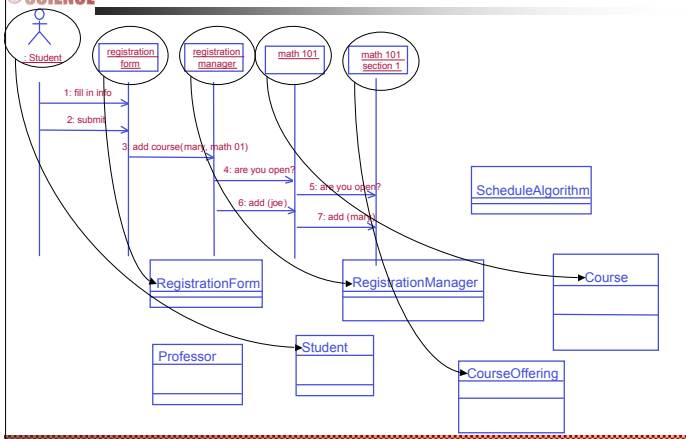
Class Modeling

- Captures **system state** – the function of the system's information content at a point in time
- Class modeling and use case modeling are typically conducted in parallel
- A **class diagram** shows the existence of classes and their relationships in the logical view of a system; in UML class diagrams elements include
  - Classes and their structure and behavior
  - Association, aggregation, generalization, dependency, and inheritance relationships
  - Multiplicity and navigation indicators
  - Role names

Classes

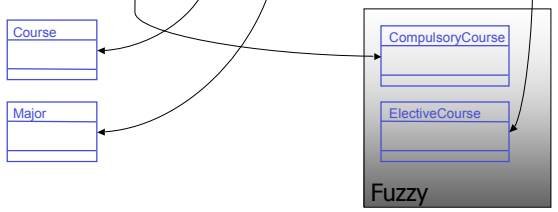
- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes may be found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
  - Naming standards should be created
  - e.g., all classes are singular nouns starting with a capital letter

Classes



Find Classes from requirements

- Consider Maciaszek's University Enrollment system:
  - each university major has a number of compulsory courses and a number of elective courses.



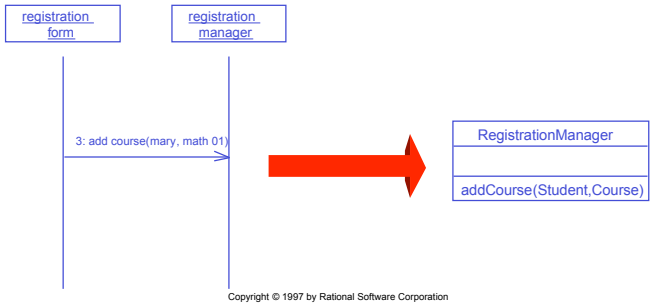
University Enrolment - Maciaszek

- More requirements:
  - 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

Relevant classes	Fuzzy classes
Course	CompulsoryCourse
Major	ElectiveCourse
Student	Sudyprogram
CourseOffering	

Operations

- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams



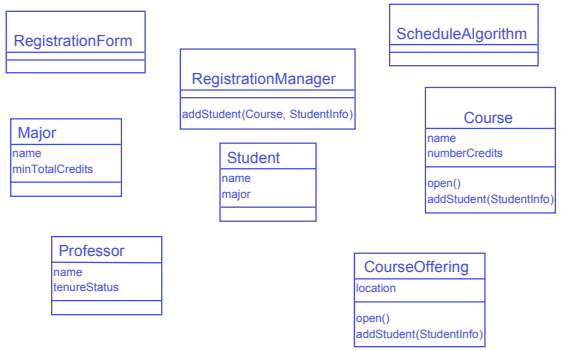
Attributes

- The structure of a class is represented by its attributes
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge
- More requirements
  - Each course offering has a number, location and time
  - Each course is at a given level and has a credit-hours value

Course
name
number
numberCredits

CourseOffering
number
location
time

Classes



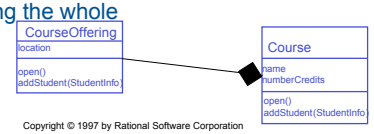
Relationships

- Relationships provide a pathway for communication between objects
- Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to "talk" there must be a link between them
- Three types of relationships are:
  - Association
  - Aggregation
  - Dependency

Relationships

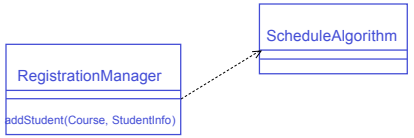
- An association is a bi-directional connection between classes
  - An association is shown as a line connecting the related classes

- An aggregation is a stronger form of relationship where the relationship is between a whole and its parts
- An aggregation is shown as a line connecting the related classes with a diamond next to the class representing the whole



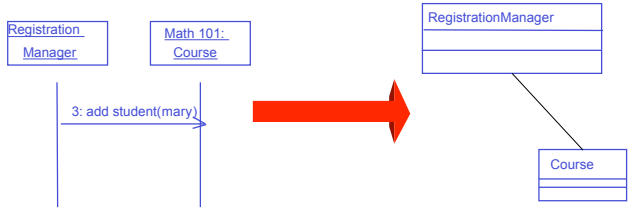
Relationships

- A dependency relationship is a weaker form of relationship showing a relationship between a client and a supplier where the client does not have semantic knowledge of the supplier
- A dependency is shown as a dashed line pointing from the client to the supplier

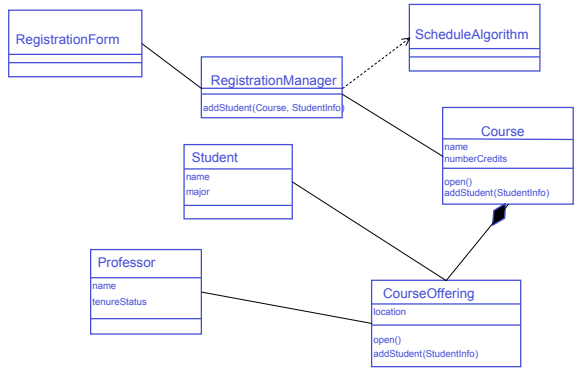


Finding Relationships

- Relationships are discovered by examining interaction diagrams
  - If two objects must "talk" there must be a pathway for communication



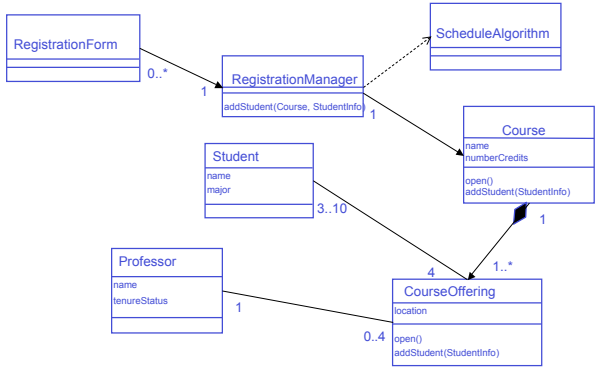
Relationships



Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationships
  - Multiplicity is the number of instances of one class related to ONE instance of the other class
  - For each association and aggregation, there are two multiplicity decisions to make: one for each end of the relationship
- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

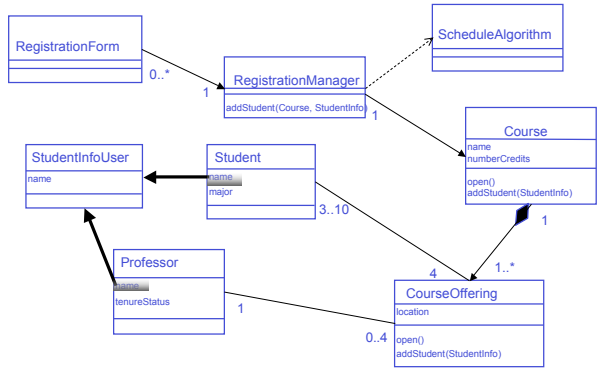
Multiplicity and Navigation



Inheritance

- Inheritance is a relationships between a superclass and its subclasses
- There are two ways to find inheritance:
  - Generalization
  - Specialization
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

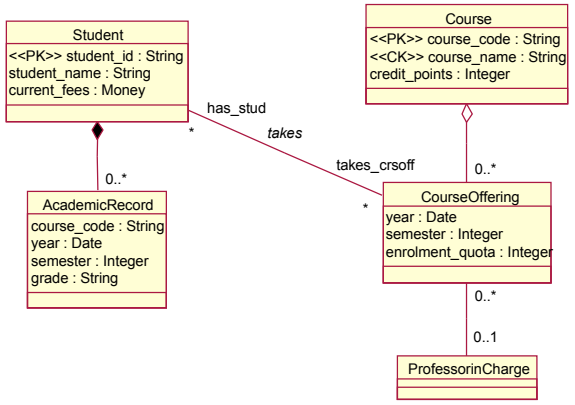
Inheritence



University Enrolment - Maciaszek

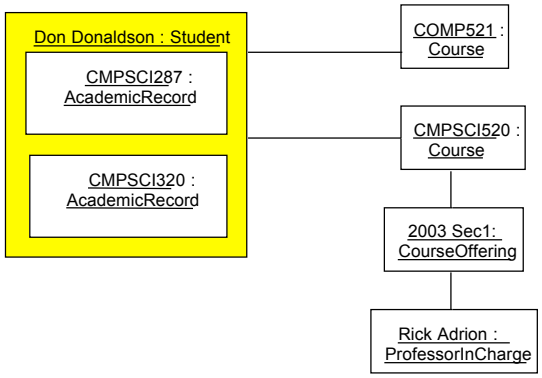
- A student's choice of courses may be restricted by timetable clashes and by limitations on the number of students who can be enrolled in the current course offering.
- A student's proposed program of study is entered on on-line enrolment system SPIRE
- The system checks the program's consistency and 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

Maciaszek Solution



Maciaszek Solution

- Show few objects representing the classes



Activity Modeling

- **Activity model**
  - Can graphically represent the flow of events of a use case
  - Can be used to understand a business process at a high-level of abstraction before any use cases are produced
- Shows the steps of a computation
  - Each step is a **state** of doing something
  - Execution steps are called **activity states**
  - Depicts which steps are executed in sequence and which can be executed concurrently
  - **Transition** – the flow of control from one activity state to the next
- **Use case descriptions** are (usually) written from an outside actor's perspective
- **Activity models** take an inside system's viewpoint

Activities

- **Activity states** can be established from the use case document
- **Activities** should be named from the system's perspective, not the actor's viewpoint
- **Activity** takes time to complete
- **Action** is so quick that – on our time scale – it is considered to take no time at all
- UML uses the same same graphical symbol for **activity state** and **action state** – rounded rectangle

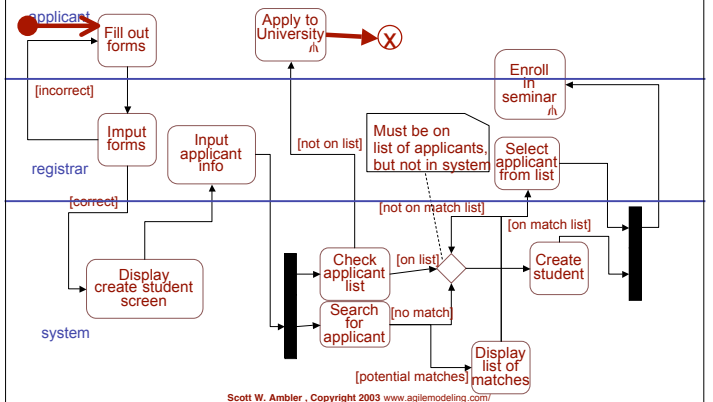
UML 2 Activity Diagrams

- object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development
- typically used for
  - business process modeling
  - modeling the logic captured by a single use case or usage scenario
  - modeling the detailed logic of a business rule
- could potentially model
  - the internal logic of a complex operation
  - far better to simply rewrite the operation so that it is simple enough that you don't require an activity diagram

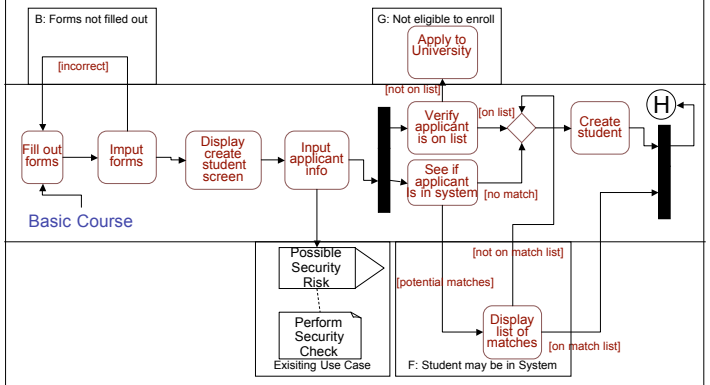
Activity Diagram Notation

- Initial node
  - filled circle = starting point of the diagram, not required
- Activity final node
  - filled circle with a border is the ending point, can have zero or more activity final nodes.
- Activity
  - rounded rectangles represent activities, may be physical or electronic
- Flow/edge
  - arrows on the diagram
- Fork
  - A black bar with one flow going into it and several leaving it, parallel activity.
- Join
  - A black bar with several flows entering it and one leaving it, denotes the end of parallel processing.
- Condition
  - a guard which must evaluate to true in order to traverse the node
- Decision
  - A diamond with one flow entering and several leaving
- Merge
  - A diamond with several flows entering and one leaving, all incoming flows must reach this point until processing continues, unless otherwise noted
- Partition
  - also called swimlanes, indicating who/what is performing the activities
- Sub-activity indicator
  - rake in the bottom corner of an activity indicates that the activity is described by a more finely detailed activity diagram.
- Flow final
  - circle with the X through it, process stops at this point.
- Use case
  - non-official? indication that an included use case is being invoked.

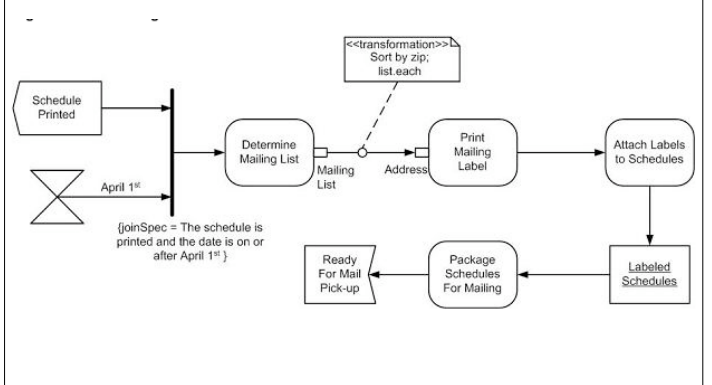
University enrollment example



University enrollment example



University enrollment example



When to Use Activity Diagrams

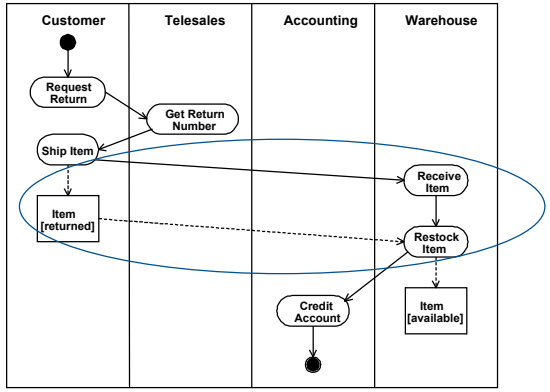
- Use activity diagrams when the behavior you are modeling ...
  - does not depend much on external events.
  - mostly has steps that run to completion, rather than being interrupted by events.
  - requires object/data flow between steps.
  - is being constructed at a stage when you are more concerned with which activities happen, rather than which objects are responsible for them (except partitions possibly).

Activity Diagram Modeling Tips

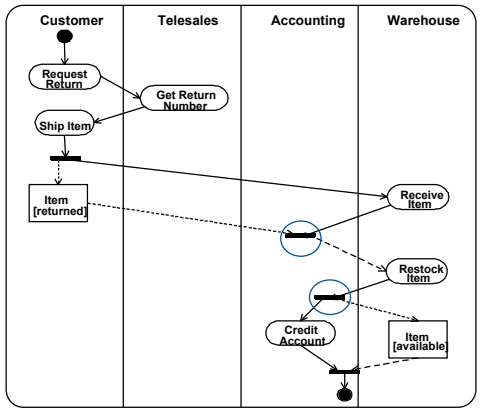
- Control flow and object flow are not separate. Both are modeled with state transitions.
- Dashed object flow lines are also control flow.
- You can mix state machine and control/object flow constructs on the same diagram (though you probably do not want to).

Activity Diagram Modeling Tips

From UML User Guide:



Activity Modeling Tips

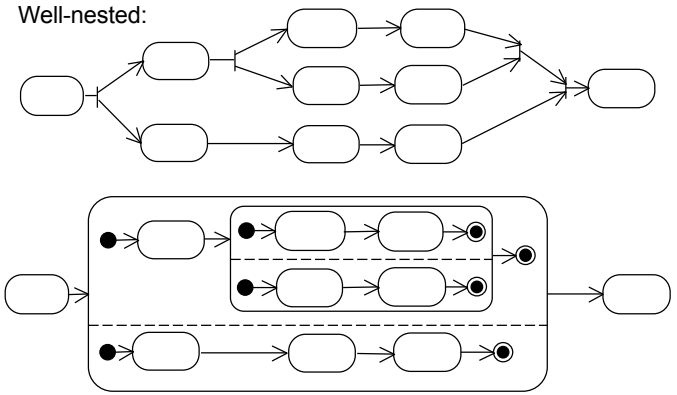


Activity Diagram Modeling Tips

- Activity diagrams inherit from state machines the requirement for well-structured nesting of composite states.
- This means you should either model as if composite states were there by matching all forks/decisions with a correspond join/merges …
- … or check that the diagram can be translated to one that is well-nested.
- This insures that diagram is executable under state machine semantics.

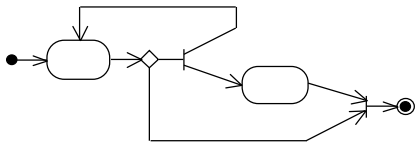
Activity Diagram Modeling Tips

Well-nested:



Activity Diagram Modeling Tips

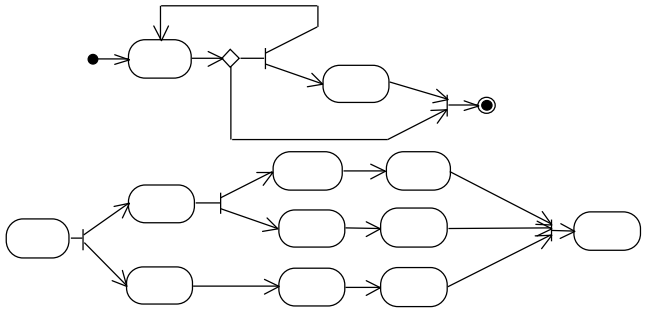
Not well-nested:



Apply structured coding principles. (Be careful with goto's!)

Activity Diagram Modeling Tips

Can be translated to well-nested diagram:



Wrap Up: Activity Diagrams

- Use Activity Diagrams for applications that are primarily control and data-driven, like business modeling …

… rather than event-driven applications like embedded systems.

- Activity diagrams are a kind of state machine until UML 2.0 …

… so control and object/data flow do not have separate semantics.

Statechart modeling

- Captures dynamic changes of class states – the life history of the class
- These dynamic changes describe typically the behavior of an object across several use cases
- State of an object – designated by the current values of the object's attributes
- Statechart Diagram – a bipartite graph of
  - states (rounded rectangles) and
  - transitions (arrows) caused by events
- The concepts of states and events are the same concepts that we know from Activity Diagrams – the difference is that "the states of the activity graph represent the states of executing the computation, not the states of an ordinary object"

Types of Events

- UML defines 4 kinds of events
  - Signal Event
    - Asynchronous signal received
    - e.g. evFlameOn
  - Call Event
    - operation call received
    - e.g. op(a,b,c)
  - Change Event
    - change in value occurred
  - Time Event
    - Relative time elapse
    - Absolute time arrived
    - e.g. tm(PulseWidthTime)

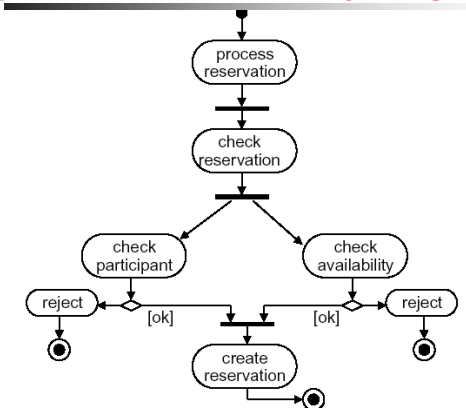
Types of Events

- Events are occurrences of interest that have both
  - Location
  - Absolute time of occurrence
- Signal events associate with Signals
  - A Signal is a specification of an asynchronous communication between structural elements (e.g. objects)
  - One type of Signal is Exception

States and transitions

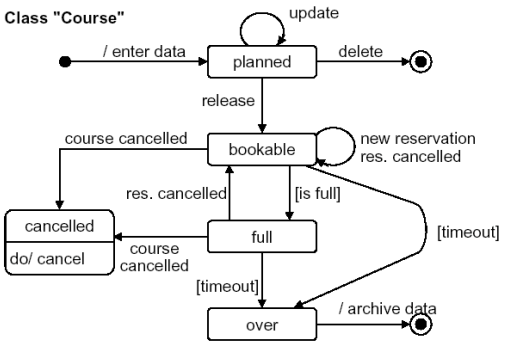
- Objects change values of their attributes but not all such changes cause state transitions
- We construct state models for classes that have interesting state changes, not any state changes
- Statechart Diagram is a model of business rules
  - Business rules are invariable over some periods of time
  - They are relatively independent of particular use cases

Statecharts & Activity Diagrams



Heinrich Hussmann

States and transitions



Heinrich Hussmann

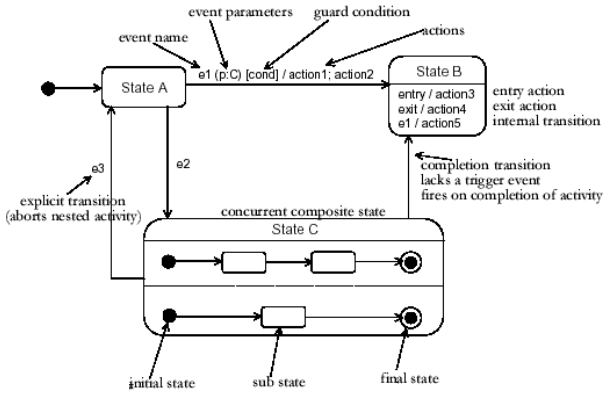
Statechart Diagram

- Normally attached to a class, but can be attached to other modeling concepts, e.g. a use case
- When attached to a class, the diagram determines how objects of that class react to events
  - Determines – for each object state – what **action** the object will perform when it receives an event
  - The same object may perform a different action for the same event depending on the object's state
  - The action's execution will typically cause a state change

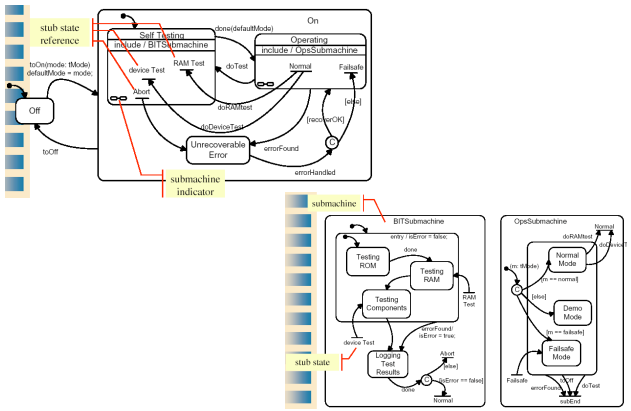
Statecharts

- Capture state-dependent requirements
- Statechart created for each state-dependent class
- UML provides hierarchical state transition diagrams
  - Based on Harel statecharts
- Information Captured
  - States
    - Capture all possible states of the class
    - Events and conditions
  - Describe transitions between states
    - Actions
  - Indicate processing that occurs on entry or exit to/from a state

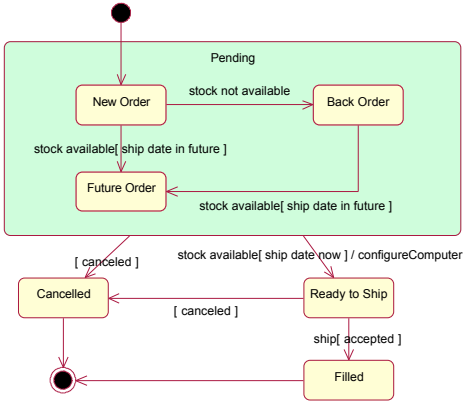
Statechart notation



Nested submachines

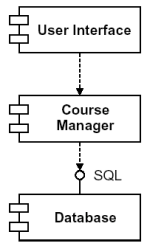


Statechart Diagram

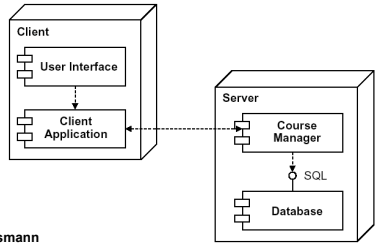


Implementation diagrams

Component Diagram



Deployment Diagram



Heinrich Hussmann