Software Development

Development Process

- Analysis
  - Requirement Analysis
    - Functional and Performance
  - Systems Analysis
    - Break-down between Hardware and Software
    - High Level Architecture
    - Control Scheme
  - Object Analysis
    - Object Structural Analysis
    - Object Behavioral Application

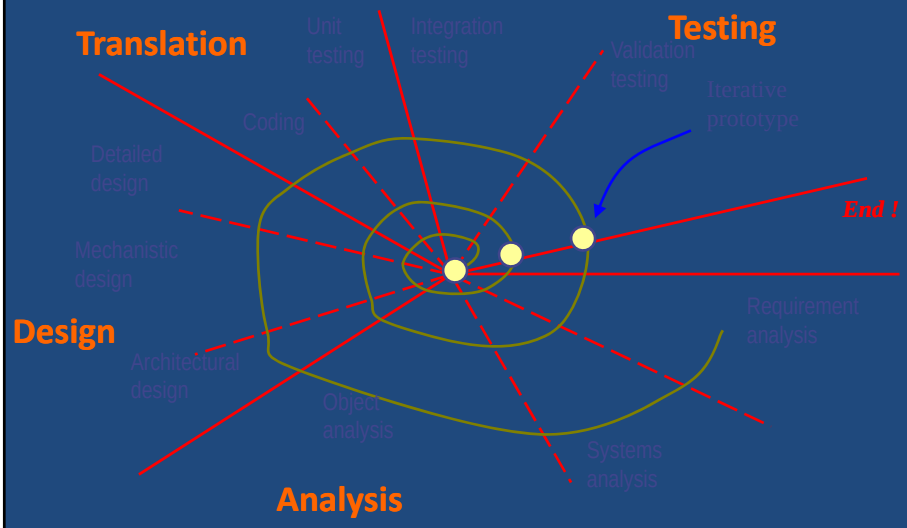
Development Process

- Design
  - Architectural
  - Mechanistic
  - Detailed Design
    - Optimize with low level information
- Translation
  - Creating an executable application
- Testing

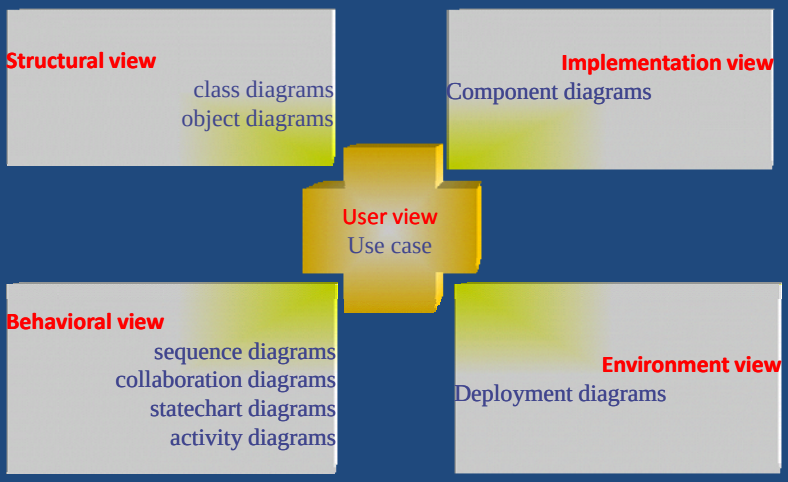
UML

- Object Model
- Use cases and Scenarios
- Behavioral model
- Representation of tasking and task synchronisation
- Models of physical topology

Iterative Rapid Development



Overview of UML



Visual Modelling and the UML

Basic Concepts

- A Class:
  - A class is a description used to instantiate objects
- An Object:
  - Is an instance of a class, it has a name, attributes and their values, and methods
  - An object models an idea found in reality, (tangible or abstract)

Basic Concepts (cont'd)

- Attributes
- Methods (Services, Messages)
- Information hiding and Encapsulation: A technique in which an object reveals as little as possible about its inner workings.
- Inheritance
- Polymorphism, Overloading, Templates

Object Oriented Analysis OOA

1. **Discovering Objects**
   - *The Data Perspective*
     - In the problem space or external systems
     - Physical devices
     - Events that need to be recorded (ex. Measurements)
     - Physical or geographical locations
     - Organizational units (departments , etc.)

OOA (cont'd)

- *The Functional Perspective*
  - What responsibilities does the object have?
    - » Ex. An event handler, a controller
- *The Behavioral Perspective*
  - Who does the object interact with? How?
  - Describe the object behavior

Identifying Objects

- An object may appear as a noun (ex. Measurement) or disguised in a verb (to measure)
- A method might appear as a verb (ex. Investigate) or disguised in a noun (investigation)
- Attributes describe some kind of characteristics for the object (adjectives). Attributes can be simple or complex. Complex attributes may lead to forming a new object. Attributes can also be nouns.

Object Types

- External Entities: Sensors, actuators, control panel, devices
- Information Items : Displays, Commands, etc.
- Entities which establishes the context of the problem : Controller, monitors, schedulers

OOA (cont'd)

2. Class Hierarchies

- Generalization
  - A manager, a commission worker --> Employee
- Specialization (IS_A)
  - Employee --> A commission worker

OOA (cont'd)

3. Relationships

- Types
  - Association
    - General form of dependency
  - Aggregation
    - An object may consist of other objects
  - Inheritance
- Cardinality (Multiplicity)
  - (Binary, Many, ..)

OOA (cont'd)

4. Object Attributes

- Discovering attributes and placing in class hierarchy
- Attribute types
  - Naming : Ex. SensorID, Account
  - Descriptive Ex. Card expiration date
  - Referential Ex. Referring to other objects

OOA (cont'd)

5. **Object Behavior**
   - Discovering states, changes in state, and conditions and actions
   - Building the state diagrams of objects

OOA (cont'd)

6. **Object Services**
   - Implicit Services (create, modify, search, delete , etc.) ex. constructors
   - Services associated with messages
   - Services associated with object relationships
   - Services associated with attributes (accessor methods ex. get, set . ..)

Object Oriented Design OOD

1. **Notation** (Unified Modeling Language)
   - Structural description (class diagrams)
   - Dynamics (Collaboration and interaction diagrams)
2. **Detailed Class and object description**
   - Visibility (Private, protected, ..)
   - Containment (ex. Packages or Components)
   - Concurrency

OOD (Cont'd)

3. Design Goodness Criteria

- Coupling:

The manner and degree of interdependence between classes (objects)

- Cohesion:

The degree and manner to which the tasks performed by an object are related to each other.

- Modularity
  - Understandability
  - Decomposability
- Clarity
- Simple classes, messages, methods

What is the UML?

- UML stands for Unified Modeling Language
- The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system
- It can be used with all processes, throughout the development life cycle, and across different implementation technologies.

UML Concepts

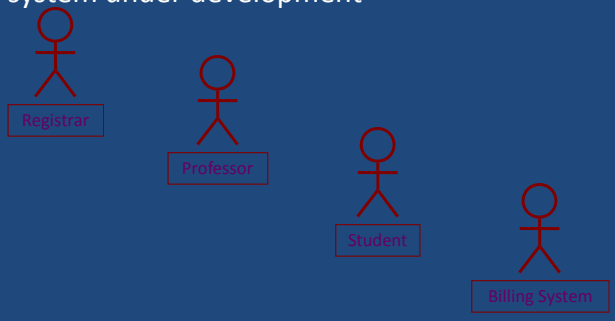
- The UML may be used to:
  - Display the boundary of a system & its major functions using use cases and actors
  - Illustrate use case realizations with interaction diagrams
  - Represent a static structure of a system using class diagrams
  - Model the behavior of objects with state transition diagrams
  - Reveal the physical implementation architecture with component & deployment diagrams
  - Extend your functionality with stereotypes

Putting the UML to Work

- 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

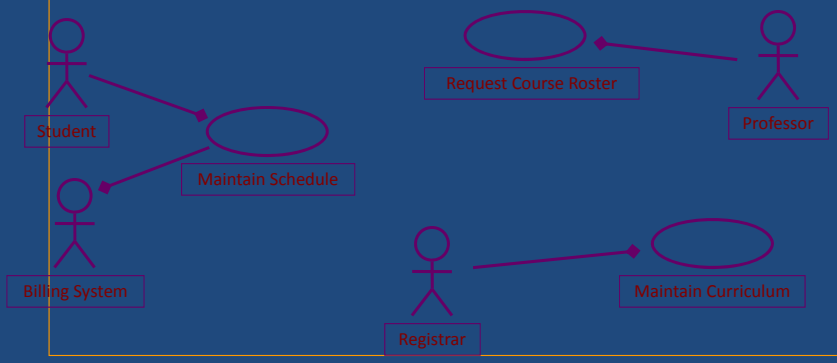
- A flow of events document is created for each use cases
  - Written from an actor point of view
- Details what the system must provide to the actor when the use cases is executed
- Typical contents
  - How the use case starts and ends
  - Normal flow of events
  - Alternate flow of events
  - Exceptional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

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.
  - ...

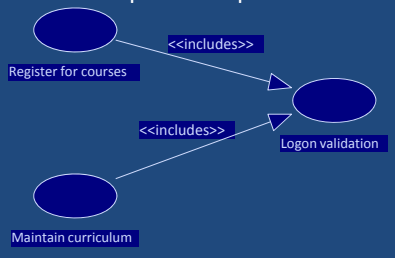
Use Case Diagram

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

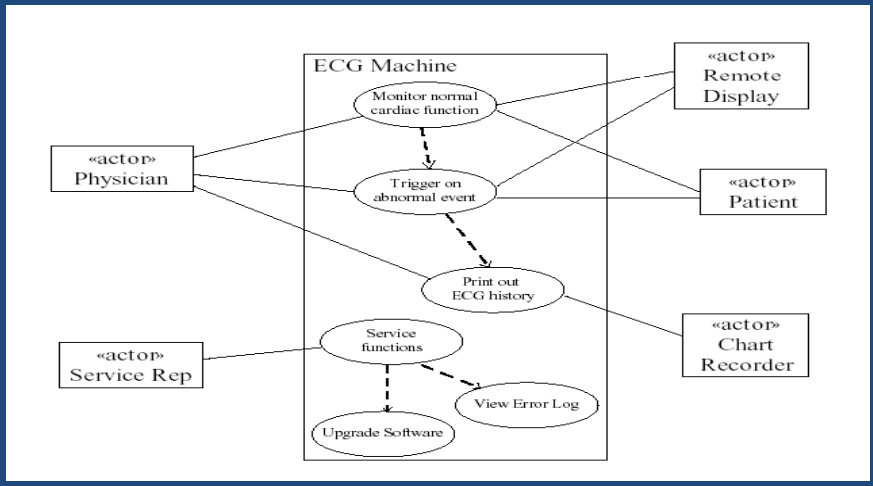


Uses and Extends Use Case Relationships

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



Example Use Case

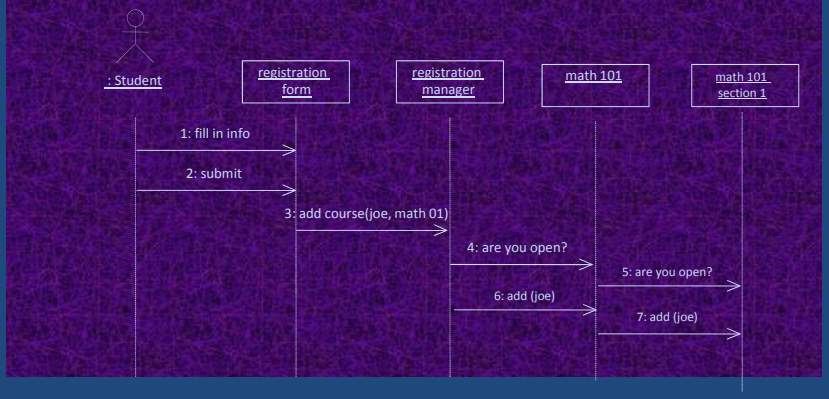


Use Case Realizations

- 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

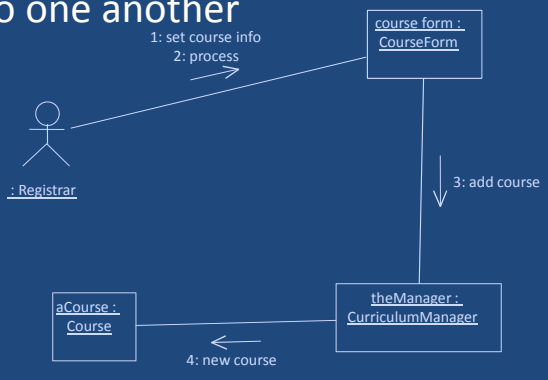
Sequence Diagram

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



Collaboration Diagram

- A collaboration diagram displays object interactions organized around objects and their links to one another



Class Diagrams

- A class diagram shows the existence of classes and their relationships in the logical view of a system
- UML modeling elements in class diagrams
  - Classes and their structure and behavior
  - Association, aggregation, and inheritance relationships
  - Multiplicity and navigation indicators
  - Role names

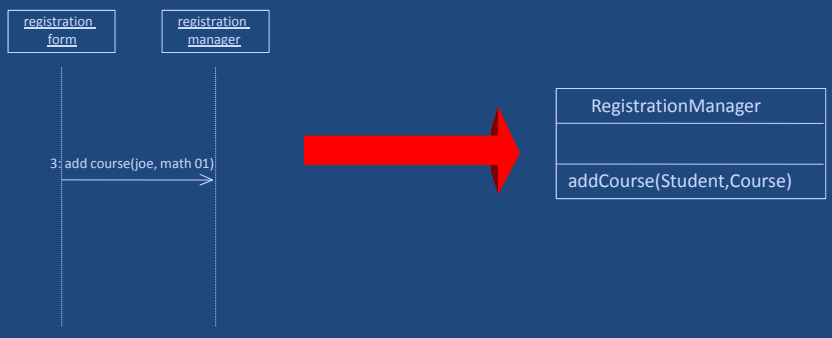
Classes

- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes are 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

Operations

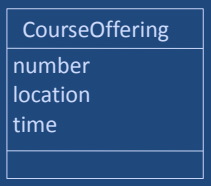
- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams



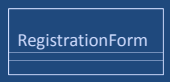
Attributes

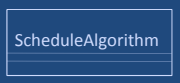
- The structure of a class is represented by its attributes
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge

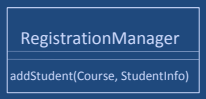
Each course offering has a number, location and time

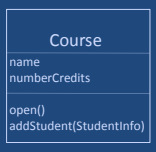


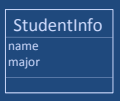
Classes

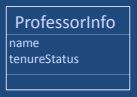


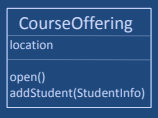




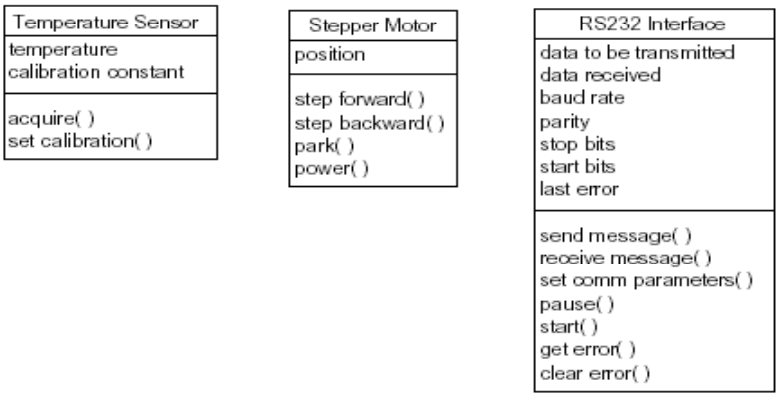








Example Classes



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
  - Inheritance

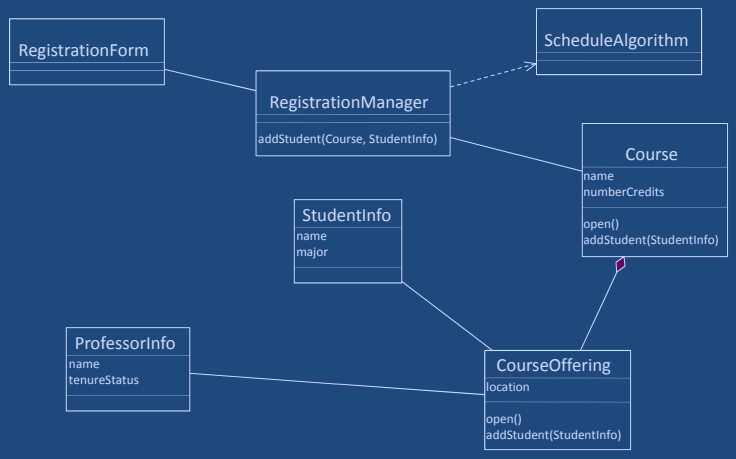
Relationships

- 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
- 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

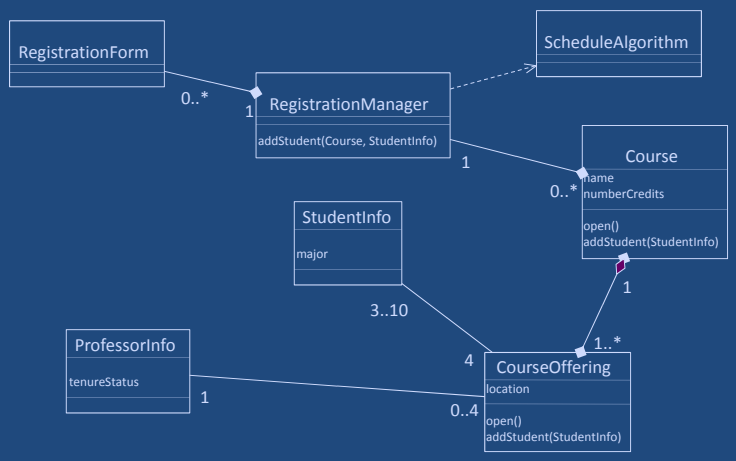
Relationships



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

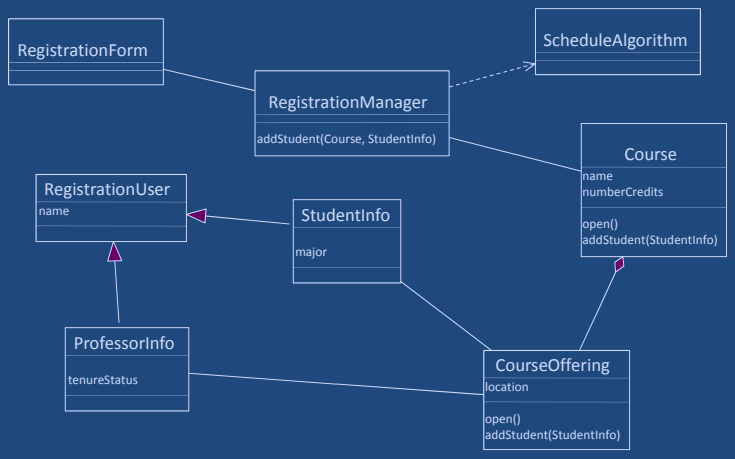
Multiplicity and Navigation



Inheritance

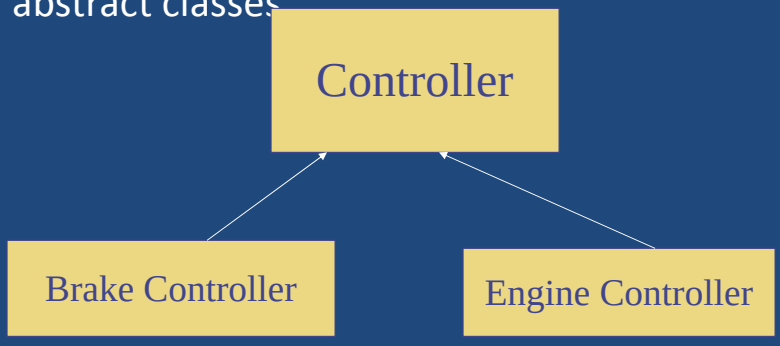
- Inheritance is a relationships between a superclass and its subclasses
- There are two ways to find inheritance:
  - Generalization
  - Specialization
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

Inheritance



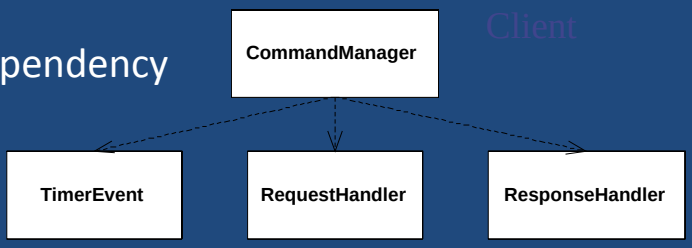
Generalization/Specialization Relation

- Controllers and Monitors are examples of abstract classes



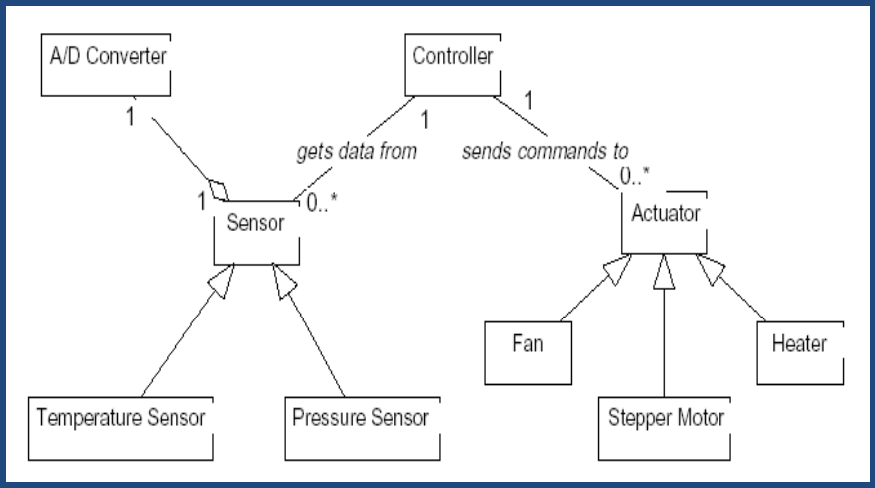
Dependency: A Special Case of Association

Dependency



CommandManager (Client class) depends on services provided by the other three server classes

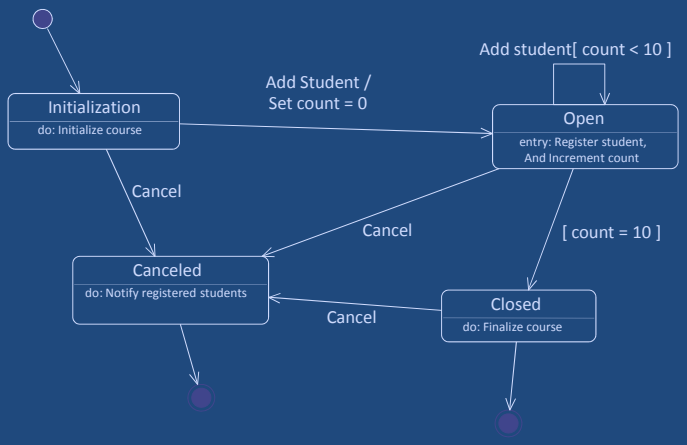
Simple Sensor Actuator Controller System Example



The State of an Object

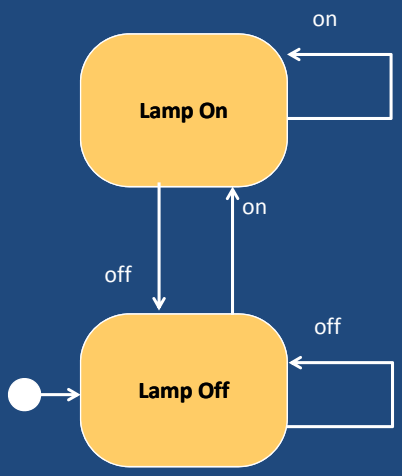
- A state transition diagram shows
  - The life history of a given class
  - The events that cause a transition from one state to another
  - The actions that result from a state change
- State transition diagrams are created for objects with significant dynamic behavior

State Transition Diagram



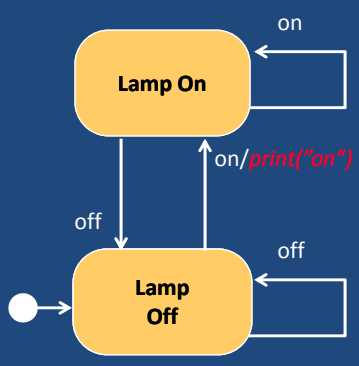
State Machine (Automaton) Diagram

- Graphical rendering of automata behavior

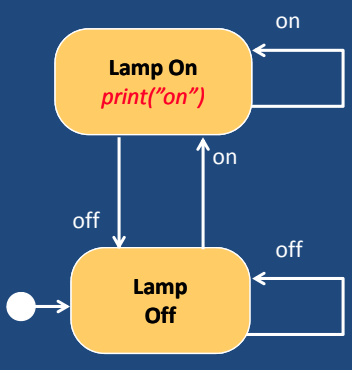


Outputs and Actions

- As the automaton changes state it can generate outputs:



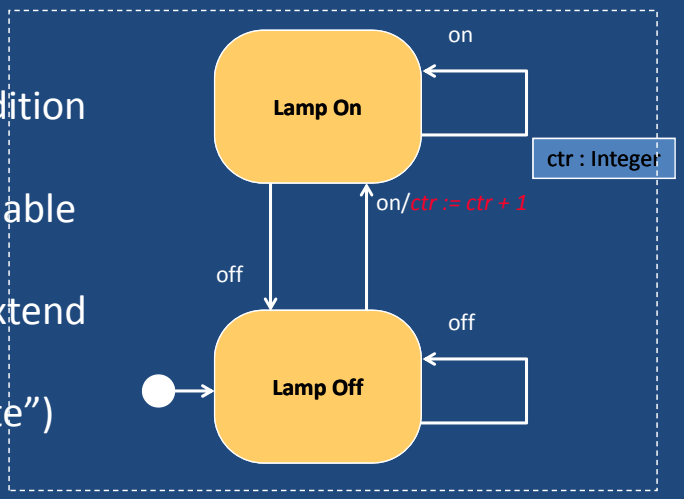
Mealy automaton



Moore automaton

Extended State Machines

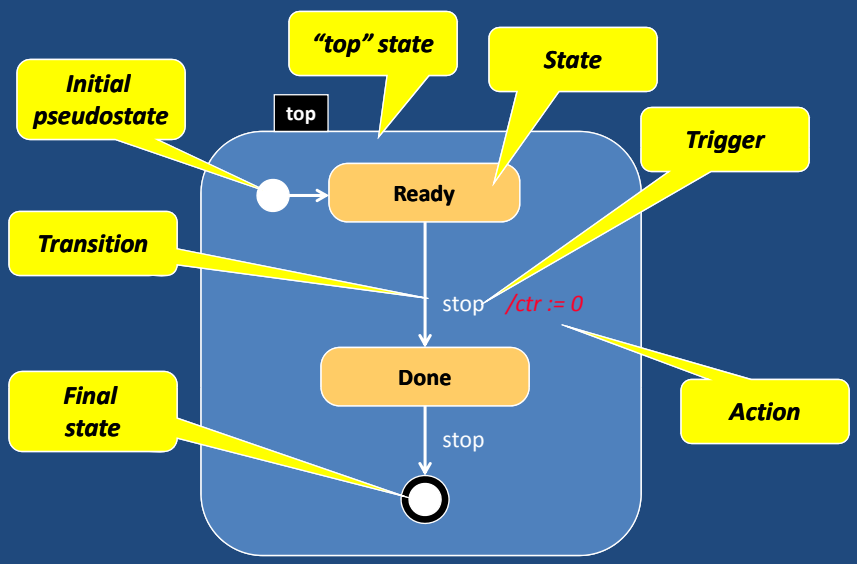
- Addition of variables (“extended state”)



A Bit of Theory

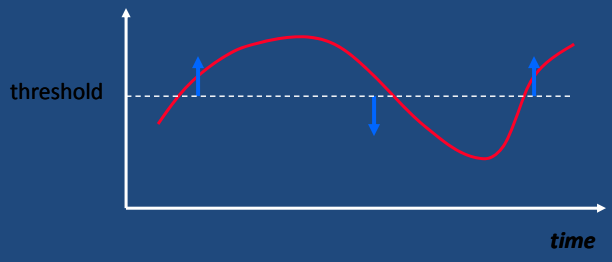
- An extended (Mealy) state machine is defined by:
  - a set of input signals (input alphabet)
  - a set of output signals (output alphabet)
  - a set of states
  - a set of transitions
    - triggering signal
    - action
  - a set of extended state variables
  - an initial state designation
  - a set of final states (if terminating automaton)

Basic UML Statechart Diagram



What Kind of Behavior?

- In general, state machines are suitable for describing event-driven, discrete behavior
  - inappropriate for modeling continuous behavior

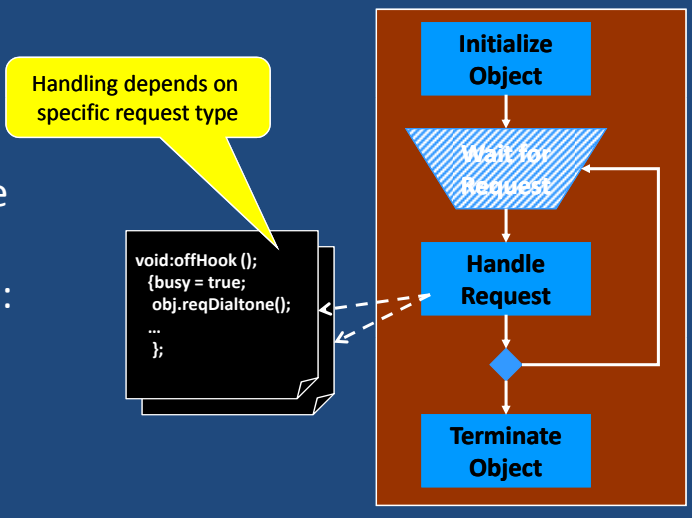


Event-Driven Behavior

- Event = a type of observable occurrence
  - interactions:
    - synchronous object operation invocation (call event)
    - asynchronous signal reception (signal event)
  - occurrence of time instants (time event)
    - interval expiry
    - calendar/clock time
  - change in value of some entity (change event)
- Event Instance = an instance of an event (type)
  - occurs at a particular time instant and has no duration

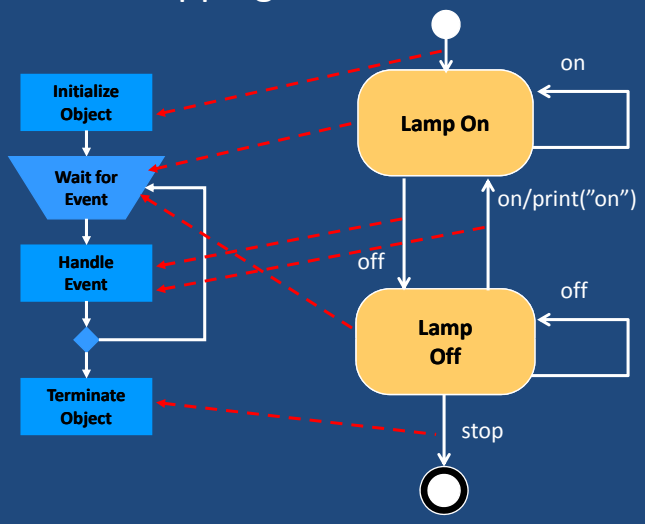
Object Behavior - General Model

- Simple server model:



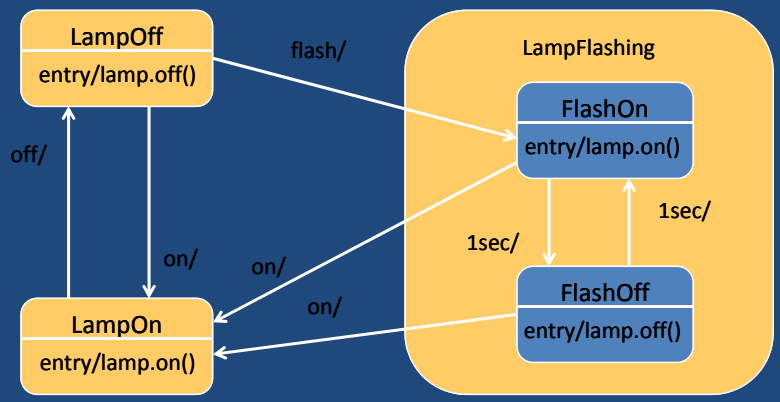
Object Behavior and State Machines

- Direct mapping:



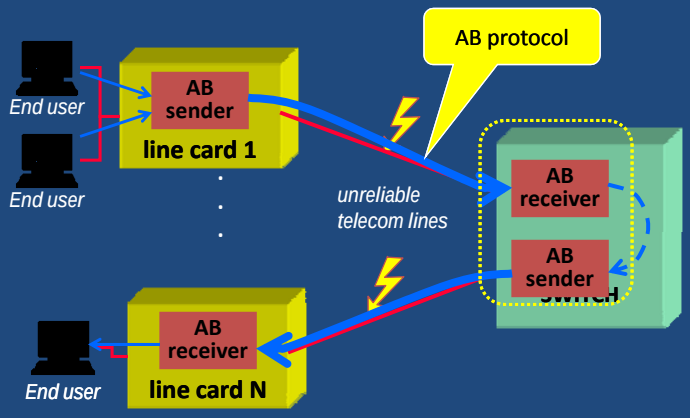
Hierarchical State Machines

- Graduated attack on complexity
  - states decomposed into state machines



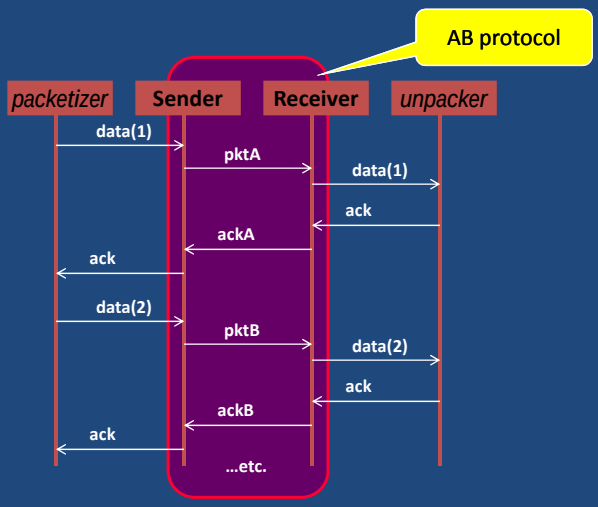
Case Study: Protocol Handler

- A multi-line packet switch that uses the alternating-bit protocol as its link protocol



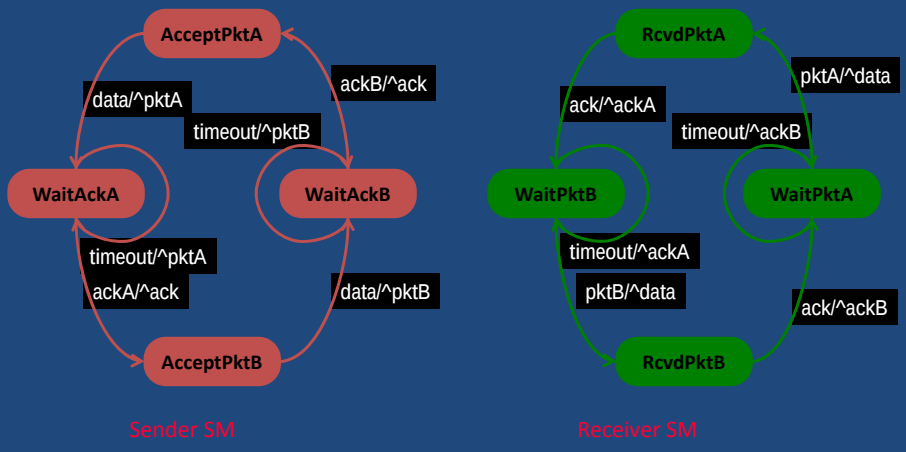
Alternating Bit Protocol (1)

- A simple one-way point-to-point packet protocol

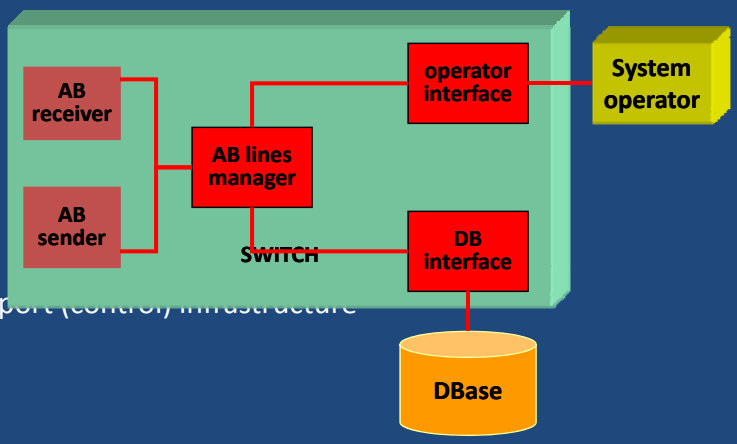


Alternating Bit Protocol (2)

- State machine specification



Additional Considerations



- Support (control) infrastructure

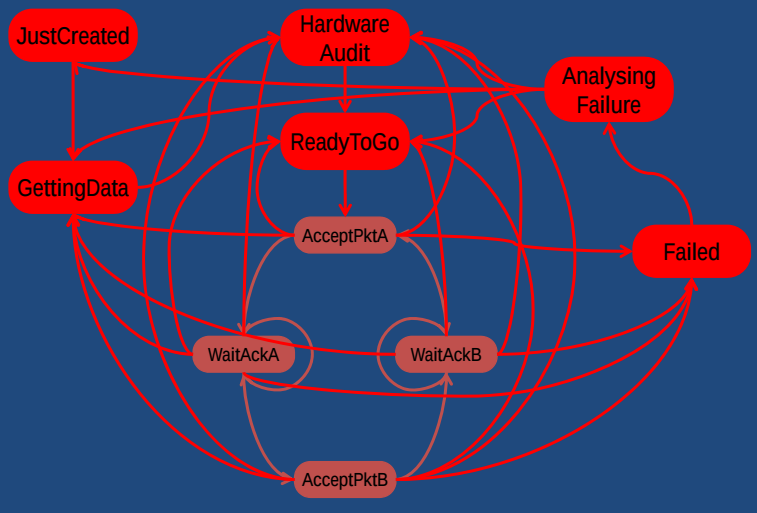
Control

The set of (additional) mechanisms and actions required to bring a system into the desired operational state and to maintain it in that state in the face of various planned and unplanned disruptions

For software systems this includes:

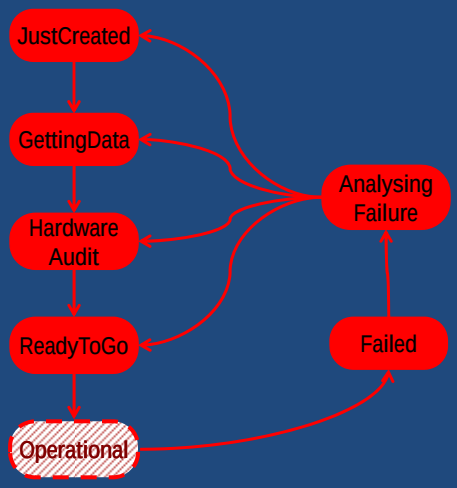
- system/component start-up and shut-down
- failure detection/reporting/recovery
- system administration, maintenance, and provisioning
- (on-line) software upgrade

Retrofitting Control Behavior



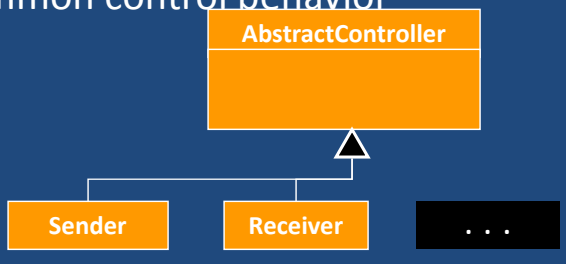
The Control Automaton

- In isolation, the same control behavior appears much simpler

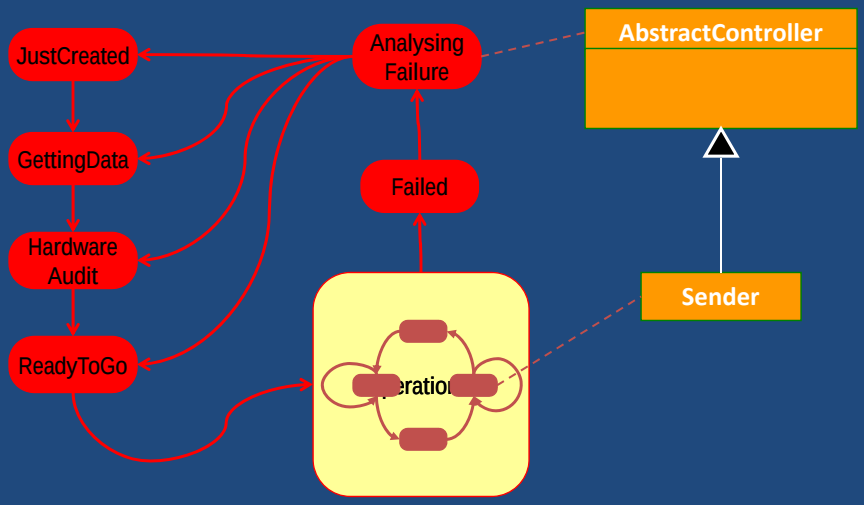


Exploiting Inheritance

- Abstract control classes can capture the common control behavior



Exploiting Hierarchical States

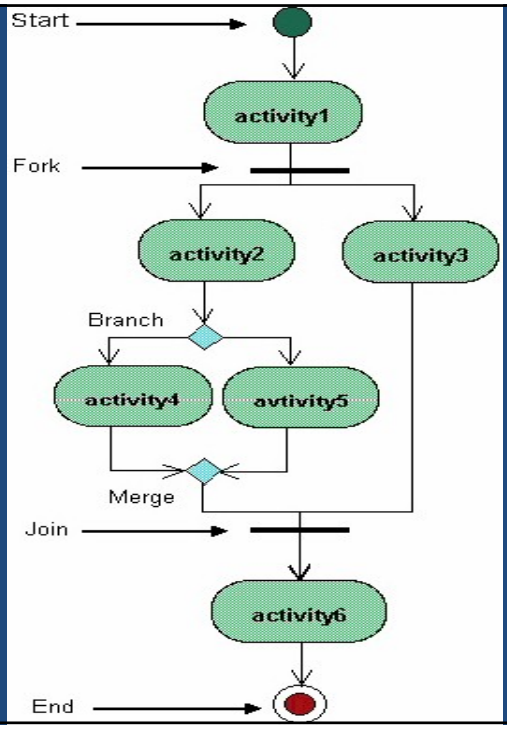


Activity Diagram

- Activity Diagram displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states.
- This diagram focuses on flows driven by internal processing.

Activity diagrams

- Activity diagrams show the flow of activities through the system.
- Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities.
- A fork is used when multiple activities are occurring at the same time (concurrently).
- A branch describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch.
- After merging all parallel activities must be combined by a join before transitioning into the final activity state.

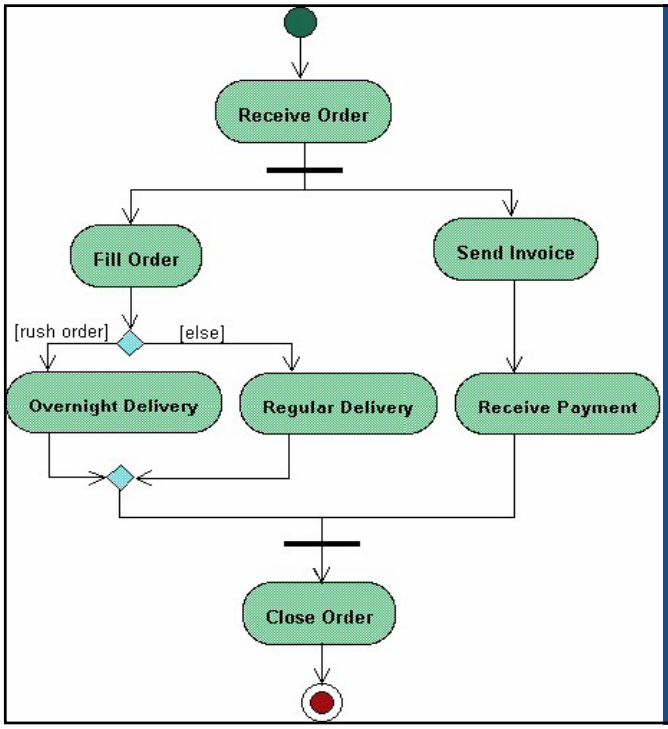


Both activity2 and activity3 are occurring at the same time.

After activity2 there is a branch.

The branch describes what

activities will take place based on a set of conditions.



Parallel activities - delivery and billing

Activity diagram